

โปรโตคอลการเดินทางชนิดมัลติคาสต์ที่ทนต่อความผิดพลาด
สำหรับการจราจรเวลาจริงในระบบอินเทอร์เน็ต



นายเดชา นุชิต กัตัญญทวิทิพย์

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2544

ISBN 974-03-0912-7

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

FAULT-TOLERANT MULTICAST ROUTING PROTOCOL
FOR REAL-TIME TRAFFIC ON THE INTERNET

Mr. Dechanuchit Katanyutaveetip

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in Computer Engineering
Department of Computer Engineering

Faculty of Engineering
Chulalongkorn University

Academic year 2001

ISBN 974-03-0912-7

Thesis Title Fault-tolerant Multicast Routing Protocol For Real-time Traffic
 on the Internet

By Dechanuchit Katanyutaveetip

Field of Study Computer Engineering

Thesis Advisor Thit Siriboon, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University
in Partial Fulfillment of the Requirements for the Doctor's Degree

..... Dean of the Faculty of Engineering
(Professor Somsak Panyakeow, D. Eng.)

THESIS COMMITTEE

..... Chairman
(Lecturer Arthit Thongtak, D. Eng.)

..... Thesis Advisor
(Lecturer Thit Siriboon, Ph.D.)

..... Member
(Associate Professor Somchai Prasitjutrakul, Ph.D.)

..... Member
(Lecturer Natawut Nupairoj, Ph.D.)

..... Member
(Wittaya Watcharawittayakul, Ph.D.)

เดชาลิขิต กตัญญูทวีทิพย์ : โปรโตคอลการเดินทางชนิดมัลติคาสต์ที่ทนต่อความผิดพลาด
สำหรับการจราจรเวลาจริงในระบบอินเทอร์เน็ต. (FAULT-TOLERANT MULTICAST
ROUTING PROTOCOL FOR REAL-TIME TRAFFIC ON THE INTERNET) อ. ที่ปรึกษา :
ดร. ลีต ศิริบุญรัตน์, 201 หน้า. ISBN 974-03-0912-7.

มัลติคาสต์เป็นวิธีการส่งข้อมูลในระบบเครือข่ายโดยข้อมูลจากผู้ส่งจะถูกกระจายไปยังผู้รับทั้งหมดพร้อม ๆ กัน วิธีการส่งข้อมูลแบบมัลติคาสต์จะเกิดขึ้นโดยสำเนาของข้อมูลเพียงสำเนาเดียวจะถูกส่งไปยังผู้รับหลาย ๆ ที่โดยพร้อมเพรียงกัน ข้อมูลจากผู้ส่งจะถูกส่งไปตามเส้นทางซึ่งอยู่ในรูปแบบ "ทรี" ซึ่งเรียกว่า "มัลติคาสต์ทรี" การส่งข้อมูลตามเงื่อนไขเวลาแบบระบบมัลติคาสต์นั้น ข้อมูลที่ผู้รับจะได้รับต้องอยู่ภายในเงื่อนไขเวลา ปัญหาของการส่งข้อมูลตามเงื่อนไขเวลาแบบมัลติคาสต์จึงเป็นปัญหาที่น่าสนใจและน่าศึกษา

จุดประสงค์แรกของวิทยานิพนธ์ ก็เพื่อศึกษาและพัฒนาโปรโตคอลของการส่งข้อมูลตามเงื่อนไขเวลาแบบมัลติคาสต์โดยพัฒนาจาก Core Based Tree ซึ่งเป็นการใช้มัลติคาสต์ทรีแบบต้นไม้ร่วมจุดประสงค์ที่สองของวิทยานิพนธ์ นี้ก็เพื่อศึกษาและพัฒนาการป้องกันการผิดพลาดของมัลติคาสต์โปรโตคอลตามเงื่อนไขเวลาดังกล่าว โดยใช้แนวคิดของการคำนวณหาเส้นทางสำรองของมัลติคาสต์ทรี และเมื่อเกิดการผิดพลาดในเส้นทางหลักเส้นทางสำรองจะถูกนำมาใช้แทนโดยข้อมูลมัลติคาสต์ยังสามารถไปถึงที่หมายภายในเงื่อนไขเวลาที่กำหนด

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา วิศวกรรมศาสตร์

สาขาวิชา วิศวกรรมคอมพิวเตอร์

ปีการศึกษา 2544

ลายมือชื่อนิสิต

ลายมือชื่ออาจารย์ที่ปรึกษา

ลายมือชื่ออาจารย์ที่ปรึกษาร่วม.....

4171816721 : MAJOR COMPUTER ENGINEERING

KEY WORD: FAULT-TOLERANT / REAL-TIME / CORE-BASED TREE / MULTICAST ROUTING

DECHANUCHIT KATANYUTAVEETIP : FAULT-TOLERANT MULTICAST ROUTING
 PROTOCOL FOR REAL-TIME TRAFFIC ON THE INTERNET. THESIS ADVISOR:

Thit Siriboon Ph.D., 201 pp. ISBN 974-03-0912-7.

Multicasting is a network-layer function that constructs paths along which data packets from a source are distributed to reach many, but not all, destinations in a communication network. Multicasting sends a single copy of a data packet simultaneously to multiple receivers over a communication link that is shared by the paths to the receivers. The sharing of links in the collection of the paths to receivers implicitly defines a tree used to distribute multicast packets. Real-time multicast is a multicast scheme in which messages should be received by all destinations within a specified delay bound. The interaction between multicasting and the delivery of applications with real-time delay requirements poses various new and interesting problems. Given the need to support group communications in real-time environments and to build efficient multicast communication, the first objective of this dissertation is to design a bounded-delay multicast algorithm while cost is our secondary concern. So it is to find an optimal multicast routing protocol such that the delay constraint criteria has to be met. We apply this concept to a shared tree multicast routing protocol, CBT (core based tree). The second objective of this dissertation is addressing the fault-tolerance aspect of our proposed real-time multicast protocol in addition to meeting the delay constraint values. A key issue here is to find a pre-computed disjoint backup paths for any two nodes in the multicast tree and use them as alternative paths when a single link failure is encountered. All disjoint backup paths must be checked for meeting real-time constraints before the rerouting of the paths take place.

Department of COMPUTER ENGINEERING Student's signature

Field of study COMPUTER ENGINEERING Advisor's signature

Academic year 2001 Co-advisor's signature

Acknowledgments

In writing this page I can at last say I have come to the end of a long road, I set out on 4 years ago, when I started this Ph.D. This dissertation was made possible by the help, support and guidance of several people. First and foremost, I would like to express my appreciation to my advisor Dr. Thit Siriboon for his support during this work. I would like to thank my other Ph.D. committee members, namely, Dr. Arthit Thongtak and Dr. Somchai Prasitjutrakul for various discussions which improved my research. I also would like to thank Dr. Natawut Nupairoj for his insightful comments and suggestions in doing Multicast research work. I am especially indebted to Dr. Wittaya Watcharawittayakul for providing me with valuable input and comments throughout my research. I am also thankful to Dr. Prabhas Chongstitwatana for pushing us hard enough, which eventually shapes all Ph.D. students and turned us to be disciplined researchers. Thanks also go to Guillermo Rigotti for helping me with ns-2 simulator and Dina Katabi of M.I.T. for sharing her expertise in anycasting.

I am deeply appreciated for my sister's and my brother's encouragement and support. Their energy and caring have been a constant source of inspiration for me. I would not have finished this dissertation without their support. I cannot find words strong enough to express my gratitude to my family. Their continuous support and supplications were crucial for the success of my studies. I am especially thankful for the love of my son and daughter. Their smiles and joy when I come home after a long day of work always make me forget my weariness. My wife was the one who stood with me and encouraged me throughout my studies. She was instrumental in my returning to my Ph.D. study. Life rarely offers a second chance, and I thank her for providing me this opportunity to pursue my study. I am grateful for the constant support and encouragement she had given me during these years. I deeply thank her for the sacrifices she made for me. I really owe this one to you. Finally, this dissertation is dedicated to my late parents whose love and support have always been the greatest inspiration for me in my pursuits for betterment. Without your sacrifices, love and discipline, none of this would have been possible.

Table of Contents

	Page
Abstract (Thai)	iv
Abstract (English)	v
Acknowledgments	vi
Table of Contents	vii
List of Tables	xii
List of Figures	xiii
CHAPTER 1	1
INTRODUCTION	1
1.1 Motivation	8
1.2 Problem Statement.....	10
1.3 Dissertation Objectives	11
1.4 Dissertation Contributions.....	13
1.5 Dissertation Outline	14
CHAPTER 2	16
MULTICAST ROUTING PROBLEM	16
2.1 Introduction	16
2.2 Multicast Routing.....	16
2.3 Multicast problem Definitions.....	20
2.4 Survey of Multicast Routing Algorithms	22
2.4.1 Source-specific Multicast trees	23
2.4.2 Shared Multicast trees.....	33
2.5 Fault-tolerant in Multicast Network.....	40
2.6 Conclusions.....	41
CHAPTER 3	43
CBT – THE PROTOCOL	43
3.1 Introduction	43
3.2 CBT Functional Overview.....	47

	Page
3.3 Protocol Specification Details	50
3.3.1 CBT HELLO Protocol	50
3.3.2 JOIN_REQUEST Processing	52
3.3.3 JOIN_ACK Processing	53
3.3.4 QUIT_NOTIFICATION Processing.....	54
3.3.5 ECHO_REQUEST Processing	55
3.3.6 ECHO_REPLY Processing.....	56
3.3.7 FLUSH_TREE Processing.....	57
3.4 Protocol Overview	58
3.4.1 CBT Group Initiation	58
3.4.2 Tree Joining Process	58
3.4.3 Tree Leaving Process.....	60
3.4.4 Tree Maintenance Issues	60
3.4.5 Core Placement	60
3.4.6 LAN Designated Router.....	61
3.4.7 Non-member Sending	62
3.4.8 Data Packet Forwarding.....	64
3.4.9 Lower Group Leave Latency	68
3.5 CBT Packet Formats and Message Types	69
3.5.1 CBT Control Packet Formats	74
3.5.2 Timers and Default Values.....	80
3.6 Core Router Discovery.....	81
3.6.1 "Bootstrap" Mechanism Overview	82
3.6.2 Core Failure	83
3.7 Node Failure.....	83
3.8 Loop Detection.....	84
3.8.1 Unicast Transient Loops.....	86
3.9 Conclusions.....	86

	Page
CHAPTER 4	89
REAL-TIME OPTIMAL MULTICAST ROUTING	89
Overview.....	89
4.1 Introduction	90
4.2 Previous Work	92
4.3 Multicast Routing Protocols	93
4.3.1 Source-Based Trees and Shared Trees.....	93
4.3.2 Real-time multicast Network Model.....	94
4.4 Weighted Dijkstra's Shortest Path Tree Algorithm.....	95
4.5 Residual delay path selection function.....	97
4.6 CBT modifications	99
4.7 Simulation.....	101
4.8 Performance evaluation	104
4.8.1 Average end-to-end delay.....	104
4.8.2 Network resource usage	107
4.8.3 Traffic concentration.....	109
4.8.4 Loss Rate	111
4.8.5 Execution Time	113
4.9 Conclusions.....	114
CHAPTER 5	116
FAULT-TOLERANT PROTOCOL	116
5.1 Introduction	116
5.1.1 Centralized vs. Distributed Schemes	117
5.1.2 Pre-computed vs. Dynamic (on-demand) computation of routes	118
5.1.3 Global vs. Local Knowledge Schemes	119
5.2 Failure Model.....	120
5.3 Fault-tolerant capability.....	120
5.3.1 Dispersion	121

Table of Contents

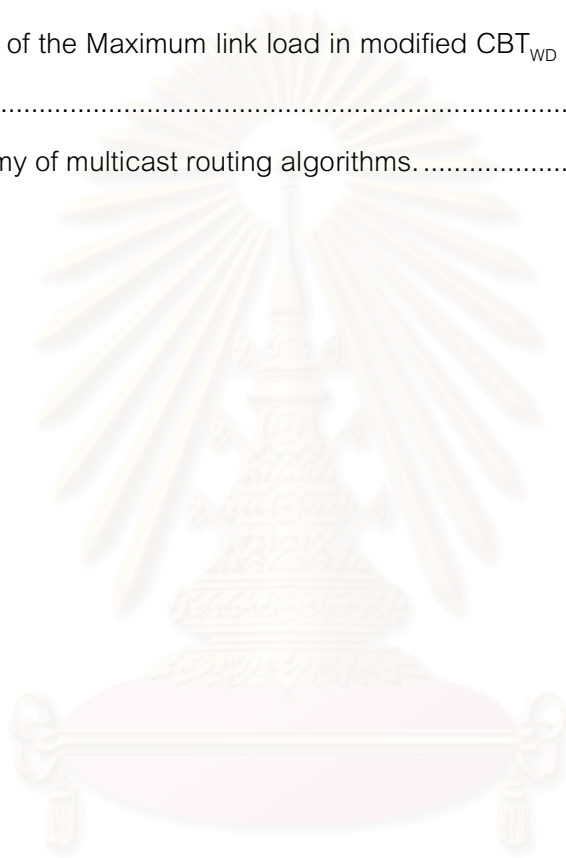
x

	Page
5.3.2 Redundancy	122
5.3.3 Disjointness.....	123
5.4 Fault-Tolerant Multicast Protocol	124
5.4.1 Initiation tasks	125
5.4.2 Fault Detection tasks	125
5.4.3 Backup path invocation tasks	126
5.4.4 Router Configuration tasks	126
5.5 Backup Path selections	127
5.5.1 Backup Core Router	127
5.5.2 Backup Paths with virtual repair.....	128
5.5.3 Backup Paths with real repair.....	128
5.6 Fault-tolerant in real-time communication	130
5.6.1 Dependable Network.....	132
5.6.2 DCM Routing Algorithm.....	132
5.7 Multiple Link faults.....	133
5.8 Conclusions.....	135
CHAPTER 6.....	137
FAULT-TOLERANT REAL-TIME MULTICAST ROUTING PROTOCOL.....	137
Overview.....	137
6.1 Introduction	138
6.2 Multicast Routing Algorithm	140
6.3 Real-time Applications	149
6.3.1 Real-time Multicast routing Model.....	149
6.4 Real-time optimal multicast routing	153
6.5 Fault-tolerant Multicast Protocol	155
6.5.1 Backup Paths with real-time constraints	156
6.6 Simulation results	157
6.7 Performance evaluation	160

	Page
6.7.1 Average end-to-end delay.....	160
6.7.2 Network resource usage	167
6.7.3 Traffic concentration.....	172
6.7.4 Loss Rate	177
6.7.5 Execution Time	180
6.7 Conclusions.....	182
CHAPTER 7.....	183
CONCLUSIONS AND FUTURE WORK.....	183
7.1 Future Work.....	185
7.2 Summary of Main Contributions.....	186
REFERENCES	188
Appendix	199
APPENDIX A : Journal Acceptance Letter.....	200
BIOGRAPHY.....	201

List of Tables

Table	Page
2.1: A taxonomy of multicast routing problems.....	20
2.2: Comparison of source and shared multicast tree router state.	34
4.1: Simulation environment.	103
4.2: The Ratio of the Maximum link load in modified CBT_{WD} and modified CBT_{RD} to that in CBT	110
6.1: A taxonomy of multicast routing algorithms.....	142



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

List of Figures

Figures	Page
2.1: Tree structures of solution approaches to the multicasting problems	22
2.2: Fault-tolerant protocol description	23
2.3: Multiple core-based trees structure	36
3.1: A CBT multicast delivery tree	45
3.2: Mesh of multicast delivery trees.....	46
3.3: A CBT FIB entry in the author's implementation	65
3.4: CBT Data Packet Forwarding on a LAN (originating case)	67
3.5: CBT Data Packet Forwarding on a LAN (receiving case)	68
3.6: CBT Header	70
3.7: CBT Control Packet Header	71
3.8: CBT Common Control Packet Header	75
3.9: HELLO Packet Format	76
3.10: JOIN_REQUEST Packet Format	76
3.11: JOIN_ACK Packet Format	77
3.12: QUIT_NOTIFICATION Packet Format	78
3.13: ECHO_REQUEST Packet Format	78
3.14: ECHO_REPLY Packet Format	79
3.15: FLUSH_TREE Packet Format	79
3.16: Bootstrap Message Format	80
3.17: Candidate Core Advertisement Message Format	80
4.1: Flow chart of our proposed protocol with two path selection functions	100
4.2: A randomly generated network, 20 nodes, average degree 4.....	102
4.3: Total network cost against end-to-end delay constraint, multicast group member = 50.....	105
4.4: Total network cost against end-to-end delay constraint, multicast group member = 100.....	106

Figures	Page
4.5: A Histogram of average delay in modified CBT_{WD} compared to its counterpart in CBT.	106
4.6: A Histogram of average delay in modified CBT_{RD} compared to its counterpart in CBT.	107
4.7: Network resource usage against end-to-end delay constraint values, multicast group member = 50.	108
4.8: Network resource usage against end-to-end delay constraint values, multicast group member = 100.	108
4.9: Network cost against number of multicast group size.....	109
4.10: Traffic Concentration in CBT.	110
4.11: Traffic Concentration in Modified CBT_{WD}	111
4.12: Traffic Concentration in Modified CBT_{RD}	111
4.13: Loss rate against end-to-end delay constraint values, multicast group member = 50.	112
4.14: Loss rate against end-to-end delay constraint values, multicast group member = 100.	113
4.15: Execution time against multicast group size.....	114
5.1: Tree structure of our fault-tolerant protocol.....	121
5.2: Reconfiguration method	129
5.3: Backup path selection for multiple faults tolerance.....	134
6.1: Flow chart of our proposed real-time multicast protocol	154
6.2: Flow chart of our proposed fault-tolerance protocol.....	155
6.3: Total cost of a multicast tree, multicast group member = 50	161
6.4: Total cost of a multicast tree, multicast group member = 100	161
6.5: A Histogram of the average end-to-end delay in modified CBT_{WD} compared to its counterpart in CBT.	162
6.6: A Histogram of the average end-to-end delay in modified CBT_{RD} compared to its counterpart in CBT.	162

Figures	Page
6.7: Total cost of a multicast tree, Pr (fault) = 0.1	163
6.8: Total cost of a multicast tree, Pr (fault) = 0.5	163
6.9: Total cost of a multicast tree relative to Jia's algorithm, multicast group size = 50, Delay constraint value = 100 ms	165
6.10: Total cost of a multicast tree relative to Jia's algorithm, multicast group size = 100, Delay constraint value = 100 ms	165
6.11: Total cost of a modified CBT_{WD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, at Pr (fault) = 0.1	166
6.12: Total cost of a modified CBT_{RD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, at Pr (fault) = 0.1	167
6.13: Total network resource of the modified CBT_{WD} , Pr (fault) = 0.1, Multicast group size = 50	168
6.14: Total network resource of the modified CBT_{RD} , Pr (fault) = 0.5, Multicast group size = 50	168
6.15: Total network resource of modified CBT_{WD} and modified CBT_{RD} relative to Jia's algorithm, multicast group size = 50, Delay constraint value = 100 ms	169
6.16: Total network resource of modified CBT_{WD} and modified CBT_{RD} relative to Jia's algorithm, multicast group size = 100, Delay constraint value = 100 ms	170
6.17: Total network resource of a modified CBT_{WD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, Pr(fault) = 0.1	171
6.18: Total network resource of a modified CBT_{RD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, Pr(fault) = 0.1	171
6.19: Traffic concentration in CBT, Pr (fault) = 0.1	172
6.20: Traffic concentration in modified CBT_{WD} , Pr (fault) = 0.1	173
6.21: Traffic concentration in modified CBT_{RD} , Pr (fault) = 0.1	173
6.22: Traffic concentration in CBT, Pr (fault) = 0.5	174
6.23: Traffic concentration in modified CBT_{WD} , Pr (fault) = 0.5	174
6.24: Traffic concentration in modified CBT_{RD} , Pr (fault) = 0.5	175

Figures	Page
6.25: Relative Maximum link load between CBT and modified CBT _{WD} with multicast group size = 20, 50 and 100	176
6.26: Relative Maximum link load between CBT and modified CBT _{RD} with multicast group size = 20, 50 and 100	176
6.27: Total loss rate relative to Jia's algorithm, Pr (fault) = 0.1, multicast group size = 50	177
6.28: Total loss rate relative to Jia's algorithm, Pr (fault) = 0.5, multicast group size = 50	178
6.29: Total loss rate of a modified CBT _{WD} multicast tree relative to Jia's algorithm with multicast group size = 20, 50 and 100, end-to-end delay = 100 ms.	179
6.30: Total loss rate of a modified CBT _{RD} multicast tree relative to Jia's algorithm with multicast group size = 20, 50 and 100, end-to-end delay = 100 ms.	179
6.31: Execution Time, Variable multicast group size, Pr (fault) = 0.1	181
6.32: Execution Time, Variable multicast group size, Pr (fault) = 0.5	181

CHAPTER 1

INTRODUCTION

Multicasting refers to the transmission of data from one node (source node) to a selected group of nodes (member nodes or destination nodes) in a communication network. Instead of sending a separate copy of the data to each individual group member, a multicast source sends a single copy to all the members. An underlying multicast routing algorithm determines, with respect to certain optimization objective, a multicast tree connecting the source(s) and the group members. Data generated by the source flows through the multicast tree, traversing each tree edge exactly once. As a result, multicast is more resource efficient, and is well suited for applications such as video distribution. Multicast services have been increasingly used by various continuous media applications. For example, the multicast backbone (Mbone) of the Internet has been used to transport real-time audio and video for news, entertainment, and distance learning. With fast development of hardware technologies, commercialization of the Internet, as well as the increasing demand of quality-of-services (QoS) fueled by emerging continuous media applications, offering guaranteed and better than best effort services will add to the competitive edge of a successful service provider. The notion of QoS was proposed to capture the qualitatively or quantitatively defined performance contract between the service provider and the user applications. QoS provisioning entails the development of several essential techniques, i.e. definition and specification of QoS, design of QoS-driven (or termed elsewhere constrained-based) unicast/multicast routing protocols, packet scheduling algorithms for link sharing, as well as resource reservation and management.

In general, a multicast communication session involves multiple sources transmitting to multiple destinations. This is the many-to-many multicasting problem. Videoconferencing is an obvious example of an application involving multiple sources and multiple receivers. The one-to-many multicasting problem is a special case of the many-to-many problem, in which the multicast session involves only one source. An

example of one-to-many communication is real-time control application in which a sensor transmits its readings to more than one remote control stations. One approach to establish a many-to-many communication session is by setting up multiple one-to-many sessions. The host group model is defined as a multicast group that is a set of receivers, identified by a unique group address. The use of a unique group address allows logical addressing, i.e., a source needs only to know the group address in order to reach all receivers. It does not need to know the addresses of the individual receivers. In addition, the source itself need not be a member of the multicast group. Logical addressing is advantageous for applications with large numbers of sources and receivers, such as mailing lists or news groups, and for dynamic applications, such as computer-supported cooperative work, where receivers may join or leave the group at any time and sources may start or stop transmission to that group at any time.

Most real-time applications of computer networks, such as teleconferencing, remote collaboration and distance learning, rely on the ability of the network to provide multicast communication. These applications may require end-to-end delays, delay jitter, and loss rate which are expressed as QoS parameters which must be guaranteed by the underlying network. The upper bound on end-to-end delay from any source to any receiver in a real-time session is the main QoS parameter we consider during our investigation in this dissertation. In high-speed wide-area networks, the transmission delay is small and the queuing delay is also small, because small buffer sizes are used. Therefore, the propagation delay is the dominant component of the link delay. The propagation delay is proportional to the distance traversed by the link. It is fixed, irrespective of the link utilization. Therefore a route selection algorithm can guarantee an upper bound on the end-to-end delay by choosing the appropriated links for the session being initiated, such that the delay from any source to any receiver does not exceed the delay bound.

With the advent of multimedia applications, the focus of the communication network has shifted from initial emphasis on reliable delivery of text to include support for QoS requirements of continuous media data. Complex control algorithms and sophisticated routing, scheduling, and resource allocation mechanisms are needed to support the bandwidth and QoS requirements in a context where the network

transmission speeds are increasing at a rate faster than the processing speed. Thus, the network control algorithms and protocols must be sufficiently simple to operate at or near link bandwidth speed, but be flexible enough to support applications with different QoS requirements.

The “best-effort paradigm” offered by the current datagram service, which proved to be very successful in the realization of a universal network in a heterogeneous environment, is not adequate to support real-time traffic. The network protocol offers no guarantees about timely, reliable, and ordered delivery of packets. Circuit switching, which is the standard method for providing real-time performance, does not optimize the utilization of the network resources, and may be very inadequate for variable bit-rate applications.

To address real-time requirements for multimedia applications in an efficient manner, the development of new network channel abstractions is required. It is clear however, that without proper characterization of the network components and without any resource allocation, it is extremely difficult, if not impossible, to provide predictable performance guarantees to multimedia applications. Two approaches can be used to provide network performance guarantees. The first approach is to overengineer the network to the extent that an application is certain to get the resource it needs. Thus, the applications can be unconstrained in their resource usages and still receive the guaranteed level of performance. In a high-speed network, this approach would require a prohibitively large amount of resources.

The second approach involves monitoring and controlling resource allocation and usage for each application. This approach requires that an application specifies its resource needs *a priori*, and unless its needs can be met, it is blocked or rejected. Once started, mechanisms are provided to ensure that the application does not use more resources than requested. Because every application uses only its share resources, all performance needs can be met. This approach has been adopted in several frameworks of integrated service packet networks. The service models described in the integrated service network focus on providing either predictive or guaranteed service for real-time applications on a one-to-one basis.

In order to support applications' real-time requirements, the network architecture should provide a basic framework upon which applications with widely varying traffic requirements can receive satisfactory service. Building such an architecture requires investigation of multiple design issues. These issues can be divided as follows:

- **Flow Specification:** The network and the applications require a common language to specify the requirements for transmitting real-time data. A source transmitting real-time data needs to inform the network of its QoS requirements. These requirements are communicated to the network nodes along the path to verify the feasibility of supporting these requirements.
- **Admission Control:** Since network resources are finite, the network cannot accept all connection requests while maintaining QoS requirements. The network has to verify that there are enough resources before granting any request. However, when the application does not operate at its peak, expensive network resources can be underutilized.
- **Path Establishment and Routing:** The network needs to decide which route to pick to send the packets of a session. Selecting a route can be harder for real-time communications where there are some requirements on the Quality of Service (QoS) that have to be satisfied.
- **Policing:** In order to verify that applications conform to their specified rates, a policing mechanism is needed at the edge of the network. By preventing an offending application from excessively utilizing the network resources, the likelihood that this application negatively affects other applications' QoS is greatly reduced.
- **Scheduling:** Scheduling policies specify how resources are allocated to incoming packets as they arrive at intermediate nodes. A scheduling policy determines when a packet will be transmitted. Depending on the scheduling policy, different guarantees can be offered. A scheduling policy can offer guarantees in terms of bandwidth, delay or both. The types of QoS guarantees provided by a scheduling policy are classified as *deterministic* or *statistical*. A deterministic guarantee policy ensures all application performance requirements will be met barring

software or hardware failures. In contrast, a statistical guarantee policy provides a probabilistic guarantee for the application traffic. In other words, given an infinite time, the statistical guarantee policy ensures that the performance requirements of a specified percentage of the applications packets will be met on average over the time.

In the current Internet, fault-tolerant or survivable capability was already incorporated. Many of the design decisions were taken to ensure that the network would survive the failure of multiple routing and transmission elements. To increase the system reliability, additional resources must be reserved as a *priori*, such as multiple copy schemes, dispersity routing [7] and spare resource allocation schemes. The major research issue with these schemes seems to be the reduction of the resource overhead by means of techniques such as sharing of resources across connections or between disjoint paths for the same connection. This also points to the main difference between such proactive schemes, and the reactive ones. With reactive schemes, there is no resource overhead in the common (no failure) case. Only when a failure occurs does the network attempt to find the resources necessary to recover from the failure. The inevitable consequence of this “laziness” is the latency of recovery, and the possibility of being unable to recover if the network is too highly loaded. We imagine that connections with a very high requirement for reliability will be set up with a priori guarantees on reliability, while most of the other connections will be satisfied with quick recovery for most of the traffic.

With the phenomenal growth of the Internet, a wide array of network applications are being developed for use over the Internet. More and more of these applications involve transmission of multimedia information and use multipoint connections. The popularity and availability of these multicast applications has lead to a phenomenal growth of the multicast backbone or Mbone. Multimedia applications are bandwidth intensive and real-time in nature on which they impose a greater load to the internetwork.

Multipoint communication can be implemented as a set of one-to-one communication which requires the sender to send an individual copy to each

destination sequentially. The increase in the number of destinations in the multicast increases the load on the wide-area network because copies of packets must be replicated to send to each destination.

With multicast approach, resource usage is improved whereby a multicast tree is constructed and used as distribution paths from a source to multiple receivers. Duplication of information is only performed at the forks in the tree defined for the multicast. Sending only a single copy saves bandwidth on links and reduces congestion in the network layer. The information is sent in parallel to the receivers along the branches of the tree, improving the average latency from the source to the receivers.

Algorithms are needed in the network to compute multicast trees; we call such algorithms, multicast algorithms. A multicast algorithm constructs a multicast tree by setting up state in the multicast routing tables of the routers in the computer network. These tables maintain pointers to neighbors to which multicast packets are forwarded in order to reach their destinations. Multicast routing can operate either in a distributed or centralized manner to compute the multicast tree. It is not desirable or feasible to have a central control in a very large network. In the centralized method, there is a high communication cost and single point of failure. It may be too expensive in terms of bandwidth, memory and processing overhead for a central control to have all the information about the group memberships and carry out all the computation to construct the multicast tree. Then the central control has to broadcast the information about the tree to all the routers in the network reliably.

Multicasting has been extended to operate in the wide-area networks by having the Internet Group Message Protocol (IGMP), to disseminate multicast membership information to multicast routers and permits routers to dynamically determine how to forward messages. A delivery tree is constructed on-demand and is data-driven. The tree in the existing IP architecture is the reverse shortest-path tree and shortest-path tree from the source to the group for distance-vector, namely DVMRP (Distance Vector Multicast Routing Protocol) and link-state routing, namely MOSPF (Multicast Extensions to Open Shortest Path First), respectively. However, there are several shortcomings with the existing IP multicast architecture, i.e. DVMRP and MOSPF. First, all routers in the Internet have to generate and process periodically control messages for every multicast

group, regardless of whether or not they belong to the multicast tree of the group. Thus, routers not on the multicast tree incur memory and processing overhead to construct and maintain the tree for the lifetime of the group. Packets that do not lead to any receivers or sources are periodically flooded throughout the Internet, thereby consuming and wasting bandwidth. In DVMRP, it is the data packets that are periodically flooded when the state information for a multicast tree times out. In MOSPF, it is the link-state packets, containing the state information for group membership, that are periodically flooded. Second, the multicast routing information in each router is stored for each source sending to a group. Finally, the IP multicast protocols, being extensions of unicast routings such as the DVMRP and MOSPF, are tightly coupled to the underlying unicast routing algorithm. This complicates inter-domain multicasting if the domains involved use different unicast routing. The unicast routing also becomes more complicated by incorporating the multicast-related requirements.

Among many multicast algorithms, the core-based tree (CBT) method has received a great deal of attention. The CBT was proposed to overcome the above shortcomings [5]. CBT protocol involves having a single node, known as the core of the tree, from which the branches stretch. These branches form the shortest paths between the members of the multicast group and the core. CBT also allows multiple Core routers to be specified which adds a little redundancy in case the core becomes unreachable. However, the core-based tree method may have a reliability problem. A single point of failure on the tree will partition the tree and make it difficult, to fulfill the requirement of multicasting. Core selection method for multicast routing has a significant impact on performance characteristics such as delay, bandwidth and traffic concentration.

Real-time multicast is a type of multicast which requires that messages be received by all destinations within a delay bound. There are many network applications relying on real-time multicast services, such as interactive voice or video conferencing systems, real-time control and monitoring systems, and so on. To operate real-time multicast on CBT, the disadvantages are its poor performance [108] in terms of delay, compared with other multicast protocol such as DVMRP and its reliability in case of faulty core router. However, CBT allows us to significantly improve the overall scaling factor of $S \times N$ we have in the source-based tree to just N , where S is the number of

active sources per multicast group, and N is the number of multicast groups present, which is the result of having just one multicast tree per group as opposed to one tree per (source, group) pair. We believe that Core Based Trees has the potential to make more efficient use of resources, and scaling over a number of hosts, provided that time delay from source to group destinations are bounded. The key issue is to model the traffic on CBT, the shared multicast tree, so that a delay bound can be derived. In this dissertation, we attempt to enhance the CBT protocol with fault-tolerant capability while improving its performance, without violating the end-to-end packet delay and minimizes the resource consumption of the network.

1.1 Motivation

The Internet has proven to be highly successful in supporting elastic applications, which adapt to varying delays and packet loss. Much of this success arises from the Internet's use of the datagram as a building block, over which other services including end-to-end reliability are built. At the heart of the Internet are routers that implement best-effort service; routers do not guarantee delivery or performance. Recently, the Internet has begun using multicast delivery to support group communications. Multicast delivers packets from a sender to a group of receivers over multicast tree. The primary advantage that multicast has over traditional unicast delivery is that the sender transmits a single packet to reach all of the group members, rather than sending a separate copy to each receiver. Replication of each packet is handled by the network and is done only when necessary, i.e. at the branching points in the multicast tree. Furthermore, the group model used by the Internet is receiver-oriented; receivers may join a group independently (i.e. senders do not control membership), and senders do not need to know the identities of group members. By avoiding the potential bottleneck at the sender, dynamic multicast applications may grow to encompass very large groups of receivers.

Another recent development has been the increasing use of real-time applications in the Internet. Real-time applications impose stringent delay and throughput constraints on the network, as compared with traditional elastic applications. When real-time applications communicate across network, data must traverse the

network in time for the application to use it. Thus, the applications' requirements, such as the end-to-end delay, delay jitter, and loss rate, are expressed as Quality of Service (QoS) parameters must be guaranteed by the underlying network. The upper bound on end-to-end delay from any source to any receiver in a real-time session is the main QoS parameter we consider in the dissertation.

At the inception of research in fault-tolerant systems as well as in multicast real-time network, somehow their inter-dependencies were not taken into account. This has led to the present state where we have a sizable amount of research activities and results in both fields, which largely tread along parallel paths. The underlying assumptions for having fault tolerance capability were often seen as conflicting with those for real-time performance, and vice versa. Due to those conflicts, today we have very few systems that can be guaranteed to meet critical timing constraints in the presence of faults. This motivates us to study the issues that consider both fault tolerance and real-time multicasting in one system.

In constructing a multicast tree, it is necessary to indicate what constitutes a "good" tree by defining the cost of the tree. There are several popular alternative ways of defining the cost of the multicast tree. The tree cost can be defined as the total cost of the links of the tree that spans the destinations in the multicast group. This definition is pertinent when the objective is to manage network resources efficiently. Another way to define the tree cost is as the sum of the cost of the path from a designated host, called the source, to the destinations in the multicast group. This definition of cost is important for delay-sensitive, real-time and interactive applications. The minimization of this tree cost is modeled as the shortest-path problems, which can be solved in polynomial time. In addition to minimizing the cost of the multicast tree to be constructed, another important consideration is constraining some properties of the tree. Constraints on the multicast tree are commonly intended to guarantee specified performance metrics about the tree. An important example of this is the delay-constrained multicast tree, where the multicast tree must satisfy an upper bound on the cost of the path from the root of the destination. Other constrained tree constructions include degree-constrained and delay-variation constrained.

Finally, it has been observed that it is expensive in terms of overhead for the construction of multicast tree for each sender. In order to conserve network resources, it is better to construct a group-shared tree, which is used to distribute the information from all sources sending to the group. Group-shared trees, however, are not appropriate when the required path metric, such as delay, to destinations has to be minimized. In a shared tree, the packets from a source do not traverse the shortest-path from the source to each destination. Moreover, group-shared trees can suffer from traffic concentration, where the traffic from all sources are concentrated on few links in the network, leading to increased congestion on the links.

Source-based trees usually contain a shortest delay path between source and each destination and are therefore appropriate for delay-sensitive applications. The main shortcomings of source-based trees are the poor scaling property and the possible high cost of the resulting tree. In this dissertation, we intend to study the issues involved in the design of fault-tolerant real-time multicast protocol, based on a practical CBT protocol. We believe our work is different from others which concentrate on source-based trees, that is suitable for a single source, while ours focus on shared-based tree with pre-computed disjoint backup paths.

1.2 Problem Statement

Much of the ongoing work related to QoS support of multimedia applications focuses on point-to-point communication. However, there are additional problems with multicast communication where unicast solutions are not necessarily extendable to multicasting. For example, while it is easy to determine an optimal route for unicast communications, it has been shown that doing the same for multicast communication is NP-complete. Also, supporting QoS requirements for multicast is more difficult than for unicast because there are multiple receivers as opposed to one receiver in unicast communications.

There is a need to develop efficient routing algorithms to support real-time multicasting. The algorithms must connect the multicast members with the least-cost tree and at the same time bounded by end-to-end delay time constraints of the applications. Link cost can be either a financial liability or can be based upon the bandwidth utilization, or distance (hop count) of the links. In addition to the cost, network

links may be associated with delays. A link delay includes CPU processing, queuing, transmission and propagation time suffered by packet over the link.

Multicasting in a multimedia environment requires satisfying the QoS requirements, as well as minimizing or reducing the overall cost. In particular, multimedia multicasting requires bounding the end-to-end delay from the source to every destination. It is not necessary to minimize the delay on all links spanning from source to all destinations.

Another aspect of the multicasting problem focuses on the issues that arise when the nodes join or leave a multicast session dynamically. This involves many additions and deletions to the multicast session over time. Support for dynamically join and leave operations brings about new challenges that need to be addressed. As nodes join and leave the group communication, the multicast tree may need to be updated. Hence, the routing algorithm that builds the tree connecting the multicast nodes should maintain efficient connections between the nodes when other nodes join or leave a multicast session.

Fault tolerance is an approach for ensuring that the system remains to be functional under faults. The key to fault tolerance in a multicast network is redundancy of the paths from source to all multicast group members. In the case of real-time systems, the stringency of timing requirements, the fault-tolerance requirements and the need for efficiency are problems which need to be addressed during the design phase of the system. The task of showing that such systems do simultaneously meet all the functional, timing and fault-tolerance requirements, remains an extremely challenging and complicated problems.

1.3 Dissertation Objectives

Given the need to support group communications in real-time environments and to build efficient multicast communication, the first objective of this dissertation is to design a bounded-delay multicast algorithm while cost is our secondary concern. So it is to find a suboptimal solution on multicast routing while delay constraint criteria has to be met. We apply this concept to a shared tree multicast routing protocol, CBT (core based tree). CBT is known for its superb quality of scalability in that only one shared tree is required

for any source, multicast group, as opposed to the poor scalability in the source tree. However, we have to overcome the problems of meeting end-to-end delay which is a major drawback of the shared tree. Also, the implication of the shared tree might result in traffic concentration around the core router. Solutions to the problems will be proposed in this dissertation. Our algorithms aim at satisfying the following design and performance criteria:

- End-to-end delay: The delay of a packet is defined as the summation of the routing delay, transmission delay and queuing delay. Also, the result should prove that our protocol can route the packet within time delay constraint of the real-time application under fault or no fault scenario.
- Network resource usage: Total number of hops a multicast packet travels to reach all destination in the multicast groups.
- Traffic concentration: Traffic concentration is measured by the maximum number of flows traversing a unidirectional link (the load of the most congested link.) This also shows link utilization of our proposed protocol against the CBT v2.
- Loss rate: The loss rate measures the fraction of the transmitted packets that are not delivered at all or are delivered so late as to be useless for real-time applications. The loss rate can be seen as the failure rate of our proposed protocol to construct the delay bound multicast tree.
- Execution time: The execution time measures the running time of our algorithm from start until the time the multicast tree is completely formed.

The second objective of this dissertation is addressing the fault-tolerance aspect of our proposed real-time multicast protocol in addition to meeting the delay constraint values. A key issue here is to find a pre-computed disjoint backup paths for any two nodes in the multicast tree and use them as alternative paths when a single link failure is encountered. All disjoint backup paths must be checked for meeting real-time constraints before the rerouting of the paths take place. In doing so, the multicast protocol should check whether the real-time constraint is not violated resulting from the router reconfiguration. Our fault-tolerant protocol aims at satisfying the performance criteria mentioned earlier.

1.4 Dissertation Contributions

Our contributions in this dissertation are the following:

- We proposed a shortest to the shortest path to optimize the path from source to the multicast tree thus bypassing the core router. The conventional CBT finds the shortest path tree from the source node to the core node and uses this path to route the traffic to all multicast members, resulting in traffic concentration around the core. With this modification, CBT has more chance to meet the end-to-end delay constraints, while traffic concentration around core router decreases significantly.
- We proposed two new path selection methods to ensure that the paths from source to all members in the shared multicast tree do not violate the end-to-end delay constraint condition, while cost of the multicast tree is reduced substantially. The first path selection method is based on weighted Dijkstra's path selection algorithm. The Dijkstra's Shortest Path Tree algorithm [30] was modified by substituting the original path selection with the weighted path selection function in order to create optimal solution balancing cost and delay parameters. The second path selection method is based on Kompella's selection function [66] which considers residual delay in addition to low cost tree. The new selection function explicitly uses both cost and delay in its functional form. It tries to choose low cost paths, but modulates the choice by trying to pick edges that maximize the residual delay. The idea is to reduce the cost of the tree through path sharing. We used simulation to prove the efficacy of our protocols against the original CBT v2.
- We proposed a new fault-tolerant protocol to enhance the proposed optimal real-time protocol. The idea is to pre-compute disjoint backup paths between each node pair for all links in the multicast tree. When a certain link on the multicast tree fails, our protocol will check if there exists a disjoint backup path for that link. The protocol will also check if the new disjoint backup paths can sustain end-to-end delay constraints. Then, the protocol verifies if the switching of routes from the failed link to the backup link takes no more than the maximum time delay allowed.

- We used simulation to evaluate the algorithms we proposed. In our simulation, we imitated the realistic Internet network environment. We conducted the experiment using ns-2 simulator to simulate a random network. The simulation shows that our proposed protocol performs well under a single link failure condition with moderate execution time and lower cost tree compared to that of the original CBT, while traffic concentration and network resource usage decreased substantially. Our proposed protocol verifies if the end-to-end delay condition is not violated in the simulated scenarios.

1.5 Dissertation Outline

Chapter 2 starts with a classification of multicast routing algorithms, followed by the survey of previous work on various multicast routing and related heuristic algorithms, based on the problems they address. In chapter 3 we study the shared tree multicast routing protocol, namely the core-based tree (CBT), its architecture, protocol overview, protocol format and its functions. We study its advantages and disadvantages compared with the source-specific multicast tree.

Chapter 4 presents a formal definition of the real-time multicast problems, previous approaches of the real-time multicast problems, and our proposed approaches. We also described how the previous approaches work and discuss advantages and disadvantages of each approach. We then focus on CBT and come up with a strategy to adapt CBT to a real-time requirement. We introduce a technique of optimizing the path previously used by a source reaching the shared multicast tree by using the shortest of the shortest path, to a specific on-tree node. Then, we proposed two new path selection methods, based on Dijkstra's algorithm [30] and Residual delay concept based on Kompella's algorithm [66], to find optimal paths for the new multicast trees whereby the end-to-end delay condition is not violated. With our approaches, the cost of the multicast tree diminished substantially. We then described our simulation to prove the efficacy of our protocol which bases on CBT v2. We compared our proposed protocol with the CBT v2 and show its improvement in terms of average end-to-end delay, network cost, network resource usage, traffic concentration, loss rate and execution time.

Chapter 5 presents the concept of fault-tolerance framework, failure model and solution to recover the network from failure. We study the concept of dependable network whereby disjoint backup paths are identified off-line. The resource to the disjoint backup path will be allocated at runtime. With this scheme, the runtime overhead remains the same, and therefore, the network performance is not compromised.

In chapter 6, we study the fault-tolerance aspect of the real-time traffic, and approaches to enhance the reliability of the network under a single link failure condition. We discuss many approaches to improve the network redundancies and introduce our fault-tolerant approach to the algorithms proposed in chapter 4. Our approach is based on first, finding the pre-computed disjoint backup paths and secondly, proposing an algorithm to switch from the primary link to secondary link within end-to-end delay constraints imposed by the applications. This involves the admission test of the pre-computed backup paths prior to switching of the failed path to the backup path. We show that our proposed approaches can reliably switch the multicast traffic along the failed link to the backup link within time bound and still performs better than the CBT v2 in terms of the performance metrics, described in previous chapter.

Conclusions and future work are presented in chapter 7. Finally, we summarize the contributions of the research conducted in this dissertation.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 2

MULTICAST ROUTING PROBLEM

2.1 Introduction

Multicasting refers to the transmission of data from one node (source node) to a selected group of nodes (member nodes or destination nodes) in a communication network. Instead of sending a separate copy of the data to each individual group member, a multicast source sends a single copy to all the members. Multicast routing requires the establishment of a multicast tree to allow group members to exchange data efficiently. In some case, the objective of constructing a multicast tree is to ensure that the QoS requirements of the underlying multicast traffic are met in addition to the data exchange to the group members.

The chapter is organized as follows. We give a discussion and background of the Multicast routing in section 2.2, followed by the Multicast problem definition in section 2.3. In section 2.4, we survey both source-specific multicast protocols, based on unconstrained and delay constrained conditions, and we discuss share-based multicast routing protocols. Section 2.5 studies the fault-tolerant aspect in multicast network. The chapter concludes with section 2.6.

2.2 Multicast Routing

A multicast routing algorithm is a method of selecting paths to connect a set of sources to one or more destinations. Support for multicast routing within the scope of a single local-area network (LAN) is simple. Broadcast LANs include the provision of multicast addresses at the medium access control level. Hosts can be configured to be a member of a selected set of multicast groups. Group members take advantage of the broadcast nature of the LAN to exchange data by transmitting a single copy of the data frame to the multicast group. Upon recognition of its multicast group address, a host captures and forwards the transmitted frame to upper layer protocols. The most constraining aspect of link-layer multicast solutions is the provisioning of multicast groups. Usually,

multicast groups are statically set up by the network provider and can be change only by a reconfiguration of the network.

Multicasting is far more complex to support in a wide-area network than a shared local-area network. Several issues make such a task difficult. Hosts must be able to join and leave the network without necessarily requiring the intervention of the network service provider. Furthermore, network multicast routers must be able to identify and locate members of the multicast group. An efficient multicast routing strategy must, therefore, be devised to disseminate transmitted data to all of the intended recipients.

Given a multicast group M and possibly a set of optimization objective functions O , multicast routing is a process of constructing, based on the network topology and the network state, a multicast tree T that optimizes the objective functions. In the case of constraint-based multicast routing, a set of constraints C in the form of end-to-end delay bound, interreceiver delay jitter bound, minimum bandwidth, loss probability and/or a combination thereof, is given. The resulting multicast tree must provide not only reachability from source(s) to a set of destinations, but also certain QoS merits on the routes found in order to satisfy the constraints. The optimization objectives sought for are usually defined in the form of minimizing the cost of a multicast tree, where the cost may be the total bandwidth used and/or a monotonically non-decreasing function of network utilization. The constraints imposed can be classified into two categories, as follows:

- Link constraints: The link constraints are restrictions on the use of links for route selection. For example, one may request that the bandwidth or buffer available on a link be greater than or equal to a pre-determined value.
- Tree constraints: The tree constraints are either (i) bounds on the combined value of a performance metric along each individual path from the source to a receiver in a multicast tree, e.g. the end-to-end delay bound on the paths from the source to all the receivers, or (ii) bounds on the difference of the combined value of a performance metric along the paths from the same source to any two different receivers, e.g. the inter-receiver delay jitter bound defined as the difference between the end-to-end delays along the paths from the same source to any two

different receivers. In the case of heterogeneous QoS, a different constraint may be imposed for each receiver.

Depending on how a tree constraint is derived from the corresponding link metrics, tree constraints can be further classified into the following three types:

- Additive tree constraints: For any path $P_T(u,v) = (u, i, j, \dots, k, v)$, the tree constraint is additive if:

$$m(u,v) = m(u, i) + m(i, j) + \dots + m(k, v). \quad (2.1)$$

For example, the end-to-end delay $d(u,v)$ from node u to node v , is additive and is equal to the sum of individual link metric $d(i, j)$ along the path $P_T(u,v)$.

- Multiplicative tree constraints: The tree constraint is multiplicative if:

$$m(u,v) = m(u, i) \times m(i, j) \times \dots \times m(k, v). \quad (2.2)$$

For example, the probability, $1-p_L(u,v)$, for a packet to reach node v from node u along $P_T(u,v)$ is multiplicative and is equal to the product of individual link metric $1-p_L(i,j)$, along the path $P_T(u,v)$.

- Concave tree constraints: The tree constraint is concave if:

$$m(u,v) = \min [m(u, i) , m(i, j) \dots, m(k, v)]. \quad (2.3)$$

For example, the bandwidth $b(u,v)$, available along a path from node u to node v , is concave and is equal to the minimum bandwidth among the links on path $P_T(u,v)$.

Depending on the link /tree constraints imposed and the objective function used, a multicast routing problem can be formulated as:

- Link constrained problem: A link constraint is imposed to construct feasible multicast trees, e.g. bandwidth constrained routing.
- Multiple link constrained problem: Two or more link constraints are imposed to construct feasible trees, e.g. bandwidth and buffer constrained routing.

- Tree constrained problem: A tree constraint is imposed to construct feasible multicast trees, e.g. delay constrained routing.
- Multiple tree constrained problem: Two or more tree constraints are imposed to construct feasible multicast trees, e.g. delay and inter-receiver delay jitter constrained routing.
- Link and tree constrained problem: A link constraint and a tree constraint are imposed to construct feasible multicast trees, e.g. delay and bandwidth constrained routing.
- Link optimization problem: A link optimization function is used to locate an optimal multicast tree, e.g. maximization of the link bandwidth over on-tree links in a multicast tree.
- Tree optimization problem: A tree optimization function is used to locate an optimal multicast tree, e.g. minimization of the total cost of a multicast tree. This is also known as the *Steiner tree* problem.
- Link constrained link optimization problem: A link constraint is imposed and a link optimization function is used to locate an optimal multicast tree that fulfills the constraint, e.g. the bandwidth constrained buffer optimization problem.
- Link constrained tree optimization problem: A link constraint is imposed and a tree optimization function is used to locate an optimal multicast tree, e.g. the bandwidth constrained Steiner tree problem.
- Tree constrained link optimization routing problem: A tree constraint and a link optimization function is used to locate an optimal multicast tree, e.g., the delay constrained bandwidth optimization problem.
- Tree constrained tree optimization routing problem: A tree constraint and a tree optimization function is used to locate an optimal multicast tree, e.g. the delay constrained Steiner tree problem.
- Link and tree constrained tree optimization routing problem: Link and tree constraints and a tree optimization function is used to locate an optimal multicast tree, e.g. the bandwidth and delay constrained tree optimization problem.

	No optimization	Link optimization	Tree optimization
Null constraint		(6) link optimization polynomial time complexity	(7) tree optimization NP-complete complexity
Link constraint	(1) link constrained polynomial complexity	(8) link constrained link optimization polynomial time complexity	(9) link constrained tree optimization NP-complete complexity
	(2) multiple link constrained polynomial complexity		
Tree constraint	(3) tree constrained polynomial time complexity	(10) tree constrained link optimization polynomial time complexity	(11) tree constrained tree optimization NP-complete complexity
	(4) multiple tree constrained NP-complete complexity		
Link & tree constraints	(5) link & tree constrained polynomial time complexity		(12) link & tree constrained tree optimization NP-complete complexity

Table 2.1: A taxonomy of multicast routing problems.

Table 2.1 gives a summary of these problems. In our formulation which falls in the category of “Tree constrained problem”, we are interested in finding the shared multicast tree which satisfies the maximum end-to-end delay bound condition and its backup paths of such tree, such that the overall tree cost is reduced. In many cases, this problem is classified as the “delay constrained multicast routing” problem. Delay constraints and Cost constraints are both considered additive metrics.

To satisfy multiple constraints, a single mixed metric has been proposed in current networks. One possible approach might be to define a function and generate a single metric from multiple parameters. The idea is to mix various pieces of information into a single measure and use it as the basis for routing decisions. For example, a mixed metric M may be produced with bandwidth B , delay D and loss probability L with a formula $f(p) = \frac{B(p)}{D(p) \times L(p)}$. A path with a large value is likely to be a better choice in terms of bandwidth, delay and loss probability. However, by mixing parameters of different composition rules, there may not be a simple composition rule for the selected function. Another alternative to a single mixed metric is the multiple metrics approach, but finding a path subject to multiple constraints is inherently hard. Polynomial-time algorithms for the problem may not exist. We will explore this idea further in chapter 4.

2.3 Multicast problem Definitions

A point-to-point communication network is represented as a directed, connected, simple network $G = (V, E)$, where V is a set of nodes and E is a set of directed links. The

existence of a link $e = (u, v)$ from node u to node v implies the existence of a link $\bar{e} = (v, u)$ for any $u, v \in V$, i.e., full duplex in networking terms. A link $(u, v) \in E$ is an outgoing link for node $u \in V$ and an incoming link for $v \in V$. Any link $e = (u, v) \in E$ has a cost $C(e)$ (same as $C(u, v)$) and a delay $D(e)$ (same as $D(u, v)$) associated with it. $C(e)$ and $D(e)$ may take any nonnegative real values. The link cost $C(e)$ may be either a monetary cost or some measure of the link's utilization. The link delay $D(e)$ is a measure of the delay a packet experiences when traversing the link e . Thus it may consist of queuing, transmission, and propagation components. Because of the asymmetric nature of computer networks, it is often the case that $C(e) \neq C(\bar{e})$ and $D(e) \neq D(\bar{e})$. If the network is symmetric, it can be represented as an undirected network in which $C(e) = C(\bar{e})$ and $D(e) = D(\bar{e})$ for all $e \in E$.

We define a path as an alternating sequence of nodes and links $P(v_0, v_k) = v_0, e_1, e_2, v_2, \dots, v_{k-1}, e_k, v_k$, such that every $e_i = (v_{i-1}, v_i) \in E$, $1 \leq i \leq k$. A path contains loops if some of its nodes are not distinct. If all nodes are distinct, then the path is loop-free. In the remainder of this dissertation, it will be explicitly mentioned if a path contains loops. Otherwise a "path" always denotes a loop-free path. We will use the following notation to represent a path: $P(v_0, v_k) = \{v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_{k-1} \rightarrow v_k\}$. The cost of a path $P(v_0, v_k)$ is defined as the sum of the costs of the links constituting $P(v_0, v_k)$:

$$\text{Cost}(P(v_0, v_k)) = \sum_{e \in P(v_0, v_k)} C(e). \quad (2.4)$$

Similarly, the end-to-end delay along the path $P(v_0, v_k)$ is defined as the sum of the delays on the links constituting $P(v_0, v_k)$:

$$\text{Delay}(P(v_0, v_k)) = \sum_{e \in P(v_0, v_k)} D(e). \quad (2.5)$$

The definitions given below apply for a multicast session with a single source. A multicast group $G = \{g_1, \dots, g_n\} \subseteq V$, where $n = |G|$, is a set of nodes participating in the same network activity, and is identified by a unique group address i . A node $s \in V$ is a multicast source for the multicast group G . A multicast source s may or may not be itself a member of the group G . A source-specific multicast tree $T(s, G) \subseteq E$, is a tree

rooted at the source s and spanning all members of the group G . The total cost of a tree $T(s, G)$ is simply the sum of the cost of all links in that tree.

$$\text{Cost} (T(s, G)) = \sum_{e \in T(s, G)} C(e). \quad (2.6)$$

In general, an algorithm that minimizes the total cost of a multicast tree will encourage the sharing of links. The maximum end-to-end delay of a multicast tree is the maximum delay from the source to any multicast group member.

$$\text{Max_Delay} (T(s, G)) = \max_{g \in G} (\sum_{e \in P_T(s, g)} D(e)), \quad (2.7)$$

where $P_T(s, g)$ is the path from s to g along the tree $T(s, G)$.

2.4 Survey of Multicast Routing Algorithms

Multicast routing algorithms are grouped together in compliance with the classification provided in the previous section. For simplicity purposes, we will consider two multicast routing tree construction namely, source-based trees and shared-based trees. The taxonomy of various types of approaches to the multicasting problems is shown in figure 2.1 and figure 2.2.

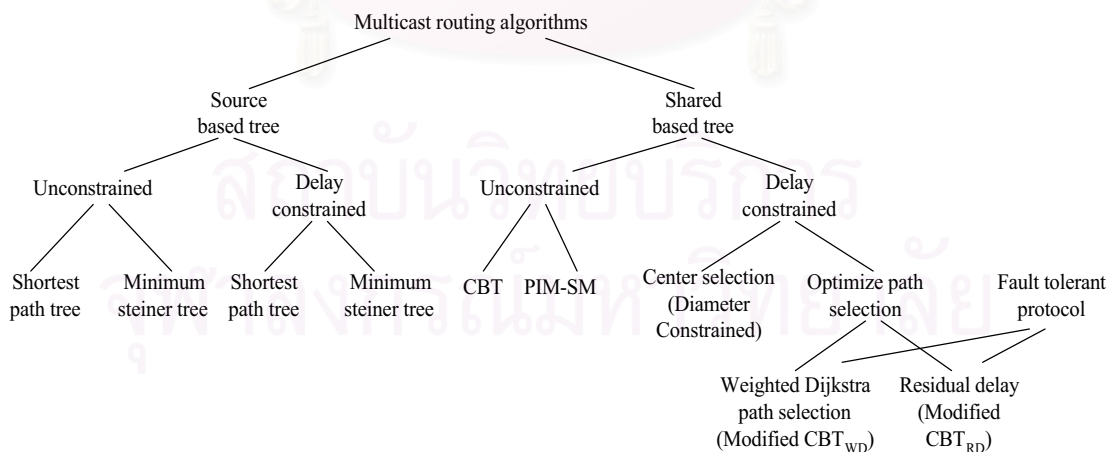


Figure 2.1: Tree structures of solution approaches to the multicasting problems

As for the fault-tolerant protocol of our proposed approach, we follow the fault-tolerant tree structure shown in figure 2.2, as follows:

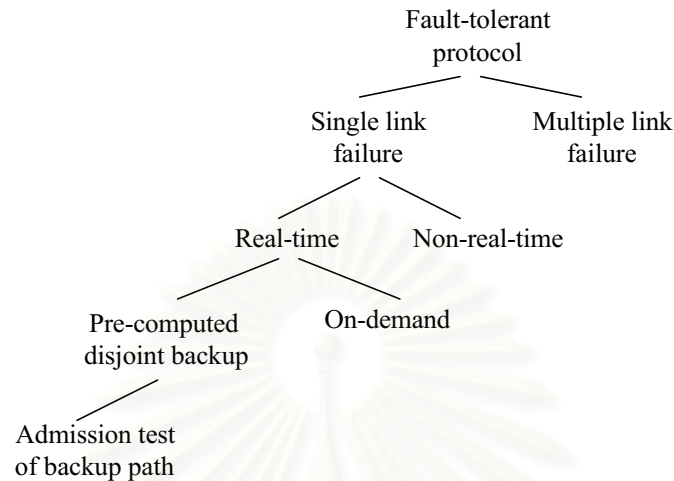


Figure 2.2: Fault-tolerant protocol description

Full detail of our fault-tolerance study can be found in chapter 5.

2.4.1 Source-specific Multicast trees

We conducted our multicast routing survey on source-specific multicast tree problem which is classified into unconstrained and delay constrained algorithms. Each classification can be subdivided further into Shortest Path algorithms and Minimum Steiner algorithms.

- **Unconstrained Shortest Path Algorithms**

As the name indicates, a shortest path algorithm minimizes the sum of the lengths of the individual links on each individual path from the source node to a multicast group member. The properties of a shortest path tree depend on the metric the link length represents. If unit link lengths are used, the resulting shortest path tree is a minimum-hop tree. If the link length is set equal to the link cost, then a shortest path algorithm, denoted as the least-cost (LC) algorithm in this case, computes the LC tree. The objective of an LC algorithm can be expressed mathematically as follows:

$$\min_{T(s, G) \in \tau(s, G)} \text{Cost}(P_T(s, g)) \quad \forall g \in G, \quad (2.8)$$

where $\tau(G)$ is the set of trees at s and spanning all nodes in G . The total cost of an LC tree is not necessarily optimal. If the length of a link is a measure of the delay on that link, then a shortest path algorithm, denoted as least-delay (LD) algorithm in this case, computes the LD tree. The objective function of an LD algorithm is to:

$$\min_{T(s, G) \in \tau(s, G)} \text{Delay}(P_T(s, g)) \quad \forall g \in G. \quad (2.9)$$

An LD tree is optimal with respect to end-to-end delay. In case of real-time applications, if the LD tree cannot satisfy the imposed delay constraint, no other multicast tree can.

Bellman-Ford algorithm [16] and Dijkstra algorithm [31] are two well known shortest path algorithms. Both algorithms are exact and run in polynomial time. The worst case time complexity of the Bellman-Ford algorithm is $O(|V|^3)$ where $|V|$ is the number of nodes in the network. An exact, distributed version of the Bellman-Ford algorithm is given in [18]. It requires only limited information about the network topology to be kept at each node. Awerbuch et al. [2] show that the worst case message complexity of the exact, distributed Bellman-Ford algorithm may grow exponentially with the number of nodes. To avoid this excessive complexity, they propose two approximate distributed versions of the algorithm. For Dijkstra's shortest path algorithm, only centralized versions exists. Its execution time is $O(|V|^2)$ time in the worst case. Efficient, nondistributed versions of both Bellman-Ford and Dijkstra's algorithms have comparable average running times [17]. Both algorithms remain exact for asymmetric networks.

The reverse path forwarding (RPF) algorithm proposed by Dalal and Metcalfe [25] is an algorithm for broadcasting in datagram networks. Each packet is forwarded from the source to the receivers over the reverse shortest paths, i.e., the shortest paths from the receivers back to the source. Thus RPF creates an optimal shortest path broadcast tree only if the network is symmetric. Deering [26, 27] generalized the RPF algorithm (TRPB) algorithm and the reverse path multicasting (RPM) algorithm. The objective function of TRPB and RPM can be stated as:

$$\min_{T(s, G) \in \tau(s, G)} \sum_{e \in P_T(s, g)} C(e') \quad \forall g \in G, \quad (2.10)$$

where $e = (u, v)$ and $e' = (v, u)$. TRPB and RPM do not suffer from some of the limitations which RPF suffers from with respect to its applicability to multi-access networks. RPF, TRPB, and RPM are distributed algorithms that rely on limited information at each node in the network. They scale well with the size of the network, and dynamic implementations of these algorithms exist.

- **Unconstrained Minimum Steiner Tree Algorithms**

The objective of the minimum Steiner tree problem is to minimize the total cost of the multicast tree, i.e.,

$$\min_{T(s, G) \in \tau(s, G)} \text{Cost}(T, (s, G)). \quad (2.11)$$

This problem is known to be *NP*-complete [56]. Hwang [44] provided an extensive survey of both exact and heuristic minimum Steiner tree algorithms. An earlier survey was given by Winter [112]. Very few algorithms have been proposed for the minimum Steiner tree problem in asymmetric networks, and all of them operate under special assumptions, e.g. acyclic networks. If the multicast group includes all nodes in the network, the minimum spanning tree problem in symmetric networks can be solved in $O(|V|^2)$ time in the worst case using Prim's algorithm [82]. Unconstrained minimum Steiner tree algorithms do not attempt to optimize the end-to-end delay at all. Therefore they may not be suitable for real-time applications. The best known minimum Steiner tree heuristics were proposed by Kou, Markowsky, and Berman (KMB heuristic) [69], Takahashi and Matsuyama (TM heuristic) [96], and Rayward-Smith (RS heuristic) [85].

The KMB heuristic [69] uses Prim's minimum spanning tree algorithm [82] during its computation. Prim's algorithm is optimal only for symmetric networks. Thus the cost performance of the KMB heuristic may be affected if it is applied to asymmetric networks. The worst case time complexity of the KMB heuristic is $O(|G| |V|^2)$, where $|G|$ is the size of the multicast group. Wall [102, 103] proposed a distributed version of the KMB heuristic. The total cost of trees generated using KMB heuristic in symmetric

networks is on the average only 5% worse than the cost of the optimal minimum Steiner tree [32, 33].

The TM heuristic [96] starts with a tree that contains the source node only. Then it adds the multicast group members, one at a time, to the existing tree via the cheapest LC path to any node already in the tree. TM heuristic runs in $O(|G| |V|^2)$ time in the worst case.

The RS heuristic [85] starts with a forest of trees, with each multicast group member constituting a tree. Then the heuristic unites trees that are closest to each other (in terms of cost) by adding the appropriate links until it ends up with a single tree. Using a limited number of simulations, Rayward-Smith and Clare [84] showed that RS heuristic yields tree costs that are closer to optimal than KMB and TM heuristics. Unfortunately, however, RS heuristic was designed for symmetric networks, and we can not envision an efficient method for implementing it in case of asymmetric networks.

Jiang [52] presented modified versions of KMB heuristic and RS heuristics that construct multicast trees with lower costs than the original heuristics. The author used heterogeneous link capacities in the symmetric, random networks he simulated, and he defined the link cost as function of the utilized link bandwidth. The same author also proposed a distributed minimum Steiner tree heuristic in [53].

Recently, Ramanathan [83] proposed a heuristic for constructing minimum Steiner trees in asymmetric networks. This heuristic permits trading off low tree cost for fast execution time by proper selection of a parameter k . The author showed that Dijkstra shortest path algorithms, KMB minimum Steiner tree heuristic, and TM minimum Steiner tree heuristic are particular cases of the proposed heuristics when k is set to 1, $(|G|+1)$, and $|V|$ respectively.

Many other heuristics for constructing minimum Steiner trees in communication networks were proposed. See for example Chow [23], Leung and Yum [70], and Bauer and Varma [14].

- **Delay-Constrained Shortest Path Algorithms**

Delay-constrained shortest path algorithms minimize the cost of each path, i.e., the sum of the link costs, from the source node to a multicast group member subject to an end-

to-end delay constraint. Thus the tree is a delay-constrained LC tree. An algorithm for solving the delay-constrained LC problem has the same objective function as that of the unconstrained LC problem, stated in equation 2.5, with the added constraint that:

$$\text{Max_Delay}(T(s , G)) \leq \Delta, \quad (2.12)$$

where Δ is the value of the imposed delay constraint. The delay-constrained shortest path problem is *NP*-hard [42]. A few algorithms for solving that problem were proposed recently, motivated by the increasing importance of end-to-end delay as a QoS constraint for real-time applications. We summarize the distinguishing characteristics of each algorithm below. Note that the delay-constrained multicast routing algorithms surveyed in this section and in the next section are only applicable for the construction of source-specific trees.

Widyono [111] presented the constrained Bellman-Ford (CBF) algorithm. CBF performs a breadth-first search to find the delay-constrained shortest path tree. CBF is optimal and therefore its running times grow exponentially with the size of the network. Widyono used CBF as a basis for several delay-constrained minimum Steiner tree heuristics which will be surveyed in the next section.

Sun and Langendoerfer [95] proposed a delay-constrained shortest path heuristic. We call it the constrained Dijkstra heuristic (CDKS) because it is based on Dijkstra shortest path algorithm. This heuristic computes an unconstrained LC tree. If the end-to-end delay to any group member violates the delay constraint, the path from the source to that group member is replaced with the LD path. Thus if the LC tree violates the delay constraint, and LD tree must be computed, and the two trees are merged. This algorithm always finds a constrained multicast tree if one exists. CDKS runs in $O(|V|^2)$ time, the same as Dijkstra's algorithm. The authors compared the cost performance of their heuristic to LD and KPP (a delay-constrained minimum Steiner tree heuristic which will be presented in the next section) using simulation over random networks. They used unit link costs and integer link delays ranging in value from 1 to 5.

Wi and Choi [110] presented a distributed LD algorithm and proposed to use it for solving the delay-constrained shortest path problem. They used simulation to

evaluate its performance and execution times relative to KPP for 20-node symmetric networks.

- **Delay-Constrained Minimum Steiner Tree Algorithms**

The delay-constrained source-specific minimum Steiner tree problem was first formulated by Kompella, Pasquale, and Polyzos [65, 66]. The authors proved the *NP*-completeness of the problem. The objective of the problem is to minimize the total cost of the tree, equation 2.11, without violating the imposed delay constraint, equation 2.12. Optimal algorithms for this problem exist. For example, Noranha and Tobagi [77] proposed an algorithm, based on integer programming, which constructs the optimal source-specific delay-constrained minimum Steiner trees for multiple multicast sessions simultaneously. However, this algorithm is rather complex and is useful only as a reference to evaluate heuristic solutions for the same problem.

The first heuristic for the delay-constrained minimum Steiner tree problem was given by Kompella, Pasquale, and Polyzos [65, 66]. We label this the KPP heuristic. KPP assumes that the link delays and the delay constraint, Δ , are integers while the link costs may take any positive real value. The heuristic is dominated by computing a constrained closure graph which takes time $O(\Delta|V|^3)$. Thus KPP takes polynomial time only if Δ has a fixed value. When the link delays and Δ take noninteger values, Kompella et al. propose to multiply out fractional values to get integers. Following this approach, KPP is guaranteed to construct a constrained tree if one exists. However, in some cases the granularity of the delay constraint becomes very small, and hence the number of bits required to represent it increases considerably. As a result the order of complexity, $O(\Delta|V|^3)$, may become too high. To avoid prohibitively large computation times, a fixed granularity may be used. However, fixing the granularity has side effects. When the granularity is comparable to the average link delays, KPP's accuracy is compromised and in many cases it fails to construct a constrained multicast tree when one exists. The authors proposed two alternative objective functions for KPP to use during tree construction. The first is a function of the link cost only. The second objective function is a function of both the link cost and the residual delay if this link is

added to the tree. The authors used simulation of random, symmetric networks with up to 100 nodes to evaluate their heuristic.

Similar to KMB, KPP uses Prim's algorithm [82] to obtain a minimum spanning tree of a closure graph. Prim's algorithm is only optimal for symmetric networks. This might affect the performance of KPP when applied to asymmetric networks.

Kompella, Pasquale, and Polyzos also proposed a distributed heuristic solution for the delay-constrained minimum Steiner tree problem [68]. The heuristic is based on Prim's algorithm [82], but it involves the making and breaking of cycles during the construction of the multicast tree. It runs in $O(|V|^3)$ time, and is guaranteed to find a multicast tree, if one exists.

Widyono [111] proposed four delay-constrained minimum Steiner tree heuristics. The four delay-constrained heuristics are based on the CBF algorithm described in the previous section. Therefore all of them have worst case scenarios with exponentially growing execution times. Widyono's constrained adaptive ordering (CAO) heuristic yields better performance than the other three constrained heuristics he proposed. In CAO, the CBF algorithm is used to connect one group member at a time to the source. After each run of CBF, the unconnected member with the cheapest constrained LC path to the source is chosen and is added to the existing subtree. The costs of links in the already existing subtree are set to zero. CAO is always capable of constructing a constrained multicast tree, if one exists, because of the nature of the breadth-first search CBF conducts. Widyono defined the link cost as a function of the available bandwidth, the residual buffer space, and the link's delay. The link delay was defined as the sum of the queuing, transmission, and propagation delays along the link. The author evaluated his heuristics using simulation of eight mesh networks.

The bounded shortest multicast algorithm (BSMA) was proposed by Zhu, Parsa, and Garcia-Luna-Aceves [114]. BSMA starts by computing an LD tree for a given source and multicast group. Then it iteratively replaces superedges in the tree with cheaper superedges not in the tree, without violating the delay constraint, until the total cost of the tree cannot be reduced any further. BSMA uses a k th-shortest path algorithm to find cheaper superedges. It runs in $O(k|V|^3 \log |V|)$ time. In case of large, densely connected networks, k may be very large, and it may be difficult to achieve

acceptable running times. It is possible to tradeoff multicast tree cost for fast execution speed when using BSMA by either limiting the value of k in the k th-shortest path algorithm or by limiting the number of superedge replacements. BSMA always finds a constrained multicast tree, if one exists, because it starts with an LD tree. The authors defined the link cost as a function of the link utilization and defined the link delay as the sum of the queueing delay, transmission delay, and propagation delay over the link. They evaluated the performance of BSMA and compared it to KMB and LD. Random networks with up to 100 nodes generated using Waxman's random network generator [107] were used.

We consider the minimum Steiner tree heuristic proposed by Waters [105] to be semi-constrained, because it uses the maximum end-to-end delay from the source to any node in the network (not to any group member) as the delay constraint. Note that this constraint is not related directly to the application's QoS constraints, and that, depending on the network delays, this internally computed constraint may be too strict or too lenient as compared to the QoS requirements of the application. The heuristic then constructs a broadcast tree that does not violate the internal delay constraint. Finally the broadcast tree is pruned beyond the multicast nodes. The authors call this the semiconstrained (SC) heuristic. In [89], they implemented the original algorithm proposed in [105] which resembles a semi-constrained minimum spanning tree, and the authors also implemented a modified version which is closer to a semi-constrained shortest paths broadcast tree. Simulation results given in [89] showed that the modified version, denoted as the modified semiconstrained (MSC) heuristic always performs better than the original heuristic with respect to tree costs, end-to-end delays, and network balancing. SC and MSC is dominated by the computation of the internal delay bound. This computation uses an extension to Dijkstra's algorithm, and therefore it takes $O(|V|^2)$ time in the worst case.

In addition to the algorithms surveyed above, many other variations of the multicast routing problem have been studied over the years. Research reports on the dynamic multicast routing problem, in particular, appeared frequently in the literature. We dedicate the next section to previous work on that problem.

- **Dynamic Multicast Routing Algorithms**

Dynamic multicast routing algorithms were proposed to avoid rerouting an entire multicast tree whenever a node joins or leaves a multicast session. In dynamic multicast routing algorithms, when a node leaves a multicast session, the path connecting that node is simply pruned from the tree if it is not used to connect any other multicast group members. The situation is more difficult when a node joins an existing multicast session.

Waxman [89, 107] presented a greedy dynamic multicast routing algorithm. The algorithm has a weighting parameter w that varies from 0 to 0.5. When $w = 0$, a node joins an existing source-specific multicast tree via the shortest path to the tree. When $w = 0.5$, the node is added to the existing tree via the shortest path to the source. Waxman evaluated his algorithm using simulation over randomly generated 56-node and 60-node networks. He proposed an algorithm for generating random networks that resemble realistic networks. This random network generator has been adopted by many researchers in subsequent years.

Doar and Leslie [33] investigated a naive approach that always connects a joining node to the existing tree via the shortest path from the source. They simulated this mechanism using randomly generated networks, both flat and hierarchical. Their random network generator is a modified version of Waxman's generator. Simulations over 100-node networks showed that the naive approach constructs trees that are on the average 50% more expensive than costs of trees constructed using the static KMB heuristic.

Kadirire [54] defined the geographic spread as the shortest distance from any node in the network to the existing tree averaged over all nodes not in the tree. He proposed a geographic spread dynamic multicast (GSDM) algorithm that maximized the performance of GSDM and compared it to Waxman's heuristic and Doar and Leslie's heuristic in [55] using simulation over random networks with up to 100 nodes. He showed that GSDM and Waxman's heuristics yield similar performance and are consistently better than Doar and Leslie's naive approach.

Biersack and Nonnenmacher [19] proposed a dynamic, distributed multicast routing algorithm named WAVE. WAVE uses a weighted function of the cost and the

delay to attach a joining node to the existing tree. The authors evaluated their algorithm in comparison to static algorithms only.

Bauer and Varma [12] presented a dynamic multicast routing algorithm: ARIES. The operation of ARIES is similar to Waxman's dynamic algorithm. In addition, however, a subtree of the multicast tree is completely reconstructed each time a pre-specified number of joins and leaves affects that subtree. The subtree reconstruction ensures that the cost of the multicast tree remains close to optimal. ARIES was evaluated using simulation over 200-node random networks. The authors used a modified version of Waxman's random network generator.

- **Other Multicast Routing Algorithms**

In this section, we briefly survey a few more multicast routing algorithms that do not belong to any of the categories listed in the previous subsections.

Bharath-Kumar and Jaffe [10] presented a tradeoff algorithm between the minimum Steiner tree and the LD tree. This algorithm constructs the minimum Steiner tree; then it locates the receiver with the largest difference between the delay along its path in the minimum Steiner tree and the delay along the LD path from the source to that receiver. The algorithm then replaces the minimum Steiner tree path with the corresponding LD path. The same authors also proposed two distributed multicast routing heuristics which are based on local information from nearby nodes only.

Rouskas and Baldine [86] studied the problem of constructing multicast trees subject to both an end-to-end delay constraint and a delay variation constraint. They defined the delay variation constraint as the maximum difference, that can be tolerated, between the end-to-end delays along the paths from the source to any two receivers. The authors proved that this problem is *NP* complete; then they proposed a heuristic solution.

Research on the degree-constrained multicast routing problem is motivated by the fact that current multicast capable high-speed switches have limited copy capability. In addition, limiting the maximum degree at any node in the multicast tree results in more evenly distributed load among all nodes in the network. Tode et al. [99] proposed two algorithms for degree-constrained multicast routing. The first algorithm minimizes

the average degree of the multicast tree it constructs, while the second algorithm attempts to construct a low-cost multicast tree subject to a given maximum degree constraint. The authors set the link costs equal to the link delays when evaluating the performance of their heuristics.

Bauer and Varma [13] investigated a variation of the degree-constrained multicast routing problem in which the degree-constraint may vary for the different nodes in the network. Using simulation, they showed that many of the existing unconstrained minimum Steiner tree heuristics are capable of constructing degree-constrained multicast trees. The authors also proposed a simple degree-constrained heuristic which performs better than all other algorithms of the same or lesser complexity.

Ammar et al. [1] studied the problem of routing virtual paths (VP) for multicast communication in ATM networks. When constructing a multicast tree, they took into account the bandwidth cost, the switching cost, and the connection establishment cost. The authors studied different types of VPs. They formulated the problem as an integer programming problem and proposed heuristic solutions based on the transshipment simplex algorithm. The authors used a single 16-node network for evaluating their heuristics.

Kim [63] studied a similar problem. He proposed an optimal solution to the problem of routing multiple multicast connections simultaneously in ATM networks.

2.4.2 Shared Multicast trees

Many network applications involve multiple sources and multiple receivers, e.g., videoconferencing. The multicast routing problem for these applications is known as the many-to-many problem. There are two alternate approaches to address this problem. One approach is to consider the many-to-many problem as multiple one-to-many multicast routing problems, and to construct a source-specific multicast tree for each source. The other approach is to use a single shared multicast tree, which is constructed for a particular source, multicast group. In this approach, traffic streams from multiple source share the links of the same tree and hence gives the name shared multicast trees. Table 2.2 compares the number of router state between the source-

specific multicast tree to that of the shared multicast tree in many-to-many multicasting scenario. The shared multicast tree offers more scalable solutions in terms of router states at larger multicast group size.

Number of multicast groups	10			100			1,000		
Multicast group size (number of members)	20			40			60		
Number of sources per group	10%	50%	100%	10%	50%	100%	10%	50%	100%
Number of source tree router entries	20	100	200	400	2,000	4,000	6,000	30,000	60,000
Number of shared tree router entries	10			100			1,000		

Table 2.2: Comparison of source and shared multicast tree router state.

Shared Multicast Trees versus Source-Specific Multicast Trees

Both source-specific multicast trees and shared multicast trees have their advantages and disadvantages. Shared multicast trees have the following advantages over source specific multicast trees.

- It takes less overhead to construct and maintain one shared tree per multicast session than to construct a source-specific multicast tree for every source transmitting to that session. With shared multicast trees, when a new source starts transmitting to an already existing multicast session, it does not have to construct an entire source-specific tree. It merely has to find a path to connect itself to the existing shared tree of that session. Similarly, when a node joins an existing multicast session as a receiver, it merely has to find a path to connect itself to that session's shared tree. If source-specific are used instead, the new receiver will have to join the source-specific trees of each source transmitting to the already existing multicast session.

- When using shared multicast trees, a source node does not have to keep an explicit list of the members of the multicast session. Similarly, when a node is a member in a multicast session, it does not have to keep an explicit list of all sources.

On the other hand, shared multicast trees suffer from the following disadvantages as compared to source-specific multicast trees.

- High traffic concentration. Traffic streams from different sources share the links of the same tree, which results in high traffic concentration on these links. The traffic of a given multicast session will be concentrated on the links of the shared tree.
- End-to-end delays along shared trees are longer than the corresponding delays when source-specific trees are used. An example, comparing the delays along a shared tree to the delays if source-specific trees are used instead.

Intensive development efforts are currently under way in the standard bodies to evolve efficient scalable multicast routing protocols, and shared multicast trees, namely PIM-SM and CBT are included in the specifications of these protocols. In both protocols, receivers join the shared multicast tree via the forward shortest paths towards the center, and sources transmit to the shared tree via the forward shortest paths towards the center. Thus receivers receive the sources' traffic streams via the shortest reverse paths from the center. The difference between the shared modes on PIM-SM and CBT is mainly in the mechanisms used to maintain the tree.

CBT permits the use of multiple cores. A receiver has to join only one core, and a source unicasts its packets towards one core only. As soon as the packets arrive at any node in the shared tree, they are multicast towards all destinations. In CBT, there is one primary core. The other cores join the primary core via the shortest paths to construct a core backbone. The result is a single shared tree with multiple cores. Figure 2.3 illustrates the multiple core-based tree structure.

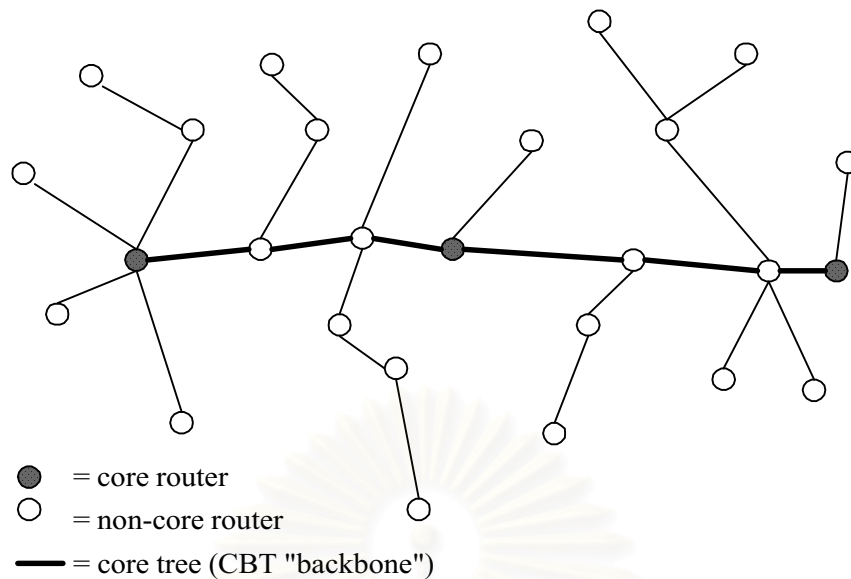


Figure 2.3: Multiple core-based trees structure

An earlier draft of PIM-SM permitted the use of multiple RPs (Rendezvous points) per group. Each source sends packets towards each of the RPs, but receivers only join towards a single RP. This results in RP having its own shared tree that spans only a subset of the multicast group members. The motivation for having multiple RPs was to achieve fault tolerant operation in case of RP failures. The most recent draft of PIM-SM, however, no longer permits the use of multiple RPs. Currently, a multicast session selects an RP list from a prespecified RP set. Only one node from the RP list is the active RP. Not much robustness is sacrificed, since the other nodes in the RP list act as multiple backup RPs, and recovery time after an RP failure is small.

In PIM-SM, an RP is chosen randomly from the candidate RPs list, and in CBT cores are placed by hand based on the topological distribution of the group membership at session initiation time. Previous work on shared based trees are reviewed as follows.

David Wall [102] investigated the problems of broadcasting and multicasting (selective broadcasting) in his PhD dissertation. His main focus was on broadcasting and not multicasting. The only cost function he considered was link delays. Therefore his shortest path trees were least-delay trees. He dedicated two chapters for broadcasting and multicasting over shared trees. He proposed different criteria to be optimized in a shared tree, e.g., the average end-to-end delay or the maximum end-to-

end delay. He also described the optimum shared tree corresponding to each criterion he proposed. The optimum solutions were complicated and in some cases even NP-complete. To avoid this excessive complexity, Wall proposed three heuristic algorithms to select the center of a broadcast (or a multicast) tree.

- The first heuristic algorithm selects the node to be the center whose shortest path tree spanning all members has the least maximum delay to any member. This is the graph theoretical center.
- The second heuristic algorithm selects the node to be the center whose shortest path tree spanning all members has the least average delay to all members. This is the graph theoretical median.
- The third heuristic algorithm selects the node to be the center whose shortest path tree spanning all members has the least diameter. Wall defined the diameter of a tree as the maximum delay between any two nodes in the tree.

In his dissertation, Wall also compared the delays along shared trees to the delay if source-specific shortest path trees were used instead. He established several upper bounds on the average and maximum delays that can be achieved by shared trees. His most important result is that if the shortest path tree of a randomly chosen center node is used as a shared tree, then the maximum delay between any two nodes along that tree is at most two times the maximum delay achieved if source-specific trees were used instead.

Finally, Wall proposed a distributed implementation scheme for his three heuristics. This scheme consists of two phases. First, each node in the network computes its criterion (maximum delay, average delay, or diameter) based on the information it has. Then each node broadcasts its criterion value to all other nodes. The values are compared distributedly and the node with best criterion is selected to be the center. To implement his scheme efficiently, Wall proposed that when a node receives the criterion values from two or more other nodes, that node should compare all these values and only continue broadcasting the best value it received so far. In Wei and Estrin's work [108, 109], they studied the tradeoffs between source-specific multicast trees and shared multicast trees using simulation. A shared tree was compared to the

corresponding source-specific trees based on the maximum end-to-end delay, the average end-to-end delay, the tree costs, and the resulting traffic concentration. They defined maximum traffic concentration as the number of traffic streams from different sources which traverse the same link. The authors used both real networks and random networks of up to 200 nodes in their simulations. Each link had a delay and a cost. Several shared tree algorithms and source-specific algorithms were studied. Some algorithms optimized cost while others optimized delay.

Wei and Estrin's simulation results showed that source-specific least-delay trees achieve, on the average, up to 20% smaller maximum delays and up to 30% less traffic concentration than shared trees. However, the cost of a shared trees was approximately 10% less than that of a source-specific least-delay tree. The authors also proposed algorithms that restrict the choice of the multicast centers to the set of multicast group members, and they showed that this restriction does not significantly affect the performance of the resulting shared trees. In [109], Shukla et al. proposed a protocol for constructing multicast trees in asymmetrically loaded networks. Their protocol allows the use of either source-specific trees or shared trees. The authors defined the cost of a link as a function of both the link's delay and its utilization. They proposed a center selection protocol for shared multicast trees based on a tournament. A simplified version of the tournament works as follows. Each receiver is paired with a source in decreasing order of distance. The node at the middle of the shortest path between each pair is the winner of that pair. All winners are then paired together, and the next group of winners is computed, and so on until only one winning node remains. This node is the center of the shared tree. The authors also proposed a shared tree routing algorithm to be applied after selecting the center. Unlike PIM and CBT, the resulting shared trees are completely forward shortest path trees. However, the term "tree" is not appropriate to describe the output of the proposed routing algorithm. Suppose we are given a node n that is both a source and a receiver and also given a center c . In an asymmetric network the forward shortest path from the source n to the center c is not necessarily the same as the forward shortest path from the center c to the receiver n , since the two paths are not necessarily identical, but their end points are the same, their superposition may result in a loop. The authors attempt to include both paths in their routing structure, and therefore

the result may contain loops, and hence it may not always be a tree. The authors' experience indicates that constructing a shared tree in asymmetric networks and trying to optimize it in both directions, from all sources towards the center and from the center towards all receivers, is very complicated.

Next, we studied the previous work by Calvert, Zegura, and Donahoo. In [20], the authors compared different algorithms for selecting a center with respect to their effect on bandwidth and delay. They used simulation over random networks in their study. They concluded that there is no single best algorithm for selecting a center. Trade-offs between performance (bandwidth and delay) and the required information must be considered when choosing a center selection algorithm. The authors also studied the effects of center selection algorithms on traffic concentration. They showed that when center selection algorithms, which distribute the centers uniformly over all nodes in the network, are used, the traffic concentration resulting from shared trees is as good as the traffic concentration resulting from source-specific trees. In another paper, Donahoo and Zegura [34], proposed an algorithm that permits dynamic center migration in order to efficiently support multicast sessions in which sources and destination are allowed to join and leave dynamically.

Distributed center selection protocols were proposed in [97] Thaler and Ravishankar proposed two distributed center selection protocols and two versions of each protocol. The two versions differ in the amount of information each of them uses for its computation. Either one of the proposed protocols can be applied to select more than one center for a given multicast session. However, these protocols do not attempt to distribute the centers evenly throughout the network. The proposed protocols allow the centers to migrate from one node to another dynamically as the group membership changes or the load on the network changes. The authors evaluated their algorithms and most of the previously proposed algorithms using simulation over random networks. In their simulations, they selected only one center for each multicast session. The authors also compared the amount of information required at each node for distributed implementation of the previously proposed algorithms.

2.5 Fault-tolerant in Multicast Network

The issue of providing fault-tolerance in Multicast Network has become a problem of growing importance. This motivates the need for survivable networks in multicast environment, the network which have the capability to tolerate or detect and recover from faults. Survivable networks can be classified into two broad categories: (i) protection based networks and (ii) restoration based networks. In protection based networks, dedicated protection mechanisms, are provided to cope up with faults. These networks use forward recovery approach. It is an example of a protection mechanism in which multiple copies of a packet (or enough redundant information to recover from the loss of a packet) are sent simultaneously over disjoint paths to mask the effect of faults. The approach used in a unicast network is very similar to the multicast network. The advantage of this approach is that it transparently handles faults without service disruption. However, it incurs extra cost in terms of the extra bandwidth used to achieve fault-tolerance. Traffic dispersion is a mechanism in which the traffic from a source node is dispersed along multiple paths to a destination(s).

In restoration based networks, on the occurrence of a fault, an attempt is made to acquire to resources necessary for restoring the channel from fault. These networks use detect and recovery approach. The detect and recovery approach can be further classified into two categories:

- Reactive methods, wherein, a new link or channel is established upon detection of a fault. This has low overhead in the absence of faults, but has more restoration time.
- Spare resource reservation methods, wherein, to avoid/minimize resource shortage during recovery, spare resources are reserved a priori.

While there has been a lot of research in fault-tolerant unicast communication, fault-tolerant multicast communication has received less attention. While the idea of dispersity routing or multipath routing has been around for many years, the traditional uses of dispersity routing have been at the physical layer of network communication. In

[9], a dispersity scheme was proposed for fault-tolerant real-time channels. Some others use dispersity schemes to provide multiple paths between a pair of source and destination nodes; one of the paths is used as the primary paths under normal conditions, and others are used as backup paths if the primary path fails. In chapter 3, we will report on how CBT originally deals with fault. In this dissertation, we complement the previous work by addressing issues related to enhancing the CBT protocol with the fault-tolerant capability while improving the overall protocol end-to-end delay performance.

2.6 Conclusions

In this chapter, we classified the multicast routing algorithms into different categories based on the problems they address. Then we presented important criteria to be considered when summarizing the features of a multicast routing algorithm. This chapter was dedicated to surveying previous work on multicast routing algorithms. However, different researchers have made different assumptions when evaluating the performance of the algorithms they proposed. So in our survey, the assumptions of each researcher must be understood to realize the different characteristics of each multicast routing algorithms.

The objective of this dissertation is to study the fault-tolerant multicast routing algorithms for real-time applications with end-to-end delay bound. A number of delay-constrained multicast routing algorithms have been proposed during the past few years. But the focus was on the source-specific routing tree, whereby ours focuses on shared tree which is more appropriate to the sparse-like network such as the Internet. Besides, the shared tree can scale better than the source-specific tree despite its disadvantages on the end-to-end delay bound. While performing our survey of multicast routing algorithms, we noticed that the concept of shared tree can work well with the real-time traffic if some modification to the shared tree has been made. In addition, with the concept of pre-computed backup off-line, it is possible to enhance the shared multicast tree with the fault-tolerant capability. This motivates us to study both cases in more detail. In chapter 3 we study CBT concept extensively and later propose our CBT modification that suits the real-time traffic in chapter 4. We then study the problems of

fault-tolerance in chapter 5 and propose our approach to further enhance our protocol with disjoint backups in chapter 6.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

CHAPTER 3

CBT – THE PROTOCOL

3.1 Introduction

The Core Based Tree (CBT) multicast architecture differs quite considerably from the existing IP multicast architecture in that it utilizes a single, shared delivery tree that spans a group's receivers. Multicast data is sent and received over the same delivery tree, irrespective of the source. A Core Based Tree involves having a single node, or router, which acts as a core of the tree (with additional cores for robustness), from which branches emanate. These branches are made up of other routers, so-called non-core router, and the core. A router at the end of a branch is known as a leaf router on the tree. The core router need not be topologically centered between the nodes on the tree, since multicasts vary in nature, and correspondingly, so should the form of a group's delivery tree. CBT is unique in that it allows the multicast tree to be built to reflect the nature of the application.

CBT takes full advantage of various aspects of the existing multicast infrastructure, such as class D IP addresses, used for identifying multicast groups, and the Internet Group Management Protocol (IGMP), which is used by multicast routers to establish group member presence of directly-connected subnetworks. A class D address identifies a single host group. The class D address space is separate portion of the IP address space, defined to be between 224.0.0.0 and 239.255.255.255 inclusive. A small number of class D, or group, addresses are reserved for use by various routing protocols, such as 224.0.0.1- the all-systems address, to which all multicast-capable UNIX end-systems are permanently subscribed.

In CBT, a "core router" (or just "core") is a router which acts as a "meeting point" between a sender and group receivers. The term "rendezvous point (RP)" is used equivalently in some contexts. A core router need not be configured to know it is a core router.

A router that is part of a CBT distribution tree is known as an “on-tree” router. An on-tree router maintains active state for the group. We refer to a broadcast interface as any interface that supports multicast transmission.

An “upstream” interface (or router) is one which is on the path towards the group’s core router with respect to this interface (or router). A “downstream” interface (or router) is one which is on the path away from the group’s core router with respect to this interface (or router). Other terminology is introduced in its context throughout the chapter.

Whenever a host wishes to subscribe to a particular group, it sets its network interface so as to receive all packets whose destination address corresponds to a particular class D address. All end-systems wishing to participate in multicast must have a directly connected multicast-capable router. This is true both for CBT and PIM, and older scheme like DVMRP.

IGMP is the protocol implemented in hosts and routers on LANs to monitor multicast group presence on a subnetwork. One router per LAN is elected as membership interrogator. This election is implicit in the IGMP protocol and happens at start-up time. At fixed intervals the elected interrogator sends a non-group-specific membership query to the all-systems multicast group. Hosts receiving this query do not respond immediately, but rather randomise their response over a ten second interval. On expiry of this interval a host sends a group membership report, once for each group it is affiliated to and addressed to the corresponding group. All local multicast routers receive this report. If a group report arrives at a host before its own response interval expires, the corresponding membership report is cancelled at the receiving host. In this way, the membership interrogating router learns of subnetwork group presence, and subnetwork bandwidth consumption due to membership reporting is minimized.

This chapter describes the CBT protocol in detail. The chapter is organized as follows. CBT functional overviews are described in section 3.2, followed by the protocol specification details in section 3.3. In section 3.4, we discuss the protocol overviews and its functionalities. We describe the process of setting up a core based delivery tree, as is typically instigated by one group member – the so – called *group initiator*. We also describe how a new group member joins an existing CBT delivery tree, and how

branches of a tree are removed when receivers (directly connected to such branches by a CBT router) leave the group. We follow with an extensive description of tree maintenance issues, showing how a CBT delivery tree adapts to router, and core failure. We go on to present some simple heuristics for core placement, and so-called *designated router (DR)* election on a multi-access subnetwork. In section 3.5, we describe the CBT's packet and message types in detail. Section 3.6 focuses on core router discovery and discusses the "bootstrap" mechanism. We give a brief discussion on node failure in section 3.7, followed by discussion on loop detection in section 3.8. We also show how a loop-free tree is formed despite the possible presence of transient loops in a CBT router's database. The chapter concludes with section 3.9.

A core based tree is shown in figure 3.1. Only one core is shown for demonstration purpose. The dotted arrowed lines indicate how a CBT multicast packet, sent from a non-member sender, spans a CBT tree

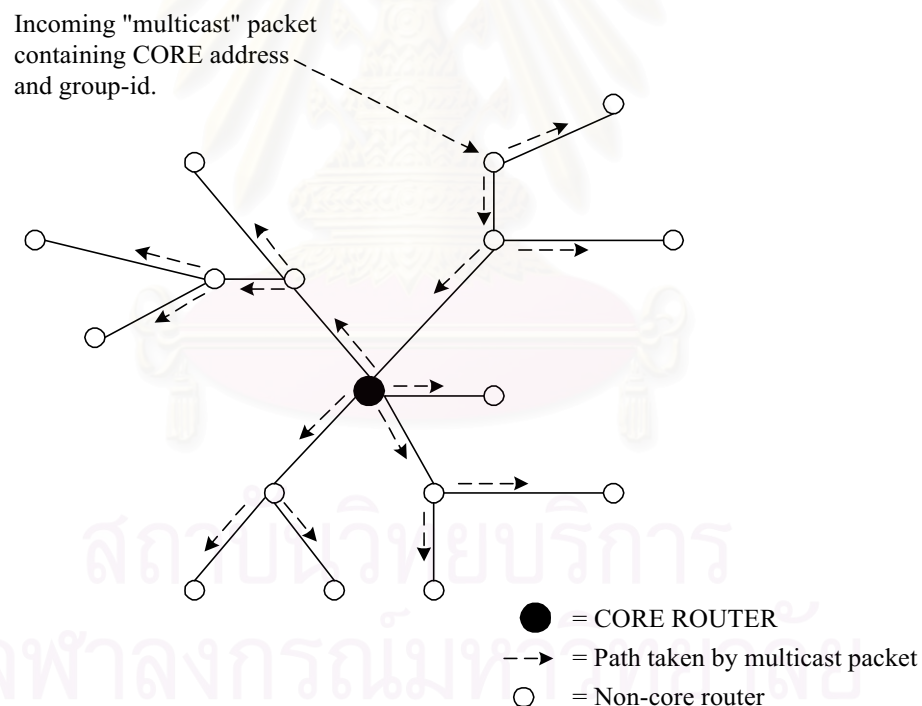


Figure 3.1: A CBT multicast delivery tree

The CBT protocol is built so as to reflect the CBT architecture, just described. This architecture allows for the enhancement of the scalability of the multicast algorithm, in terms of group-specific state maintained in the network, particularly for the case

where there are many active senders in a particular group. The CBT architecture offers an improvement in scalability over existing source-specific tree by a factor of the number of active sources.

The reason for the scalability improvement is because the forwarding of multicasts over a shared tree does not depend on the source of those multicasts, unlike source-specific tree schemes. Routers that comprise a source-rooted maintain source-specific group information. Thus, if within some timeframe, all members of a group become active senders or send simultaneously, a mesh of trees results, with network routers having to keep state for each source tree. Figure 3.2 illustrates the mesh of multicast delivery tree.

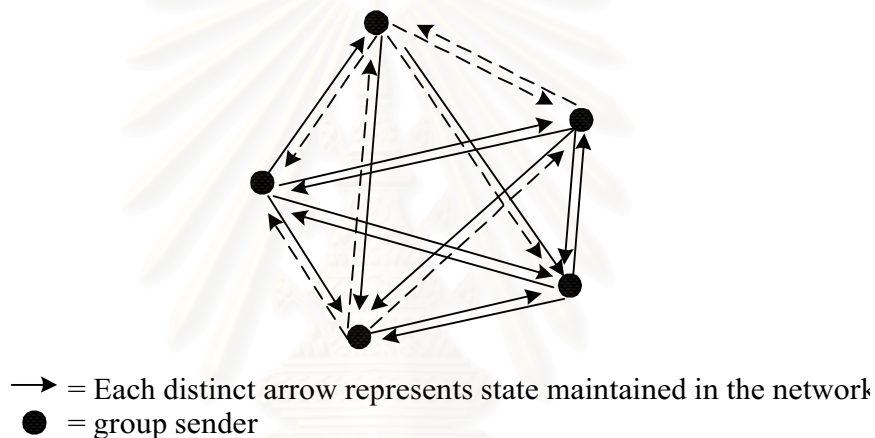


Figure 3.2: Mesh of multicast delivery trees.

Therefore, a core-based architecture allows us to significantly improve the overall scalability of $S \times N$ we have in the source-based tree architecture, to just N . This is the result of having just one multicast tree per multicast group as opposed to one tree per (source, multicast group) pair.

It is also interesting to note that routers between a non-member sender and the CBT delivery tree need no knowledge of the multicast tree (i.e. multicast group) whatsoever in order to forward CBT multicasts, since packets from non-member senders are unicast towards the core. This two-phase routing approach is unique to the CBT architecture. One such application that can take advantage of this two-phase routing is resource discovery, whereby a resource, for example, a replicated database, is distributed in different locations throughout the Internet. The databases in the different

locations make up a single multicast group, linked by a CBT tree. A client need only know the address of (one of) the cores for the group in order to send (unicast) a request to it. Such a request would not span the tree in this case, but would be answered by the first tree router encountered, making it quite likely that the request is answered by the “nearest” server. This is an example of anycasting. The other main driving the CBT approach is that it is unicast routing protocol independent, meaning that the correct operation of the multicast protocol is not linked to the existence of a particular underlying unicast protocol. For example, DVMRP is a multicast protocol that relies on such features. We will discuss the functional overviews of the CBT next.

3.2 CBT Functional Overview

The CBT protocol is designed to build and maintain a shared multicast distribution tree that spans only those networks and links leading to interested receivers.

To achieve this, a host first expresses its interest in joining a group by multicasting an IGMP host membership report across its attached link. On receiving this report, a local CBT aware router invokes the tree joining process by generating a JOIN_REQUEST message, which is sent to the next hop on the path towards the group’s core router. This join message must be explicitly acknowledged (JOIN_ACK) either by the core router itself, or by another router that is on the path between the sending router and the core, which itself has already successfully joined the tree.

The join message sets up transient join state in the routers it traverses, and this state consists of <group, incoming interface, outgoing interface>. “Incoming interface” and “outgoing interface” may be “previous hop” and “next hop”, respectively, if the corresponding links do not support multicast transmission. “Previous hop” is taken from the incoming control packet’s IP source address, and “next hop” is gleaned from the routing table - the next hop to the specified core address. This transient state eventually times out unless it is “confirmed” with a join acknowledgment (JOIN_ACK) from upstream. The JOIN_ACK traverses the reverse path of the corresponding join message, which is possible due to the presence of the transient join state. Once the acknowledgement reaches the router that originated the join message, the new receiver can receive traffic sent to the group.

Loops cannot be created in a CBT tree because there is only one active core per group, and tree building/maintenance scenarios which may lead to the creation of tree loops are avoided. For example, if a router's upstream neighbor becomes unreachable, the router immediately "flushes" all of its downstream branches, allowing them to individually rejoin if necessary. Transient unicast loops do not pose a threat because a new join message that loops back on itself will never get acknowledged, and thus eventually times out.

The state created in routers by the sending or receiving of a JOIN_ACK is bi-directional - data can flow either way along a tree "branch", and the state is group specific - it consists of the group address and a list of local interfaces over which join messages for the group have previously been acknowledged. There is no concept of "incoming" or "outgoing" interfaces, though it is necessary to be able to distinguish the upstream interface from any downstream interfaces. In CBT, these interfaces are known as the "parent" and "child" interfaces, respectively. A router is not considered "on-tree" until it has received a JOIN_ACK for a previously sent JOIN_REQUEST.

With regards to the information contained in the multicast forwarding cache, on link types not supporting native multicast transmission an on-tree router must store the address of a parent and any children. On links supporting multicast however, parent and any child information is represented with local interface addresses (or similar identifying information, such as an interface "index") over which the parent or child is reachable.

Data from non-member senders must be encapsulated (IP-in-IP) by the first-hop router, and is unicast to the group's core router. Consequently, no group state is required in the network between the first hop router and the group's core. On arriving at the core router, the data packet's router encapsulating header is removed and the packet is disseminated over the group shared tree as described below.

When a multicast data packet arrives at a router, the router uses the group address as an index into the multicast forwarding cache. A copy of the incoming multicast data packet is forwarded over each interface (or to each address) listed in the entry except the incoming interface.

Each router that comprises a CBT multicast tree, except the core router, is responsible for maintaining its upstream link, provided it has interested downstream receivers, i.e. the child interface list is not NULL. A child interface is one over which a member host is directly attached, or one over which a downstream on-tree router is attached. This “tree maintenance” is achieved by each downstream router periodically sending a CBT “keepalive” message (ECHO_REQUEST) to its upstream neighbor, i.e. its parent router on the tree. One keepalive message is sent to represent entries with the same parent, thereby improving scalability on links which are shared by many groups. On multicast capable links, a keepalive is multicast to the “all-CBT-routers” group (IANA assigned as 224.0.0.15); this has a suppressing effect on any other router for which the link is its parent link. If a parent link does not support multicast transmission, keepalives are unicast.

The receipt of a keepalive message over a valid child interface prompts a response (ECHO_REPLY), which is either unicast or multicast, as appropriate. The ECHO_REPLY message carries a list of groups for which the corresponding interface is a child interface.

It cannot be assumed all of the routers on a multi-access link have a uniform view of unicast routing; this is particularly the case when a multi-access link spans two or more unicast routing domains. This could lead to multiple upstream tree branches being formed (an error condition) unless steps are taken to ensure all routers on the link agree which is the upstream router for a particular group. CBT routers attached to a multi-access link participate in an explicit election mechanism that elects a single router, the designated router (DR), as the link’s upstream router for all groups. Since the DR might not be the link’s best next-hop for a particular core router, this may result in join messages being re-directed back across a multi-access link. If this happens, the re-directed join message is unicast across the link by the DR to the best next-hop, thereby preventing a looping scenario. This re-direction only ever applies to join messages. Whilst this is suboptimal for join messages, which are generated infrequently, multicast data never traverses a link more than once (either natively, or encapsulated).

In all but the exception case described above, all CBT control messages are multicast over multicast supporting links to the “all-CBT- routers” group, with IP TTL 1.

The IP source address of CBT control messages is the outgoing interface of the sending router. The IP destination address of CBT control messages is either the “all-CBT-routers” group address, or a unicast address, as appropriate. All the necessary addressing information is obtained by on-tree routers as part of tree set up.

If CBT is implemented over a tunneled topology, when sending a CBT control packet over a tunnel interface, the sending router uses as the packet's IP source address the local tunnel end point address, and the remote tunnel end point address as the packet's IP destination address.

3.3 Protocol Specification Details

Details of the CBT protocol are presented in the context of a single router implementation.

3.3.1 CBT HELLO Protocol

The HELLO protocol is used to elect a designated router (DR) on broadcast-type links. It is also used to elect a designated border router (BR) when interconnecting a CBT domain with other domains. Alternatively, the designated BR may be elected as a matter of local policy.

A router represents its status as a link's DR by setting the DR-flag on that interface; a DR flag is associated with each of a router's broadcast interfaces. This flag can only assume one of two values: TRUE or FALSE. By default, this flag is FALSE.

A network manager can preference a router's DR eligibility by optionally configuring an HELLO preference, which is included in the router's HELLO messages. Valid configuration values range from 1 to 254 (decimal), 1 representing the “most eligible” value. In the absence of explicit configuration, a router assumes the default HELLO preference value of 255. The elected DR uses HELLO preference zero (0) in HELLO advertisements, irrespective of any configured preference. The DR continues to use preference zero for as long as it is running.

HELLO messages are multicast periodically to the all-CBT- routers group, 224.0.0.15, using IP TTL 1. The advertisement period is [HELLO_INTERVAL] seconds.

HELLO messages have a suppressing effect on those routers which would advertise a “lesser preference” in their HELLO messages; a router resets its [HELLO_INTERVAL] if the received HELLO is “better” than its own. Thus, in steady state, the HELLO protocol incurs very little traffic overhead.

The DR election winner is that which advertises the lowest HELLO preference, or the lowest-addressed in the event of a tie.

The situation where two or more routers attached to the same broadcast link are advertising HELLO preference 0 should never arise. However, should this situation arise, all but the lowest addressed zero advertising router relinquishes its claim as DR immediately by unsetting the DR flag on the corresponding interface. The relinquishing router(s) subsequently advertise their previously used preference value in HELLO advertisements.

- Sending HELLOs

When a router starts up, it multicasts two HELLO messages over each of its broadcast interfaces in succession. The DR flag is initially unset (FALSE) on each broadcast interface. This avoids the situation in which each router on a multi-access subnet believes it is the DR, thus preventing the multiple forwarding of join-requests should they arrive during this start up period. If no “better” HELLO message is received after HOLDTIME seconds, the router assumes the role of DR on the corresponding interface.

A router sends an HELLO message whenever its [HELLO_INTERVAL] expires. Whenever a router sends an HELLO message, it resets its hello timer.

- Receiving HELLOs

A router does not respond to an HELLO message if the received HELLO is “better” than its own, or equally preference but lower addressed.

A router must respond to an HELLO message if that received is lesser preference (or equally preference but higher addressed) than would be sent by this router over the same interface. This response is sent on expiry of an interval timer which

is set between zero (0) and [HOLDTIME] seconds when the lesser preference HELLO message is received.

3.3.2 JOIN_REQUEST Processing

A JOIN_REQUEST is the CBT control message used to register a member host's interest in joining the distribution tree for the group.

- Sending JOIN_REQUESTs

A JOIN_REQUEST can only ever be originated by a leaf router, i.e. a router with directly attached member hosts. This join message is sent hop-by-hop towards the core router for the group. The originating router caches <group, NULL, upstream interface> state for each join it originates. This state is known as "transient join state". The absence of a "downstream interface" (NULL) indicates that this router is the join message originator, and is therefore responsible for any retransmissions of this message if a response is not received within [RTX_INTERVAL]. It is an error if no response is received after [JOIN_TIMEOUT] seconds. If this error condition occurs, the joining process may be re-invoked by the receipt of the next IGMP host membership report from a locally attached member host.

Note that if the interface over which a JOIN_REQUEST is to be sent supports multicast, the JOIN_REQUEST is multicast to the all-CBT-routers group, using IP TTL 1. If the link does not support multicast, the JOIN_REQUEST is unicast to the next hop on the unicast path to the group's core.

- Receiving JOIN_REQUESTs

On broadcast links, JOIN_REQUESTs which are multicast may only be forwarded by the link's DR. Other routers attached to the link may process the join (see below). JOIN_REQUESTs which are multicast over a point-to-point link are only processed by the router on the link which does not have a local interface corresponding to the join's network layer (IP) source address. Unicast JOIN_REQUESTs may only be processed by the router which has a local interface corresponding to the join's network layer (IP) destination address.

With regard to forwarding a received JOIN_REQUEST, if the receiving router is not on-tree for the group, and is not the group's core router, and has not already forwarded a join for the same group, the join is forwarded to the next hop on the path towards the core. The join is multicast, or unicast, according to whether the outgoing interface supports multicast. The router caches the following information with respect to the forwarded join: <group, downstream interface, upstream interface>. Subsequent JOIN_REQUESTs received for the same group are cached until this router has received a JOIN_ACK for the previously sent join, at which time any cached joins can also be acknowledged.

If this transient join state is not "confirmed" with a join acknowledgment (JOIN_ACK) message from upstream, the state is timed out after [TRANSIENT_TIMEOUT] seconds.

If the receiving router is the group's core router, the join is "terminated" and acknowledged by means of a JOIN_ACK. Similarly, if the router is on-tree and the JOIN_REQUEST arrives over an interface that is not the upstream interface for the group, the join is acknowledged.

If a JOIN_REQUEST for the same group is scheduled to be sent over the corresponding interface (i.e. awaiting a timer expiry), the JOIN_REQUEST is unscheduled. If this router has a cache-deletion-timer [CACHE_DEL_TIMER] running on the arrival interface for the group specified in a multicast join, the timer is cancelled.

3.3.3 JOIN_ACK Processing

A JOIN_ACK is the mechanism by which an interface is added to a router's multicast forwarding cache; thus, the interface becomes part of the group distribution tree.

- Sending JOIN_ACKs

The JOIN_ACK is sent over the same interface as the corresponding JOIN_REQUEST was received. The sending of the acknowledgment causes the router to add the interface to its child interface list in its forwarding cache for the group, if it is not already.

A JOIN_ACK is multicast or unicast, according to whether the outgoing interface supports multicast transmission or not.

- Receiving JOIN_ACKs

The group and arrival interface must be matched to a <group, ..., upstream interface> from the router's cached transient state. If no match is found, the JOIN_ACK is discarded. If a match is found, a CBT forwarding cache entry for the group is created, with "upstream interface" marked as the group's parent interface.

If "downstream interface" in the cached transient state is NULL, the JOIN_ACK has reached the originator of the corresponding JOIN_REQUEST; the JOIN_ACK is not forwarded downstream. If "downstream interface" is non-NULL, a JOIN_ACK for the group is sent over the "downstream interface" (multicast or unicast, accordingly). This interface is installed in the child interface list of the group's forwarding cache entry.

Once transient state has been confirmed by transferring it to the forwarding cache, the transient state is deleted.

3.3.4 QUIT_NOTIFICATION Processing

A CBT tree is "pruned" in the direction downstream-to-upstream whenever a CBT router's child interface list for a group becomes NULL.

- Sending QUIT_NOTIFICATIONs

A QUIT_NOTIFICATION is sent to a router's parent router on the tree whenever the router's child interface list becomes NULL. If the link over which the quit is to be sent supports multicast transmission, if the sending router is the link's DR the quit is unicast, otherwise it is multicast.

A QUIT_NOTIFICATION is not acknowledged; once sent, all information pertaining to the group it represents is deleted from the forwarding cache immediately.

To help ensure consistency between a child and parent router given the potential for loss of a QUIT_NOTIFICATION, a total of [MAX_RTX] QUIT_NOTIFICATIONs are sent, each HOLDDTIME seconds after the previous one.

The sending of a quit (the first) also invokes the sending of a FLUSH_TREE message over each downstream interface for the corresponding group.

- Receiving QUIT_NOTIFICATIONs

The group reported in the QUIT_NOTIFICATION must be matched with a forwarding cache entry. If no match is found, the QUIT_NOTIFICATION is ignored and discarded. If a match is found, if the arrival interface is a valid child interface in the group entry, how the router proceeds depends on whether the QUIT_NOTIFICATION was multicast or unicast.

If the QUIT_NOTIFICATION was unicast, the corresponding child interface is deleted from the group's forwarding cache entry, and no further processing is required. If the QUIT_NOTIFICATION was multicast, and the arrival interface is a valid child interface for the specified group, the router sets a cache-deletion-timer [CACHE_DEL_TIMER].

Because this router might be acting as a parent router for multiple downstream routers attached to the arrival link, [CACHE_DEL_TIMER] interval gives those routers that did not send the QUIT_NOTIFICATION, but received it over their parent interface, the opportunity to ensure that the parent router does not remove the link from its child interface list. Therefore, on receipt of a multicast QUIT_NOTIFICATION over a parent interface, a receiving router schedules a JOIN_REQUEST for the group for sending at a random interval between 0 (zero) and HOLDTIME seconds. If a multicast JOIN_REQUEST is received over the corresponding interface (parent) for the same group before this router sends its own scheduled JOIN_REQUEST, it unschedules the multicasting of its own JOIN_REQUEST.

3.3.5 ECHO_REQUEST Processing

The ECHO_REQUEST message allows a child to monitor reachability to its parent router for a group (or range of groups if the parent router is the parent for multiple groups). Group information is not carried in ECHO_REQUEST messages.

- Sending ECHO_REQUESTs

Whenever a router creates a forwarding cache entry due to the receipt of a JOIN_ACK, the router begins the periodic sending of ECHO_REQUEST messages over its parent interface. The ECHO_REQUEST is multicast to the “all-CBT-routers” group over multicast-capable interfaces, unless the sending router is the DR on the interface over which the ECHO_REQUEST is being sent, in which case it is unicast (as is the corresponding ECHO_REPLY).

ECHO_REQUEST messages are sent at [ECHO_INTERVAL] second intervals. Whenever an ECHO_REQUEST is sent, [ECHO_INTERVAL] is reset.

If no response is forthcoming, any groups present on the parent interface will eventually expire [GROUP_EXPIRE_TIME]. This results in the sending of a QUIT_NOTIFICATION upstream, and sends a FLUSH_TREE message downstream for each group for which the upstream interface was the parent interface.

- Receiving ECHO_REQUESTs

If an ECHO_REQUEST is received over any valid child interface, the receiving router schedules an ECHO_REPLY message for sending over the same interface; the scheduled interval is between 0 (zero) and HOLDDTIME seconds. This message is multicast to the “all-CBT-routers” group over multicast-capable interfaces, and unicast otherwise.

If a multicast ECHO_REQUEST message arrives via any valid parent interface, the router resets its [ECHO_INTERVAL] timer for that upstream interface, thereby suppressing the sending of its own ECHO_REQUEST over that upstream interface.

3.3.6 ECHO_REPLY Processing

ECHO_REPLY messages allow a child to monitor the reachability of its parent, and help ensure the group state information is consistent between them.

- Sending ECHO_REPLY messages

An ECHO_REPLY message is sent in response to receiving an ECHO_REQUEST message, provided the ECHO_REQUEST is received over any one of this router’s valid child interfaces. An ECHO_REPLY reports all groups for which the link is its child.

ECHO_REPLY messages are unicast or multicast, as appropriate.

- Receiving ECHO_REPLY messages

An ECHO_REPLY message must be received via a valid parent interface. For each group reported in an ECHO_REPLY, the downstream router attempts to match the group with one in its forwarding cache for which the arrival interface is the group's parent interface. For each successful match, the entry is "refreshed". If however, after [GROUP_EXPIRE_TIME] seconds a group has not been "refreshed", a QUIT_NOTIFICATION is sent upstream, and a FLUSH_TREE message is sent downstream, for the group.

If this router has directly attached members for any of the flushed groups, the receipt of an IGMP host membership report for any of those groups will prompt this router to rejoin the corresponding tree(s).

3.3.7 FLUSH_TREE Processing

The FLUSH_TREE (flush) message is the mechanism by which a router invokes the tearing down of all its downstream branches for a particular group. The flush message is multicast to the "all-CBT-routers" group when sent over multicast-capable interfaces, and unicast otherwise.

- Sending FLUSH_TREE messages

A FLUSH_TREE message is sent over each downstream (child) interface when a router has lost reachability with its parent router for the group (detected via ECHO_REQUEST and ECHO_REPLY messages). All group state is removed from an interface over which a flush message is sent. A flush can specify a single group, or all groups (INADDR_ANY).

- Receiving FLUSH_TREE messages

A FLUSH_TREE message must be received over the parent interface for the specified group, otherwise the message is discarded. The flush message must be

forwarded over each child interface for the specified group. Once the flush message has been forwarded, all state for the group is removed from the router's forwarding cache.

3.4 Protocol Overview

This section describes how CBT works in terms of its functionality, as follows.

3.4.1 CBT Group Initiation

Like any of the other multicast schemes, one user, the *group initiator*, or *group leader*, initiates a CBT multicast group. The procedures involved in initiating and joining a CBT group involves a little more user interaction than current IP multicast schemes, for example, it is necessary to supply information such as desired *group score* so that the group's cores can be selected appropriately within the desired region. The current implementation makes use of three configuration files which correspond to the three currently-defined group scopes: *local*, *national*, and *international*. Each of these files contains an ordered list of cores which are used for all groups selecting that group scope.

Explicit *core rankings* help prevent loops when the core tree is initially set up. It also assists in the tree maintenance process should the tree become partitioned.

Group initiation could be carried out by a network management center, or by some other external means, rather than have a user act as group initiator

3.4.2 Tree Joining Process

Once the cores have been selected by a group's initiator, the group-initiating host sends a special CORE-NOTIFICATION message to each of them, which is acknowledged. The purpose of this message is twofold: firstly, to communicate the identities of all of the cores, together with their rankings, to each of them individually; secondly, to invoke the building of the core backbone. These two procedures follow on one to the other in the order just described. New receivers attempting to join whilst the building of the core backbone is still in progress have their explicit JOIN-REQUEST messages stored by whichever CBT-capable router, involved in the core joining process, is encountered first.

Routers on the core backbone will usually include not only the cores themselves, but intervening CBT-capable routers on the unicast path between them. Once this set up is complete, any pending joins for the same group can be acknowledged.

All the CBT-capable routers traversed by a JOIN-ACKnowledgment change their status to *CBT-non-core routers* for the group identified by group-id. The JOIN-ACK follows the *reverse* of the path traced out by the corresponding JOIN-REQUEST. It is the JOIN-ACK that actually creates a tree branch. The tree branch is a *reverse-shortest path* rooted at the node where the corresponding JOIN-REQUEST originated.

The JOIN-ACK carries the complete core list for the group, which is stored by each of the routers it traverses. Between sending a JOIN-REQUEST and receiving a JOIN-ACK, a router is in a state of *pending* membership. A router that is in the *join pending* state cannot send join acknowledgements in response to other join requests received for the same group, but rather caches them for acknowledgment subsequent to its own join being acknowledged. Furthermore, if a router in the pending state gets a better route after canceling its previous join (this is required to deal with unicast transient loops).

Non-member senders, and new group receivers, are expected to know the address of at least one of the corresponding group's cores in order to send to/join a group. The current specification does not state how this information is gleaned, but it might be obtainable from a directory such as "sd" (the multicast session directory) or from the Domain Name System (DNS).

In accordance with exiting IP multicast schemes, CBT multicasting requires the presence of at least one CBT-capable router per subnetwork for hosts on that subnetwork to utilize CBT multicasting. Only one local router, the *designated router*, is allowed to send to receive from uptree (i.e. the branch leading to/from the core) for a particular group. We therefore make a clear distinction between a *group membership interrogator*-the router responsible for sending IGMP host-membership queries onto the local subnet, and the *designated router*. However, they may not be one and the same. LAN specifics are discussed later on in the chapter.

Once the most appropriate designated router (DR) has been established, i.e. the router that is on the shortest-path to the corresponding core, the new receiver (host)

sends a special CBT report to it, requesting that it join the corresponding delivery tree if it has not already. If it has, then the DR multicasts to the group a notification to that effect back across the subnet. Information included in this notification includes whether the DR was successful in joining the corresponding tree, and *actual* core affiliation. If the local DR has not joined the tree, then it proceeds to send a JOIN-REQUEST and awaits an acknowledgement, at which time the notification, as described above, is multicast across the subnetwork.

3.4.3 Tree Leaving Process

A QUIT-REQUEST is a request by a CBT router to leave a group. A QUIT-REQUEST may be sent by a router to detach itself from a tree if and only if it has no members for that group on any directly attached subnets, and it has received a QUIT-REQUEST on each of its child interfaces for that group (if it has any). The QUIT-REQUEST can only be sent to the parent router. The parent immediately acknowledges the QUIT-REQUEST with a QUIT-ACK and removes that child interface from the tree. Any CBT router that sends a QUIT-ACK in response to receiving a QUIT-REQUEST should itself send a QUIT-REQUEST upstream if the criteria described above are satisfied.

Failure to receive a QUIT-ACK despite several re-transmissions gives the sending router the right to remove the relevant parent interface information, and by doing so, removes itself from the CBT tree for that group.

3.4.4 Tree Maintenance Issues

Robustness features/mechanisms have been built into the CBT protocol as have been deemed appropriate to ensure timely tree re-configuration in the event of a node or core failure. These mechanisms are implemented in the form of *request-response* messages. Their frequency is configurable, with the trade-off being between protocol overhead and timeliness in detecting a node failure, and recovering from that failure.

3.4.5 Core Placement

As it stands, the current implementation of CBT (version 2) uses trial heuristic algorithms for core placement. Careful placement of core(s) no doubt assists in optimizing the

routes between any sender and group members on the tree. Depending on particular group dynamics, such as sender/receiver population, and traffic patterns, it may well be counter-productive to place a core(s) near or at the center of a group. In any event, there exists no polynomial time algorithm that can find the center of a dynamic multicast spanning tree [102].

One suggestion might be that cores be statically configured throughout the Internet—there need only be some relatively small number of cores per backbone network, and the addresses of these cores would be “well-known”.

Alternatively, and possibly more appropriately, any router could become a core when a host on one of its attached subnets wishes to initiate a group. This is particularly attractive for a one-to-many “broadcast” where the sender remains constant, since, if the sender is the core, the multicast tree formed will be a shortest-path spanning tree rooted at the sender.

We have stressed that the placement of a group’s core should positively reflect that group’s characteristics. In the absence of any better mechanism, CBT adopts the “hand-selection” approach to selecting a group’s cores, based on a judgment of what is known about the network topology between the current members.

A fast and scalable algorithm for locating “distribution centers” was used in simulations [91]. This algorithm takes into account network load, and participants’ resource requirements. However, it requires *a priori* knowledge of participants’ locations.

3.4.6 LAN Designated Router

As we have said, there must only ever exist one DR (Designated Router) for any particular group that is responsible for uptime forwarding/reception of data packets. A group’s DR is elected by means of an explicit mechanism. Whenever a host initiates/joins a group, part of the process is for it to send a CBT-DR-SOLICITATION message, addressed to the CBT “all-routers” address, which is a request for the best next-hop router to a specified core.

If the group is being initiated, a DR will almost certainly not be present on the local subnet for the group, whereas if a group is being joined, the DR may or may not be present, depending on whether there exist other group members on the LAN (subnet).

If a DR is present for the specified group, it responds to the solicitation with a CBT-DR-ADVERTISEMENT, which is addressed to the group.

If no DR is present, each CBT router inspects its unicast routing table to establish whether it is the next best-hop to the specified core.

A router which considers itself the best next-hop does not respond immediately with an advertisement, but rather sends a CBT-DR-ADV-NOTIFICATION to the CBT "all-routers" address. This is a precautionary measure to prevent more than one router advertising itself as the DR for the group (it is conceivable that more than one router might think itself as the best next-hop to the core). If router with the lowest address winning the election. The lowest addressed router subsequently advertises itself as DR for the group.

3.4.7 Non-member Sending

For non-member senders wishing to send multicasts beyond the scope of the local subnetwork, the presence of a local CBT-capable router is mandatory. The sending of multicast packets from a non-member host to a particular group is two-phase: the first phase involves a host unicasting the packet from the originating host to one of the group's cores (the destination field of the IP header carries the unicast address of the core". The second phase is the dissemination of the packet by the receiving router to neighboring (adjacent" routers on the corresponding tree. Similarly, when an on-tree neighbor receives the packet, it distributes it in the same fashion.

Before the multicast leaves the originating subnetwork, it is necessary for the local CBT DR to append d CBT header to the packet (behind the IP header), and change the IP destination address field from a multicast address to the unicast address of a core for the group. How does the CBT DR know that this multicast address is associated with a CBT group? The answer is that there must be some form of mapping mechanism, which has information about which group address correspond to CBT

multicast groups. This mechanism maps an IP multicast address to a unicast core address.

Packets sent from a non-member sender will first encounter the corresponding delivery tree either at the addressed core, or hit an on-tree router that is on the shortest-path between the sender and the core. CBT's 2-phase routing is the catalyst behind the CBT solution to "anycasting". So, now should multicast packets reach the correct distribution tree if they are simply sent destined to a multicast address? The answer is that the first-hop multicast router should be incorporated with functionality that distinguishes between CBT multicasts and "other" multicasts. There are two obvious possible ways to implement this functionality in routers: firstly, a separate part of the multicast address space could be set aside for CBT multicast. Whenever a multicast capable router receives a multicast packet, if the group address in the destination field of the IP header falls within a particular range, the router could instigate a lookup in an advertising service such as "sd", which should hold (core, group) mappings. If a group address corresponds to a mapping, then the unicast core address specified in the mapping is placed in the IP header, and the router appends a CBT header behind the IP header, inserting the necessary information into the given CBT header fields.

The second approach does not involve segmenting the group address space, but would instead involve a multicast router performing a lookup for every distinct multicast address in an attempt to find a mapping entry in the session advertising service. If none is found, it is assumed the multicast is to be forwarded using the default multicast protocol operating in the router.

Both schemes have their advantages and disadvantages-segmenting the multicast address space is an administrative burden, whilst having multicast routers perform a lookup for every distinct multicast address has a performance impact. Both schemes require all multicast routers to have a minimum CBT functionality. This enables CBT multicast to remain invisible to the end-systems. No host changes are required for CBT. CBT hosts are simply required to run the CBT application-level software that provides the CBT user group management interface.

3.4.8 Data Packet Forwarding

In this section we describe in more detail how multicast data packets span a CBT tree. CBT uses the *Internet Group Management Protocol (IGMP)* in much the same way as traditional IP schemes, namely to establish group presence on directly-connected subnets, and to exchange CBT routing information. A new IGMP message type has been created for exchanging CBT routing messages. Some slight modifications have been made to IGMP specifically for CBT in order to significantly reduce *leave latency* (although the new version of IGMP can be easily adopted by other multicast protocols).

It has been an important engineering design goal for CBT to be backwards compatible with IP-style multicast. Until the interface with other multicast protocols is clearly defined, CBT routing information is not exchanged with that of any other schemes.

IP-style multicast data packets arriving at a CBT router are checked to see if they originated locally. If not, they are discarded. Otherwise, the local CBT DR for the group first sends a copy of the IP-style packet over any directly-connected subnetworks with group member presence (provided the TTL allows), then appends a CBT header to the packet for forwarding over outgoing tree interfaces.

CBT-style packets arriving at a CBT router are forwarded over tree interfaces for the group, and sent IP-style over any directly-connected subnetworks with group member presence. The conversion from a CBT-style packet to an IP-style packet requires the copying of various fields of the CBT header to the IP header.

The child (ren) or parent of a CBT router may be reachable over a multi-access LAN. This is the case where a subnetwork and a tree branch are one and the same. In this case, the forwarding of the CBT-style packets is achieved with multicast as opposed to unicast. End-systems subscribed to the same group may receive these packet, but they will be discarded, since end-systems will not recognize the upper-layer protocol identifier, i.e. CBT. It was an engineering design decision to *multicast* data packets with a CBT header on multi-access links-the case of unicasting separately from parent to n children is clearly more costly. Multicasting also reduces traffic-when a parent receives a packet from the multi-access link, it does not need to re-send the packet to any of its

other children that may be present on the multi-access link, since they will have received a copy from the child's multicast.

Data arriving at a CBT router is always multicast IP-style onto any directly-connected subnets with group member presence, and only subsequently unicast (multicast on multi-access links) to parent/children with a CBT header.

The primary forwarding rules for CBT-capable routers are simple, and are as follows:

- a data packet is only forwarded if a CBT header is present in the packet.
- a data packet is *only* forwarded by a CBT-capable router if there is a forwarding information base (FIB) entry for the group specified in the CBT header of the data packet, i.e. the CBT router must be on-tree for the group. CBT does *not* forward data packets if they do not have a CBT header. The forwarding router *must* be the designated router for the group on the subnetwork.

Exception: if a data packet *originates* on a directly-connected subnetwork, the local multicast-capable router may be required to append a CBT header, if that router has established that the specified group is a CBT group. In order to forward such a packet, a CBT-capable router need not be on-tree for the specified group.

A FIB entry is shown below.

32-bits	4	4	4	4	4
group identifier	parent addr index	parent vif index	No. of children	Children...	
Group identifier-- analogous to a class-D address Vif--Virtual interface. A physical interface may act as a virtual interface for a tunnel. Several such tunnels can be configured on one physical interface.				child addr index	child vif index
				child addr index	child vif index
				child addr index	child vif index

Figure 3.3: A CBT FIB entry in the author's implementation

The CBT DR for the specified group fills in the CBT and IP headers as follows:

- The multicast group address (group-id) is inserted into the **group-id** field of the CBT header.
- The unicast address of a core router for the corresponding group is placed in the core address field of the CBT header.
- The IP address of the originating host is inserted into the **origin** field of the CBT header.
- The **proto** field of the CBT header is set to identify the upper-layer (transport) protocol.
- The **tll** field of the CBT header is set to the value reflected in the packet's IP header (if the packet originated locally).
- The **on-tree** field of the CBT header is set (provided this CBT router is on-tree for the specified group). It is left unset otherwise.
- The source address field of the IP header is set to the unicast address of the originating host (the IP source address changes as the CBT-style packet is passed router-to-router on a CBT tree).
- The destination field of the IP header is set to the unicast address of the on-tree neighbor (set to group address if more than one neighbor is reachable over the same interface).
- The protocol field of the IP header is set to the CBT protocol value.
- The TTL value of the IP header is set to the value specified in the packet from the original source.

This packet is now ready for sending. Once this packet arrives at a CBT router, the packet is “reverse-engineered” (using the information carried in the CBT header) to produce an IP-style multicast for sending on directly-connected subnets with group presence. For forwarding CBT-style over on-tree interfaces, only the IP header need be manipulated, as would be expected (i.e. source address, destination address, TTL value).

What happens to a multicast packet *originated* on a subnetwork is illustrated in figure 3.4

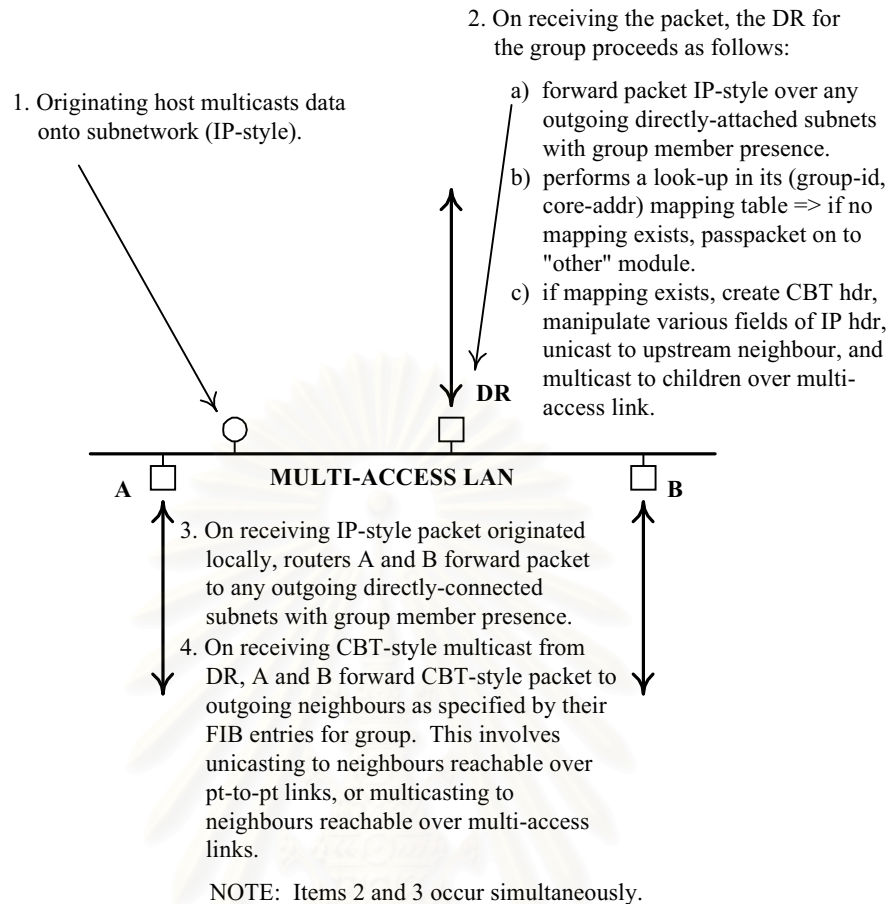


Figure 3.4: CBT Data Packet Forwarding on a LAN (originating case)

Figure 3.5 illustrates how a CBT router handles an incoming CBT data packet, and generates an IP-style multicast.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

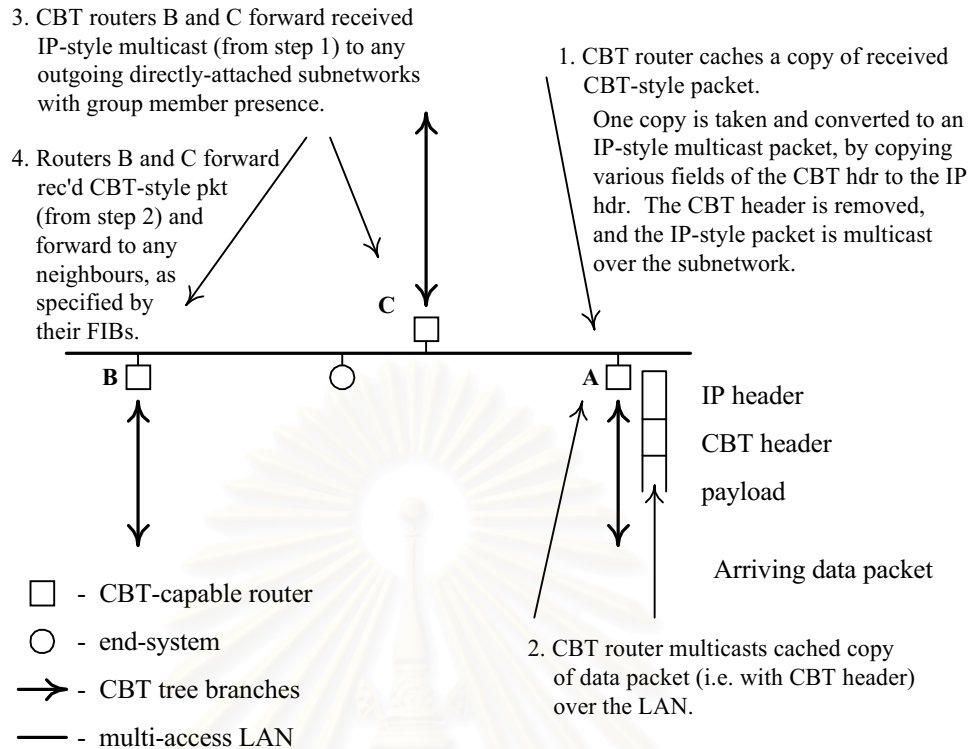


Figure 3.5: CBT Data Packet Forwarding on a LAN (receiving case)

3.4.9 Lower Group Leave Latency

One of the design goals of CBT was to modify the *Internet Group Management Protocol (IGMP)* to reduce group leave latency, i.e. the time between the last claim to a group on a particular subnet being relinquished, and the time group traffic is no longer forwarded onto that subnet. Using DVMRP as an example, this takes around four and a half minutes. The reason leave latency is currently so long is because this is the shortest time considered reasonable for multicast routers to implicitly deduce, from the absence of group membership report messages, that there are no longer any claims to a particular group on a subnet.

CBT introduces an explicit *IGMP group leave* message to drastically reduce leave latency. It was considered an important design goal since, over the last few years, multicast has been adopted as the preferred transport mechanism for many high-bandwidth applications, including multimedia applications. Now, even comparatively resource-rich LANs are not immune to the congestion problems usually only witnessed

on slower, wide-area links. It is therefore essential that, once there are no longer any receivers on a subnet, the corresponding traffic flow should cease as soon as possible thereafter.

As we have said, lower leave latency is possible through the introduction of a new IGMP message type: IGMP-HOST-MEMBERSHIP-LEAVE. When host relinquishes its last claim to a particular group membership, it multicasts a IGMP-HOST-MEMBERSHIP-LEAVE message to the “all-CBT-routers” address. This message contains the multicast address of the group being relinquished. Irrespective of whether any of the receiving CBT routers is the subnet’s membership interrogator, a CBT router responds to the LEAVE message by sending an IGMP-HOST-MEMBERSHIP-QUERY to the “all-systems” multicast address. Only one CBT router actually responds with a query, since the responses are randomized over an interval of five seconds, and the receipt of a query cancels out a CBT router’s pending query.

From the moment the LEAVE message arrives at a CBT router, a timer starts running for the group being relinquished, and is only cancelled if, subsequent to the query, a report arrives before the timeout period, which currently is around 12 seconds. This is comprised of the 10 seconds randomized response interval of hosts after hearing a query, plus a 2 second safety margin.

Leave latency *could* be further reduced if hosts’ randomized response intervals were shortened to, say, 5 seconds. The trade-off then is between increased protocol over-head/bandwidth consumption of more frequent IGMP messages, and a shorter group leaving time.

3.5 CBT Packet Formats and Message Types

CBT packets travel in IP datagrams. For clarity, we distinguish between three types of CBT packet: those directly concerned with tree building, and re-configuration-so called *primary maintenance messages*; those concerned with general tree maintenance-so called *auxiliary maintenance messages*; those carrying multicast *data*.

All of the above message types are encapsulated in a *CBT header*. Primary and auxiliary maintenance messages are additionally encapsulated in a *CBT control header*. All packets then, data and control, carry the CBT header, but control packets only

require the parsing of four of the fields of the CBT header. The reason a CBT header is present even partially in control packets is partly administrative-it requires the definition of just one protocol number. This protocol number was set to be 7. Control packets therefore, travel inside (a portion of) a CBT header, and are identifiable as such by the contents of the TYPE field in the CBT header (TYPE can only be “control” or “data”).

- CBT Header Format

The CBT header is illustrated below:

0	7	8	15	16	23	24	31
Vers	Unused		type	hdr length		protocol	
checksum				IP TTL		on-tree	unused
group identifier							
core address							
packet origin							
flow-identifier							
security fields (T.B.D.) * * *							

Figure 3.6: CBT Header

We proceed to describe each of the fields of the CBT header:

- **Vers:** Version number-this release specifies version 1.
- **Type:** indicates whether the payload is data or control information.
- **Hdr length:** length of the header, for purpose of checksum calculation.
- **Protocol:** upper-layer protocol number.
- **Checksum:** the 16-bit one's complement of the one's complement of the CBT header, calculated across all fields.
- **IP TTL:** TTL value gleaned from the IP header where the packet originated. It is decremented each time it traverses a CBT router.
- **On-tree:** indicates whether the packet is on - or - off-tree. Once this field is set (i.e. on-tree), it is non-changing.
- **Group identifier:** multicast group address.
- **Core address:** the unicast address of a core for the group. A core address is always inserted into the CBT header by an originating host, since at any

instant, it does not know if the local DR for the group is on-tree. If it is not, the local DR must unicast the packet to the specified core.

- **Packet origin:** source address of the originating end-system.
 - **Flow-identifier:** value uniquely identifying a previously set up data stream.
 - **Security fields:** these fields (T.B.D) will ensure the authenticity and integrity of the received packet.
- **Control Packet Header Format**

Figure 3.7 shows a CBT control packet header. The individual fields are described below. It should be noted that the contents of the fields beyond “group identifier” are empty in some control messages:

- **Vers:** Version number-this release specifies version2.
- **Type:** indicates control message type.

0	7	8	15	16	23	24	31
Vers	unused	type	code	unused			
header length				checksum			
group identifier							
packet origin							
core address							
Core #1							
Core #2							
Core #3							
Core #4							
Core #5							
Resource Reservation fields (T.B.D.) *****							
security fields (T.B.D.) *****							

Figure 3.7: CBT Control Packet Header

- **Code:** indicates sub-code of control message type.
- **Header length:** length of the header, for purpose of checksum calculation.
- **Checksum:** the 16-bit one’s complement of the one’s complement of the CBT control header, calculated across all fields.

- **Group identifier:** multicast group address.
 - **Core address:** desired/actual core affiliation of control message.
 - **Core # Z:** Maximum of 5 core addresses may be specified for any one group.
 - **Resource reservation fields:** these fields (T.B.D.) are used to reserve resources as part of the CBT tree set up procedure.
 - **Security fields:** these fields (T.B.D.) ensure the authenticity and integrity of the received packet.
- **Primary Maintenance Message Types**

There are six types of CBT primary maintenance message, namely:

- **JOIN-REQUEST:** invoked by an end-system, generated and sent (unicast) by a CBT router to the specified core address. Its purpose is to establish the sending CBT router as part of the corresponding delivery tree.
- **JOIN-ACK:** an acknowledgement to the above. The full list of core addresses is carried in a JOIN-ACK, together with the actual core affiliation (the join may have been terminated by an on-tree router on its journey to the specified core, and the terminating router may or may not be affiliated to the core specified in the original join). A JOIN-ACK traverses the same path as the corresponding JOIN-REQUEST, and it is the receipt of a JOIN-ACK that actually creates a tree branch.
- **JOIN-NACK:** a negative acknowledgement, indication that the tree join process has not been successful.
- **QUIT-REQUEST:** a request, sent from a child to a parent, to be removed as a child to that parent.
- **QUIT-ACK:** acknowledgement to the above. If the parent, or the path to it is down, no acknowledgement will be received within the timeout period. This results in the child nevertheless removing its parent information.
- **FLUSH-TREE:** a message sent from parent to all children, which traverses a complete branch. This message results in all tree interface information

being removed from each router on the branch, possibly because of a re-configuration scenario.

The JOIN-REQUEST has three valid sub-codes, namely JOIN-ACTIVE, RE-JOIN-ACTIVE, and RE-JOIN-NACTIVE.

A JOIN-ACTIVE is sent from a CBT router that has no children for the specified group.

A RE-JOIN-ACTIVE is sent from a CBT router that has at least one child for the specified group.

A RE-JOIN-NACTIVE originally started out as an active re-join, but has reached an on-tree router for the corresponding group. At this point, the router changes the join status to *non-active* re-join and forwards it on its parent branch, as does each CBT router that receives it. Should the router that originated the active re-join subsequently receive the non-active re-join, it must immediately send a QUIT-REQUEST to its parent router. It then attempts to re-join again. In this way the re-join acts as a *loop-detection* packet.

- Auxiliary Maintenance Message Types

There are eleven CBT auxiliary maintenance message types:

- **CBT-DR-SOLICITATION:** a request sent from a host to the CBT “all-routers” multicast address, for the address of the best next-hop CBT router on the LAN to the core as specified in the solicitation.
- **CBT-DR-ADVERTISEMENT:** a reply to the above. Advertisements are addressed to the “all-systems” multicast group.
- **CBT-CORE-NOTIFICATION:** unicast from a group initiating host to each core selected for the group, together with their core ranking. The receipt of this message invokes the building of the *core tree* by all cores other than the highest-ranked (primary core).
- **CBT-CORE-NOTIFICATION-REPLY:** a notification of acceptance to becoming a core for a group, to the corresponding end-system.

- **CBT-ECHO-REQUEST:** once a tree branch is established, this message acts as a “keepalive”, and is unicast from child to parent.
- **CBT-ECHO-REPLY:** positive reply to the above.
- **CBT-CORE-PING:** unicast from a CBT router to a core when a tree router’s parent has failed. The purpose of this message is to establish core reachability before sending a JOIN-REQUEST to it.
- **CBT-PING-REPLY:** positive reply to the above.
- **CBT-TAG-REPORT:** unicast from an end-system to the designated router for the corresponding group, subsequent to the end-system receiving a designated router advertisement (as well as a core notification reply if group-initiating host). This message invokes the sending of a JOIN-REQUEST if the receiving router is not already part of the corresponding tree.
- **CBT-CORE-CHANGE:** group-specific multicast by a CBT router that originated a JOIN-REQUEST on behalf of some end-system on the same LAN (subnet). The purpose of this message is to notify end-systems on the LAN belonging to the specified group of such things as: success in joining the delivery tree; actual core affiliation.
- **CBT-DR-ADV-NOTIFICATION:** multicast to the CBT “all-routers” address, this message is sent subsequent to receiving a CBT-DR-SOLICITATION, but prior to any CBT-DR-ADVERTISEMENT being sent. It acts as a tie-breaking mechanism should more than one router on the subnet think itself the best next-hop to the addressed core. It also prompts an already established DR to announce itself as such if it has not already done so in response to a CBT-DR-SOLICITATION.

3.5.1 CBT Control Packet Formats

CBT control packets are encapsulated in IP. CBT has been assigned IP protocol number 7 by IANA. CBT packet types are summarized as follows:

- CBT Common Control Packet Header

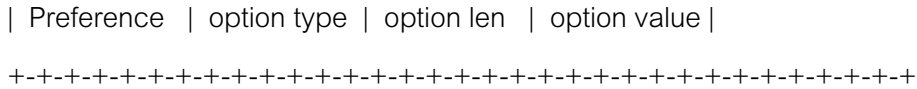


Figure 3.9: HELLO Packet Format

HELLO Packet Field Definitions:

- preference: sender’s HELLO preference.
- option type: the type of option present in the “option value” field. One option type is currently defined: option type 0 (zero) = BR_HELLO; option value 0 (zero); option length 0 (zero). This option type is used with HELLO messages sent by a border router (BR) as part of designated BR election (see [4]).
- option len: length of the “option value” field in bytes.
- option value: variable length field carrying the option value.

JOIN_REQUEST Packet Format

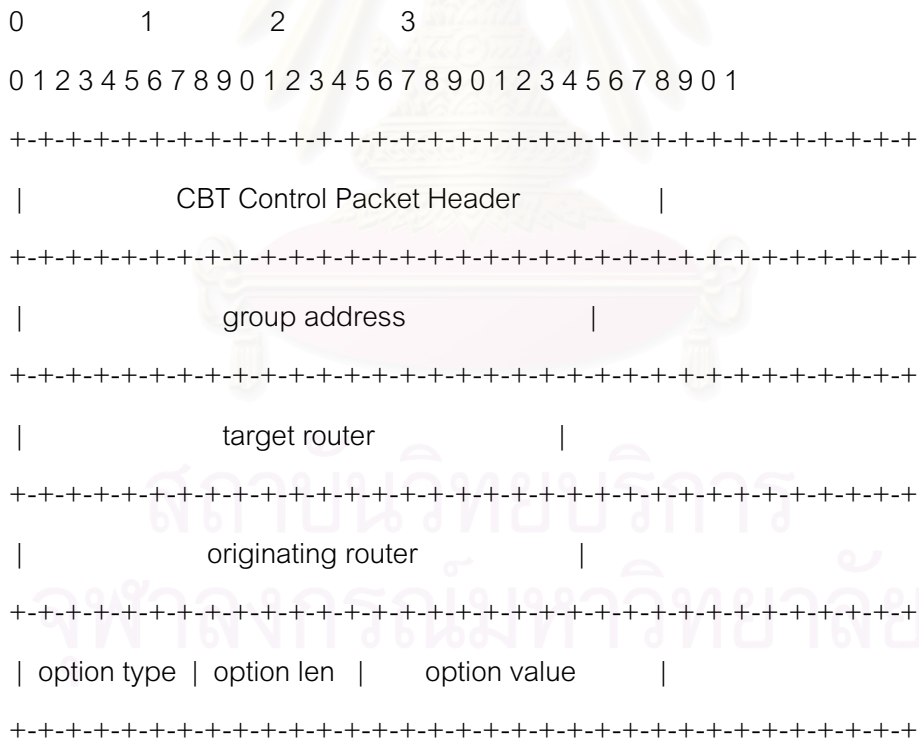


Figure 3.10: JOIN_REQUEST Packet Format

JOIN_REQUEST Field Definitions

- group address: multicast group address of the group being joined. For a “wildcard” join (see [4], this field contains the value of INADDR_ANY.
- target router: target (core) router for the group.
- originating router: router that originated this JOIN_REQUEST.
- option type, option len, option value.

JOIN_ACK Packet Format

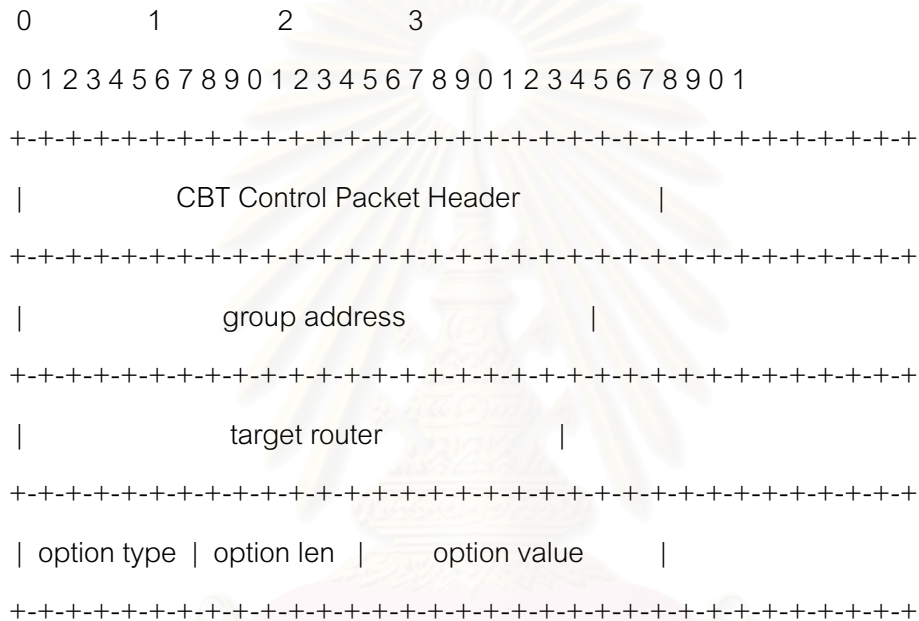
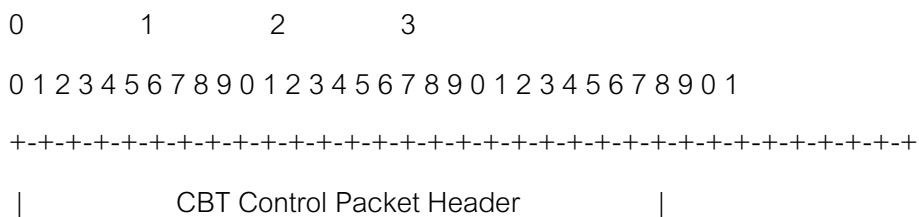


Figure 3.11: JOIN_ACK Packet Format

JOIN_ACK Field Definitions

- group address: multicast group address of the group being joined.
- target router: router (DR) that originated the corresponding JOIN_REQUEST.
- option type, option len, option value.

QUIT_NOTIFICATION Packet Format



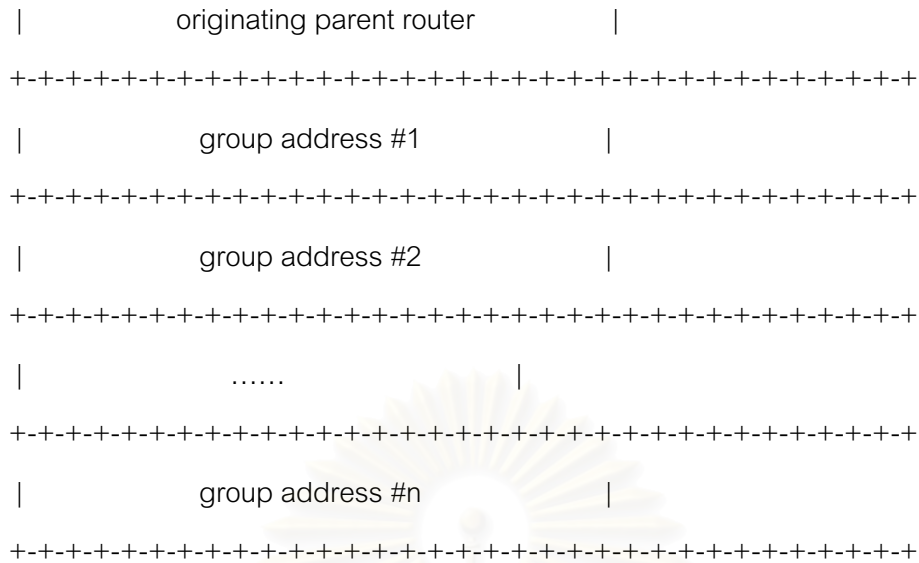


Figure 3.14: ECHO_REPLY Packet Format

ECHO_REPLY Field Definitions

- originating parent router: address of the router originating this ECHO_REPLY.
- group address: a list of multicast group addresses for which this router considers itself a parent router w.r.t. the link over which this message is sent.

FLUSH_TREE Packet Format

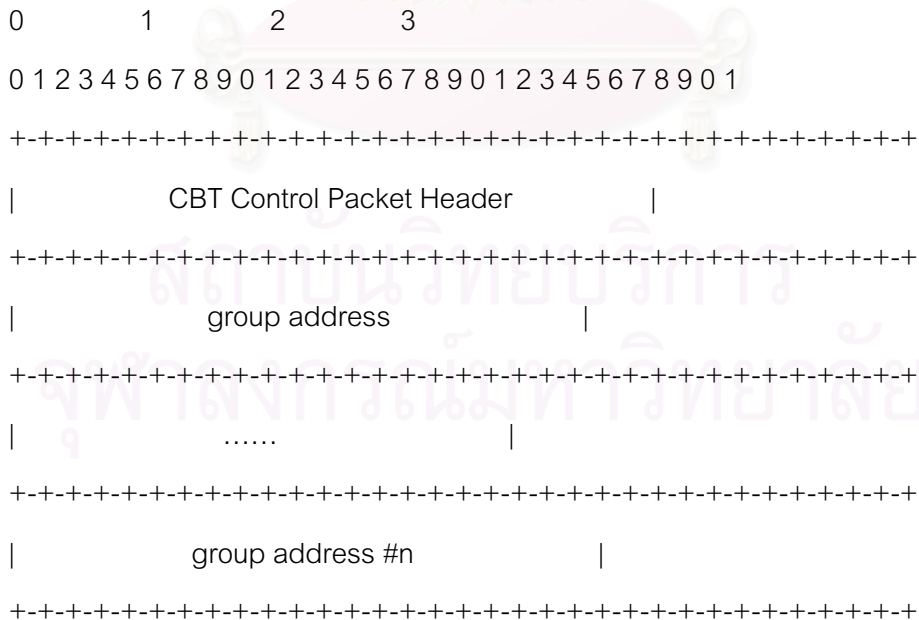


Figure 3.15: FLUSH_TREE Packet Format

FLUSH_TREE Field Definitions

- group address(es): multicast group address(es) of the group(s) being “flushed”.

Bootstrap Message Format

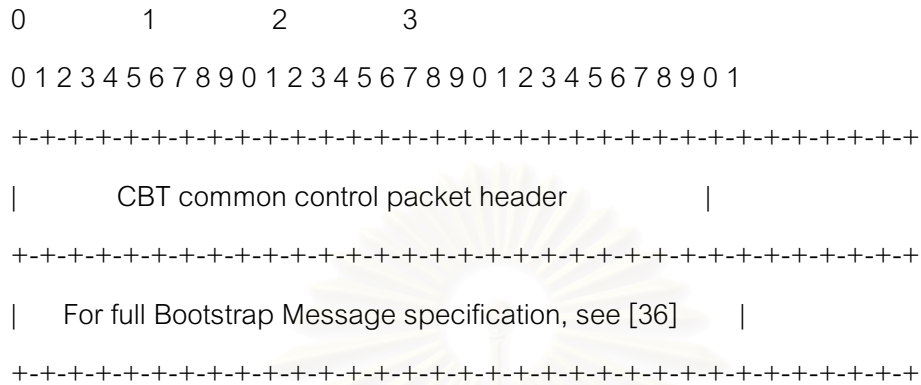


Figure 3.16: Bootstrap Message Format

Candidate Core Advertisement Message Format

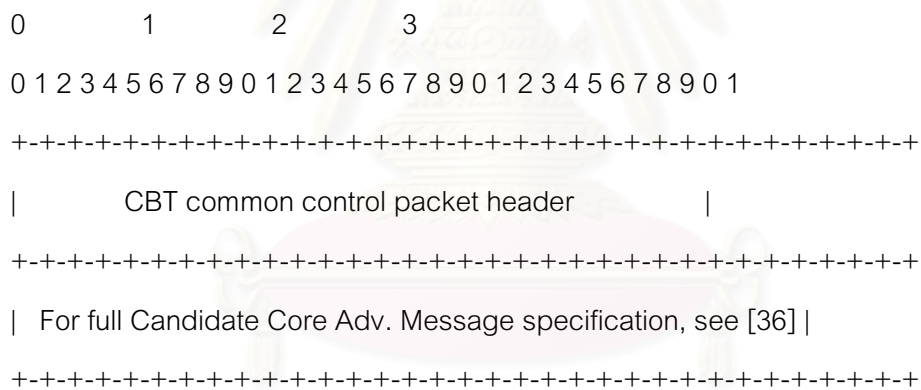


Figure 3.17: Candidate Core Advertisement Message Format

3.5.2 Timers and Default Values

This section provides a summary of the timers described above, together with their recommended default values. Other values may be configured; if so, the values used should be consistent across all CBT routers attached to the same network.

* [HELLO_INTERVAL]: the interval between sending an HELLO message.

Default: 60 seconds.

* [HELLO_PREFERENCE]: Default: 255.

- * [HOLDTIME]: generic response interval. Default: 3 seconds.
- * [MAX_RTX]: default maximum number of retransmissions. Default 3.
- * [RTX_INTERVAL]: message retransmission time. Default: 5 seconds.
- * [JOIN_TIMEOUT]: raise exception due to tree join failure. Default: 3.5 times [RTX_INTERVAL].
- * [TRANSIENT_TIMEOUT]: delete (unconfirmed) transient state. Default: (1.5*RTX_INTERVAL) seconds.
- * [CACHE_DEL_TIMER]: remove child interface from forwarding cache. Default: (1.5*HOLDTIME) seconds.
- * [GROUP_EXPIRE_TIME]: time to send a QUIT_NOTIFICATION to our non-responding parent. Default: (1.5*ECHO_INTERVAL).
- * [ECHO_INTERVAL]: interval between sending ECHO_REQUEST to parent routers. Default: 60 seconds.
- * [EXPECTED_REPLY_TIME]: consider parent unreachable. Default: 70 seconds.

3.6 Core Router Discovery

There are two available options for CBTv2 core discovery; the “bootstrap” mechanism (as currently specified with the PIM sparse mode protocol [35]) is applicable only to intra-domain core discovery, and allows for a “plug & play” type operation with minimal configuration. The disadvantage of the bootstrap mechanism is that it is much more difficult to affect the shape, and thus optimality, of the resulting distribution tree. Also, to be applicable, all CBT routers within a domain must implement the bootstrap mechanism.

The other option is to manually configure leaf routers with `<core, group>` mappings (note: leaf routers only); this imposes a degree of administrative burden - the mapping for a particular group must be coordinated across all leaf routers to ensure consistency. Hence, this method does not scale particularly well. However, it is likely that “better” trees will result from this method, and it is also the only available option for inter-domain core discovery currently available.

3.6.1 "Bootstrap" Mechanism Overview

It is unlikely that the bootstrap mechanism will be appended to a well-known network layer protocol, such as IGMP [37], though this would facilitate its ubiquitous (intra-domain) deployment. Therefore, each multicast routing protocol requiring the bootstrap mechanism must implement it as part of the multicast routing protocol itself. A summary of the operation of the bootstrap mechanism follows (details are provided in [36]). It is assumed that all routers within the domain implement the "bootstrap" protocol, or at least forward bootstrap protocol messages.

A subset of the domain's routers are configured to be CBT candidate core routers. Each candidate core router periodically (default every 60 secs) advertises itself to the domain's Bootstrap Router (BSR), using "Core Advertisement" messages. The BSR is itself elected dynamically from all (or participating) routers in the domain. The domain's elected BSR collects "Core Advertisement" messages from candidate core routers and periodically advertises a candidate core set (CC-set) to each other router in the domain, using traditional hop- by-hop unicast forwarding. The BSR uses "Bootstrap Messages" to advertise the CC-set. Together, "Core Advertisements" and "Bootstrap Messages" comprise the "bootstrap" protocol.

When a router receives an IGMP host membership report from one of its directly attached hosts, the local router uses a hash function on the reported group address, the result of which is used as an index into the CC-set. This is how local routers discover which core to use for a particular group. Note the hash function is specifically tailored such that a small number of consecutive groups always hash to the same core. Furthermore, bootstrap messages can carry a "group mask", potentially limiting a CC-set to a particular range of groups. This can help reduce traffic concentration at the core.

If a BSR detects a particular core as being unreachable (it has not announced its availability within some period), it deletes the relevant core from the CC-set sent in its next bootstrap message. This is how a local router discovers a group's core is unreachable; the router must re-hash for each affected group and join the new core after removing the old state. The removal of the "old" state follows the sending of a QUIT_NOTIFICATION upstream, and a FLUSH_TREE message downstream.

3.6.2 Core Failure

Once the *core tree* has been established as the initial step of group initiation, core router failure thereafter is handled no differently than non-core router failure, with a core attempting to re-connect itself to the corresponding tree by means of either a join or re-join.

When a core router re-starts subsequent to failure, it will have no knowledge of the tree for which it is supposed to be currently a core. The only means by which it can find out, and therefore re-establish itself on the corresponding tree is if some other on-tree router sends it a CBT-CORE-PING message. This message, by default, always contains the identities of all the cores for a group, together with the group-id.

On receipt of a CBT-CORE-PING, a recently re-started core will re-join the tree by means of a JOIN-REQUEST.

It had been considered to just send a JOIN-REQUEST, rather than the apparent overhead of sending a CBT-CORE-PING first. The reason this design option was chosen was because a JOIN-REQUEST instantiates state along the path from the sending router all the way to the core (or an on-tree router on the way to it). If the target core was down, a mechanism would have to be introduced to explicitly remove that state- a disadvantage of not using the “soft state” approach. However, the *unicast* CBT-CORE-PING instantiates no such state.

3.7 Node Failure

The CBT protocol treats core- and non-core failure in the same way, using the same mechanisms to re-establish tree connectivity.

Each child node on a CBT tree monitors the status of its parent/parent link at fixed intervals by means of a “keepalive” mechanism operating between them. The “keepalive” mechanism is implemented by means of two CBT control messages: CBT-ECHO-REQUEST and CBT-ECHO-REPLY.

For any non-core router, if its parent router, or path to the parent, fails, that non-core router is initially responsible for re-attaching itself, and therefore all routers

subordinate to it on the same branch, to the tree (Note: re-joining is not necessary just because unicast calculates a new next-hop to the core).

Subsequent to sending a QUIT-REQUEST on the parent link, a non-core router initially attempts to re-join the tree by sending a RE-JOIN-REQUEST (see under “Loop Detection” in this section) on an alternate path (the alternate path is derived from unicast routing) to an arbitrary alternate core selected from the core list. The corresponding core is tested for reachability before the re-join is sent, by means of the control message: CBT CORE-PING. Failure to receive a response from the selected core will result in another being selected, and the process continues to repeat itself until a reachable core is found.

The significance of sending a RE-JOIN-REQUEST (as opposed to a JOIN-REQUEST) is because of the presence of subordinate routers, i.e. there exists a downstream branch connected to the re-joining router. Care must be taken in this case to avoid loops forming on the tree. If the joining router did not have downstream routers connected to it, it would not be necessary to take precautions to avoid loops since they could not occur (this is explained in more detail below).

It was an engineering design decision not to flush the complete (downstream) branch when some (upstream) router detects a failure. Whilst each router would join via *its* shortest-path to the corresponding core, it would result in an overall longer *re-connectivity latency*.

A FLUSH-TREE control message is however sent if the best next-hop of the re-join is a child on the same tree.

3.8 Loop Detection

The CBT protocol incorporates an explicit *loop-detection* mechanism. Loop detection is only necessary when a router, with at least one child, is attempting to re-connect itself to the corresponding tree.

We distinguish between three types of JOIN-REQUEST: active; active re-join; and non-active re-join.

An *active* JOIN-REQUEST for group A is one which originates from a router which has no children belonging to group A.

An *action re-join* for group A is one which originates from a router that has children belonging to group A.

A *non-active re-join* is one that originally started out as an active re-join, but has reached an on-tree router for the corresponding group. At this point, the router changes the join status to *non-active re-join* and forwards it on its parent branch, as does each CBT router that receives it. Should the router that originated the active re-join subsequently receive the non-active re-join, a loop is obviously present in the tree. The router must therefore immediately send a QUIT-REQUEST to its parent router, and attempt to re-join again. In this way the re-join acts as a *loop-detection* packet.

Another scenario that requires consideration is when there is a break in the path (tunnel) between a child and its parent. Although the parent is active, the child believes that the parent is down—the child cannot distinguish between the parent being down and the path to it being down. If the path failure is short-lived, whilst the child will have chosen a new route to the core, the parent will be unaware of this, and will continue forwarding over its child interfaces, the potential risk being apparent.

We guard against this using a *parent assert* mechanism, which is implicit, i.e. involves no control message overhead, in the reception of CBT-ECHO-REQUESTs from a child. If no CBT-ECHO-REQUEST is heard, after a certain interval the corresponding child interface is removed by the parent.

As an additional precaution against packet looping, multicast data packets that are in the process of spanning a CBT's delivery tree branches (remember, we distinguish between actual tree branches and attached subnetworks, although there are cases when they are one and the same) carry an *on-tree* indicator in the CBT header of the packet. Provided a data packet arrives via a valid tree interface, all routers are obliged to check that the on-tree indicator is set accordingly. A data packet arriving at the tree for the first time from a non-member sender will have the *on-tree* indicator bits set by the receiving router. These bits should never subsequently be modified by any router. Should a packet be erroneously forwarded by an on-tree router over an off-tree interface, should that packet somehow work its way back on tree, it can be immediately recognized and discarded, since it will have arrived via a non-tree interface, but will have its on-tree bits set.

3.8.1 Unicast Transient Loops

Routers rely on underlying unicast routing to carry JOIN-REQUESTs towards the core of a core-based tree. However, subsequent to a topology change, transient routing loops, so called because of their short-lived nature, can form in routing tables whilst the routing algorithm is in the process of converging or stabilizing.

These are two cases to consider with respect to CBT and unicast transient loops, namely:

- a join is sent over a transient loop, but no part of the corresponding CBT tree forms part of that loop. In this case, the join will never get acknowledged and will therefore timeout. Subsequent re-tries will succeed after the transient loop has disappeared.
- a join is sent over a transient loop, and the loop consists either partly or entirely of routers on the corresponding CBT tree. If the loop consists only partly of routers on the tree and the join originated at a router that is not attempting to re-join the tree, then the JOIN-REQUEST will be acknowledged. No further action is necessary since a loop-free path exists from the originating router to the tree. If the loop consists entirely of routers on the tree, then the router originating the join is attempting to re-join the tree. In this case also, the join could be acknowledged which would result in a loop forming on the tree, so we have designed a loop-detection mechanism which is described below.

3.9 Conclusions

In this chapter, we introduced the CBT multicast protocol, and described in detail the specifics of its operation, such as CBT tree-building, tree maintenance, tree teardown, LAN designated router election, and data packet forwarding and reception. We discussed the protocol functionalities in detail, to understand the states of the CBT protocol and its protocol data units used in the CBT routing processes. Packet formats and message types were also presented. We described the various types of control packets and their functions.

The fundamental motivation behind the design of the CBT architecture was the ability to significantly improve multicast scalability. This improvement does not come without its cost, and in the case of CBT the primary disadvantage of the architecture is the potential for increased delay between two multicast receivers – a consequence of the absence of source-based shortest path trees. The trade-offs introduced by the CBT architecture focus primarily between a reduction in the overall state the network must maintain and the potential increased delay imposed by a shared delivery tree. However, because of CBT's "hard-state" approach to tree building, i.e. group tree link information does not time out after a period of inactivity, as opposed to the "soft-state" approach (for example: DVMRP), CBT branches do not automatically adapt to underlying multicast route changes. This is in contrast to the "soft-state" or "data-driven" approach-while data always follows the path as specified in the routing table. In addition, "hard-state" requires the incorporation of control messages that monitor reachability between adjacent routers on the multicast tree. This control message overhead can be quite considerable unless some form of message aggregation is employed.

The maximum delay bound of a CBT has been proven to be twice that of a shortest-path tree, which may not be acceptable for real-time applications, such as voice conferencing. However, with some modifications to the original CBT protocol, it is possible to satisfy the end-to-end time delay imposed by the real-time applications. Another disadvantages of CBT is that the cores for a particular group, especially large, widespread groups, can potentially become traffic "hot-spots" or "bottlenecks".

In summary, the CBT architecture has several advantages compared with existing schemes, which include:

- CBT architecture eliminates the source scaling factor of the source-based architecture, in terms of group-specific state maintained in the network.
- No state is maintained by the network between a non-member sender and the delivery tree.
- CBT's two-phase routing approach means that it is a candidate for information discovery applications. CBT can offer an efficient solution to "anycasting".

We also looked at the implications of shared multicast delivery trees, and conclude that they also offer several disadvantages, including:

- Shared delivery trees do not optimize delay, which is critical to some multicast applications.
- CBTs can result in so-called “traffic concentration” or “hot spots” around core routers.
- CBTs require the selection of a small number of core routers by a group’s initiator. Core selection and placement, i.e. core management, are not required in existing IP multicast schemes.

Despite its disadvantages from our study, we still think, with its relatively simple protocol structures, its scalability and low overhead, CBT has the potential for real-time multicast applications in the Internet, provided that an improvement to the original CBT be incorporated without sacrificing its advantages. Hence, this motivates us to conduct the research towards the modification of the original CBT to adapt to the real-time multicast traffic with low cost. In the next chapter, we will formulate the real-time optimal multicast routing problems and propose a solution based on the CBT protocol. We then propose the strategy to optimize the path from source to the shared tree. And we later propose two path selection algorithms thus optimizing the overall CBT trees to balance the tree cost factor and subject to the end-to-end delay constraints.

CHAPTER 4

REAL-TIME OPTIMAL MULTICAST ROUTING

Overview

The Internet has begun using multicast delivery to support group communication. Multicast delivers packets from a sender to a group of receivers over a multicast tree. The primary advantage that multicast has over traditional unicast delivery is that the sender transmits a single packet to reach all of the group members, rather than sending a separate copy to each receiver. Replication of each packet is handled by the network and is done only when necessary, i.e. at the branching points in the multicast tree. Another recent development has been the increasing use of real-time applications in the Internet. Real-time applications impose stringent delay and throughput constraints on the network, as compared with traditional elastic applications. When real-time applications communicate across a network, data must traverse the network in time for the application to use it. Likewise, a real-time application often needs a certain amount of throughput, below which it does not receive adequate service. Because of these characteristics, real-time applications require new mechanisms, beyond TCP and best-effort service, to cope with delay and loss.

One approach to improving the performance of real-time applications over the Internet is to design new adaptive techniques, similar to the use of TCP for elastic applications. These techniques include adapting a receiver's audio playback point and varying a receiver's subscription to levels of hierarchically encoded video. Another approach is to upgrade the best-effort service model of the Internet to include enhanced levels of service characterized by reduced delay or increased bandwidth. Many researchers in this area have proposed an integrated services architecture that uses a combination of scheduling algorithms, admission control, and a reservation protocol to control access to these services levels. Huge efforts are currently under way, at both the algorithm and protocol levels, to develop multicast routing mechanisms which:

- Satisfy the QoS requirements of the rapidly evolving real-time applications,
- Are capable of managing the network resources efficiently, and,
- Scale well to large network sizes.

Researchers have proposed the core-based trees (CBT) and protocol independent multicasting (PIM) protocols to route multicast data on internetworks. Algorithms utilizing CBT construct a single tree for each group, regardless of the number of senders. However, the end-to-end delivery delay is larger for CBT than for a source-based tree (SBT). In this chapter, we present a real-time optimal Multicast routing based on modified CBT v2 protocol. The proposed protocol can sustain the scaling advantage of CBT without letting the delay from source to any destination exceeds a real-time constraint.

We adopted a strategy called “the shortest of the shortest path” to optimize the path from source to the shared multicast tree, replacing the original approach of CBT, which uses a shortest path from source to a core router. In addition to selecting a path in a multicast tree which does not violate the end-to-end deadline, we present two new path selection methods that take both cost and delay into consideration. By changing the path selection function, the overall cost of the multicast tree can be reduced significantly while satisfying the real-time delay constraint of the applications. Our simulation results show significant improvement for our proposed protocol over the CBT v2. On average our proposed protocol allows us to achieve the balance of optimizing cost and delay of the shared multicast group tree.

4.1 Introduction

Multicasting is a communication service that allows an application to efficiently transmit copies of a data packet to a set of receivers that are members of a multicast group. The group is identified by a location independent multicast group address. Senders use this address in the destination field of the packet; multicast routers forward the packet to group members using routing table entries for this address. The entries form a tree, which may be a source-based tree or a center-based tree depending on the multicast routing protocol. Multicast group members may be spread across separate physical

networks, they may join and leave a group during the life of the group, and they may be members of multiple groups. Multicast routing has been performed by a multicast-capable, virtual network running on top of the internet called the multicast backbone “Mbone”. The Mbone uses the distance vector multicast routing (DVMRP) or the multicast extensions for open shortest path first protocol (MOSPF) to route multicast traffic. Common uses of multicasting include audio and video conferencing, distributed interactive simulation (DIS) activities such as tank battle simulations, and exchanging experimental data and weather maps. DVMRP and MOSPF depend on features of underlying point-to-point (unicast) routing protocols. Efforts to remove this dependency and to develop point-to-multipoint (multicast) routing protocols that operate in a hierarchical manner which subnet multicast routing protocols led to the development of the core-based tree (CBT) protocol [5, 6] and the sparse mode of the protocol independent multicasting (PIM) protocol [28].

In multicast communication, there is a source node s and a set of destination nodes D . The multicast routing is to find a routing tree which is rooted from s and contains all nodes in D . Multicast routing has two important requirements: minimal network cost and shortest network delay. The network cost is the overall network cost of transmitting a message to all destinations; and the network delay is measured by the longest delay from the source to any destinations. In real-time applications, there often exists a real-time constraint and it is required that the communication be done within the constraint. There is no need to achieve the shortest delay to each of the destinations.

We propose a real-time multicast routing algorithm which reduces overall network cost without letting the delay from a source to any destination exceed a real-time constraint. Simulation results have shown that our algorithm has a significant improvement over the original CBT v2.

The rest of the chapter is organized as follows. Section 4.2 presents the brief previous research in real-time multicast network. Section 4.3 describes multicast routing background and our two proposed path selection functions are discussed in section 4.4 and section 4.5. Section 4.6 describes the modification to the original CBT v2, with the flow chart demonstrating the modification detail. Section 4.7 shows the simulation setup

demonstrating the capability of our proposed protocol, followed by the discussion of the simulation results in section 4.8. The chapter concludes with Section 4.9.

4.2 Previous Work

Detail survey of the multicast routing problems can be found in chapter 2. We briefly discussed some of the multicast routing algorithms which are relevant to the context of real-time multicast routing problems. Optimal algorithms for constructing delay-constrained minimum Steiner trees exist, but their execution times are prohibitively large, because the problem is *NP*-complete [56]. Several delay-constrained Steiner tree heuristics have been proposed during the past few years. The heuristics proposed in [111] use a delay-constrained Bellman-Ford shortest path algorithm during the computation of the delay-constrained Steiner tree. [30] presented a delay-constrained heuristic based on Dijkstra shortest path algorithm. A heuristic that constructs a Multicast tree subject to both an upper bound on end-to-end delay and on delay variation is given in [86]. In KMB algorithm [69], a network is abstracted to a complete distance graph consisting of edges that represent the shortest paths between the source node and each destination node. The KMB algorithm constructs a minimum spanning tree in the complete distance graph, and the Steiner tree of the original network is obtained by achieving the shortest paths represented by edges in the minimum spanning tree. Kompella, Pasquale, and Polyzos [65] proposed two heuristics that address delay-bounded multicast trees, called KPP. In their formulation, the delay bound for all destinations is the same; furthermore, the algorithm assumes that link delays and the delay bound are integer valued and that link costs and delays are symmetric. KPP extends the KMB algorithm by taking into account the constraint of the specified delay bound in the construction of the complete distance graph. [81] proposed the bounded shortest multicast algorithm (BSM) algorithm for constructing minimum-cost multicast trees with delay constraints. BSMA handled asymmetric link characteristics and variable delay bounds on destinations, specified as real values, and minimizes the total cost of a multicast routing tree. But BSMA is not suitable for large-scale networks whereby hierarchical routing should be applied to cope with network size such as OSPF (Open shortest path first) routing protocol.

[93] proposed a hybrid multicast routing algorithm of the CBT and PIM-SM. The proposed algorithm builds a bidirectional shared tree centered around a rendezvous point as in the CBT and also builds a shortest path tree rooted at a sender as in the PIM-SM if the message transmission rate from that sender is above a threshold. The message transmission delay is lower than both the CBT and the PIM-SM. However, the overheads due to control messages to build and maintain these trees are slightly higher than the PIM-SM and much higher than the CBT. Similar idea was proposed by [64] in coping with the disadvantages of CBT whereby a non-cored based tree (NCBT) is used. With this scheme no core nodes are assigned, but a multicast node among on-tree nodes is assigned to each new incoming member. This multicast node is selected such that the length of the path from the incoming user to a multicast node and the maximum end-to-end delay on the tree are jointly minimized. [101] proposed QoS guaranteed on CBT with member join/leave admission tests so as to provide adequate QoS. However, it is interesting to find an appropriate threshold level at which the advantage of completely reconstruction the multicast tree (e.g. higher probability of approving more join requests) offset the corresponding overhead. Recent works on real-time multicast routing were described in [51, 100, 104].

We believe that our work is different from previous real-time multicast algorithms which focus on the Steiner trees, that is suitable for a single source and static multicast group environment, while ours bases on more practical shared trees and more scalable multicast routing protocol such as CBT.

4.3 Multicast Routing Protocols

4.3.1 Source-Based Trees and Shared Trees

Data packets addressed to a multicast group may be routed on a tree that is specific to the particular sender and group or a tree that is shared by all of the senders to the group. The first approach uses a source-based tree (SBT) that is a shortest-path tree rooted at a sender. The branches of the tree are the shortest paths from the sender to each of the group members. A separate tree must be constructed for each sender to each active multicast group. A protocol that implements SBT is the dense mode of PIM

(PIM Dense Mode) [28]. The shared tree approach uses a single centered-based tree or core-based tree to route traffic from all senders to the group. The tree is a shortest-path tree rooted at one or more predefined nodes in the network called “Core” nodes. A protocol based on center-based trees is CBT [5, 6]. CBT sets up and maintains a single shared tree for every multicast group that is active in the network. When a multicast router is notified via IGMP that a local host would like to join group, the router sends a join message for that group toward the Core node via the shortest path. A tree rooted at the Core is constructed as the acknowledgments to the join messages are processed. The resulting tree is a bidirectional acyclic graph that reaches every group member. Forwarding packets to the group members using CBT is straightforward. When a node on the tree receives a packet addressed to the group, it forwards copies of the packet on all branches of the group’s tree except for the branch on which the packet arrived. There is currently some debate over which type of tree provides the best performance. Algorithms that use CBT construct a single tree for each group, regardless of the number of senders. Because the packets are not guaranteed to travel the shortest path, one expects the end-to-end delay to be larger for CBT algorithms than for SBT ones. Another drawback is the traffic concentration near the core router. However, SBT algorithms scale poorly for large numbers of senders because the router resources required to maintain knowledge of the tree structure are considerable.

Real-time multicast refers to a multicast in which a message will be received by all destinations within a specified time delay. In real-time communication, a channel must be established before any user’s message can be transmitted. During the channel establishment, the system selects a route along which sufficient resources can be reserved to meet the user specified requirements, such as network bandwidth and maximum message delay. This is an important step to guarantee a real-time message to be delivered to destinations on time. These real-time routing algorithms aim to find a route whose delay is within a user-specific constraint. The network cost of the route is not a primary concern.

4.3.2 Real-time multicast Network Model

The real-time multicast routing is defined as follows:

Given a network graph $G(V,E)$, a source node $s \in V$, a set of destination nodes $D \subset V$, and a real-time constraint Δ , a real-time routing tree for multicast connections is a subtree of the graph $G(V,E)$ rooted from s , that contains all of the nodes of D and an arbitrary subset of nodes of D , and the delay from s to any node in D is within the time constraint Δ . The best known of route selection protocols subject to the cost constraint which can be minimum delay bound or minimum number of links are: Dijkstra algorithm and Bellman-Ford algorithm. The former algorithm minimizes the cost of the route, while the latter minimizes the number of links in the path, subject to cost constraint. As mentioned earlier that, if we can set the cost of a link to the minimum delay bound which can be offered on that link, the Bellman-Ford algorithm can satisfy the requirement of real-time traffic, assuming that the resource state information are available in the routing tables.

In CBT routing, one node of the multicast group is selected as the “core” and the routing tree is the shortest path trees rooted from the core to all the other group member. With our modification to the CBT, we use the shortest to the shortest path trees [46], if we can find them under the condition that the end-to-end delay from a source to all nodes in the multicast group does not violate the delay bound Δ , then the condition of real-time CBT can be fulfilled. Detail of the CBT modification is mentioned in the subsequent section. In the next section, we propose two path selection functions which will take into account both the cost and the delay factors. This path selection can be applied to various multicast tree building algorithms.

4.4 Weighted Dijkstra's Shortest Path Tree Algorithm

For real-time multicasting, the network delay of the path to each destination in a multicast routing tree should not exceed a delay constraint imposed by the time-sensitive applications. Unfortunately, the objectives of optimizing the network cost and optimizing the network delay are often conflicting in nature as the two parameters are assumed to be independent from each other. For example, a multicast tree based on the minimum-cost Steiner Tree may contain paths with unacceptable path delay, while a multicast tree based on the Shortest Path Tree may yield high overall network cost. A compromise is needed between the two objectives.

Since building a real-time multicast tree requires optimizing of the network cost while meeting the delay constraint, both the cost and delay factors should be considered at the time of building the multicast tree. Therefore, a path selection function which is based on both cost and delay is proposed [76]. The path selection function is defined by means of the “Weighted path selection criterion”. Basically, it selects an optimal path by weighting the path cost against its path delay. The optimal path is defined as :

$$(C_w * Path Cost) + (D_w * Path Delay). \quad (4.1)$$

The path cost and the path delay are assumed to be two independent network parameters, although, in reality, they may be closely related and are inversely proportional to each other. The Cost Weight, C_w , and Delay Weight, D_w , simply reflect the relative importance of the network cost against the network delay. They allow us to specify the desired balance between optimizing the network cost and optimizing the network delay. In some case, either one of them can be set to zero to ignore either the network cost or the network delay.

In this dissertation, we intend to use the path selection function in the Dijkstra’s Shortest Path Tree Algorithm whereby cost function represents “hop count” or “distance” between router nodes. As a result the Weighted Dijkstra’s Shortest Path Tree can be generated. Note that the word “shortest” here refers to the smallest combined path cost and path delay. We modified the Dijkstra’s Shortest Path Tree algorithm by substituting the original path selection with weighted path selection function in order to create optimal solution balancing cost and delay parameters. By properly assigning the value of C_w and D_w , the overall cost of the multicast tree can be reduced significantly while satisfying the end-to-end delay constraint. As the weighted path selection function does not guarantee that a path can meet the delay constraint, in the situation when the path generated between a new node and the source cannot meet the delay constraint, the Weighted Shortest Path algorithm will use the shortest path based on path delay only. Although this path may have significantly higher cost, it can be easily computed and it is the absolute shortest path from the new node to the source node with the best chance of

meeting the timing requirements. Since the Weighted Shortest Path Tree algorithm is basically a modified Dijkstra's algorithm, the complexity of this algorithm is also $O(|V|^2)$.

Weighted Dijkstra's Shortest Path Tree algorithm:

Definitions:

G = set of nodes in a network

S = set of nodes in the Weighted Shortest Path Tree

s = source node

$Cost_{x,y}$ = Accumulated path cost from node x to node y , or ∞ if no path exist yet

$Delay_{x,y}$ = Accumulated path delay from node x to node y or ∞ if no path exist yet

$$CD_{x,y} = (Cw * Cost_{x,y}) + (Dw * Delay_{x,y}) \quad (4.2)$$

1. $S \leftarrow \{s\}$
2. $Cost[n] = Cost_{n,s} \quad \forall n \neq s$
3. $Delay[n] = Delay_{n,s} \quad \forall n \neq s$
/* Find next node not in S and nearest to source node */
4. while $S \neq G$
5. Find $w \notin S$ such that $CD_{w,s} = \min CD_{j,s} \quad \forall j \notin S$
6. $S \leftarrow S \cup \{w\}$
7. For all $n \notin S$, /*Update least cost paths */
8. If $(CD_{n,w} + CD_{w,s} < CD_{n,s})$
9. $Cost[n] = Cost_{n,w} + Cost[w]$,
 $Delay[n] = Delay_{n,w} + Delay[w]$
10. For each destination node, if the path delay $>$ delay constraint Δ ,
replace the path by the shortest path based on delay only
11. Remove all branches leading to non-destination nodes

4.5 Residual delay path selection function

In previous section, we presented a new path selection function to find an optimal path connecting the source via a new on-tree node which does not violate the real-time delay

constraint values, with reduced cost tree. However, the cost of the multicast tree is not optimal. To further decrease tree cost, we propose another path selection function to optimize cost and delay based on Kompella's algorithm [65], called "residual delay path selection function". The new selection function explicitly uses both cost and delay in its functional form. It tries to choose low-cost edges, but modulates the choice by trying to pick edges that maximize the residual delay. It is given by

$$\text{Path selection function } (v, w) = \frac{C(v, w)}{\Delta - P(v) - D(v, w)} \quad \text{if } P(v) + D(v, w) < \Delta \quad (4.3)$$

Where $C(v, w)$ = cost of the link (v, w)

$D(v, w)$ = delay on the link (v, w)

$P(v)$ = delay on the path from s to v in the tree

This new path selection function has two components, $C(v,w)$, which if minimized would lead to a low cost solution, and the denominator, which we call the residual delay, the remainder of the delay left over after traversing the path from s to w that can be used to reach other nodes without violating the bounds Δ . If the function is minimized, then the residual delay would tend to increase. This yields a larger probability of being able to select an edge out of w to another destination. Thus, while maintaining the delay bound, we are presented with the opportunity of path sharing, i.e., the overlapping of paths from source to more than one destination. This, in turn, leads to lower tree costs, in contrast to divergent paths that usually tend to raise tree costs. However, this new selection function has a tendency to optimize on delay also, in that it may find paths with delays lower than Δ , at the expense of added cost to the tree. In conclusion, the selection function can be used to simultaneously minimize the cost of the path and maximize the residual delay. We modified the Dijkstra's Shortest Path Tree algorithm by substituting the original path selection with the residual delay path selection function in order to create optimal solution balancing cost and delay parameters. In the next section, we will describe the modification of the CBT v2.

4.6 CBT modifications

In our formulation, the original CBT version 2 was modified to optimize the transmission path from a source which is off-tree to the on-tree node of the shared multicast tree. To improve network performance, we propose a new method: for an off-tree router, we first find the shortest paths from the source router to all the nodes on the multicast tree. Then, we select the path that is the shortest among these shortest paths and use it to route a multicast packet from this source router to the shared tree.

This on-tree node will be used as a new access point from the source to the shared tree. This method is called “shortest of the shortest path” [46].

After the shortest of the shortest path is obtained, then we apply our Weighted Dijkstra’s Shortest Path Tree algorithm to find the paths to all nodes in the multicast group so that the total path delay from the source node via an on-tree node to each multicast node in the group does not exceed the maximum end-to-end delay Δ . By simply replacing the path selection function in the multicast tree building algorithms with the proposed weighted path selection function, the overall cost can be reduced significantly while the real-time constraint can still be maintained. We then use our residual delay function to replace the path selection of the Dijkstra’s algorithm and compare the results from our experiments between the two path selection functions. The flow chart of our proposed approaches is shown in figure 4.1.

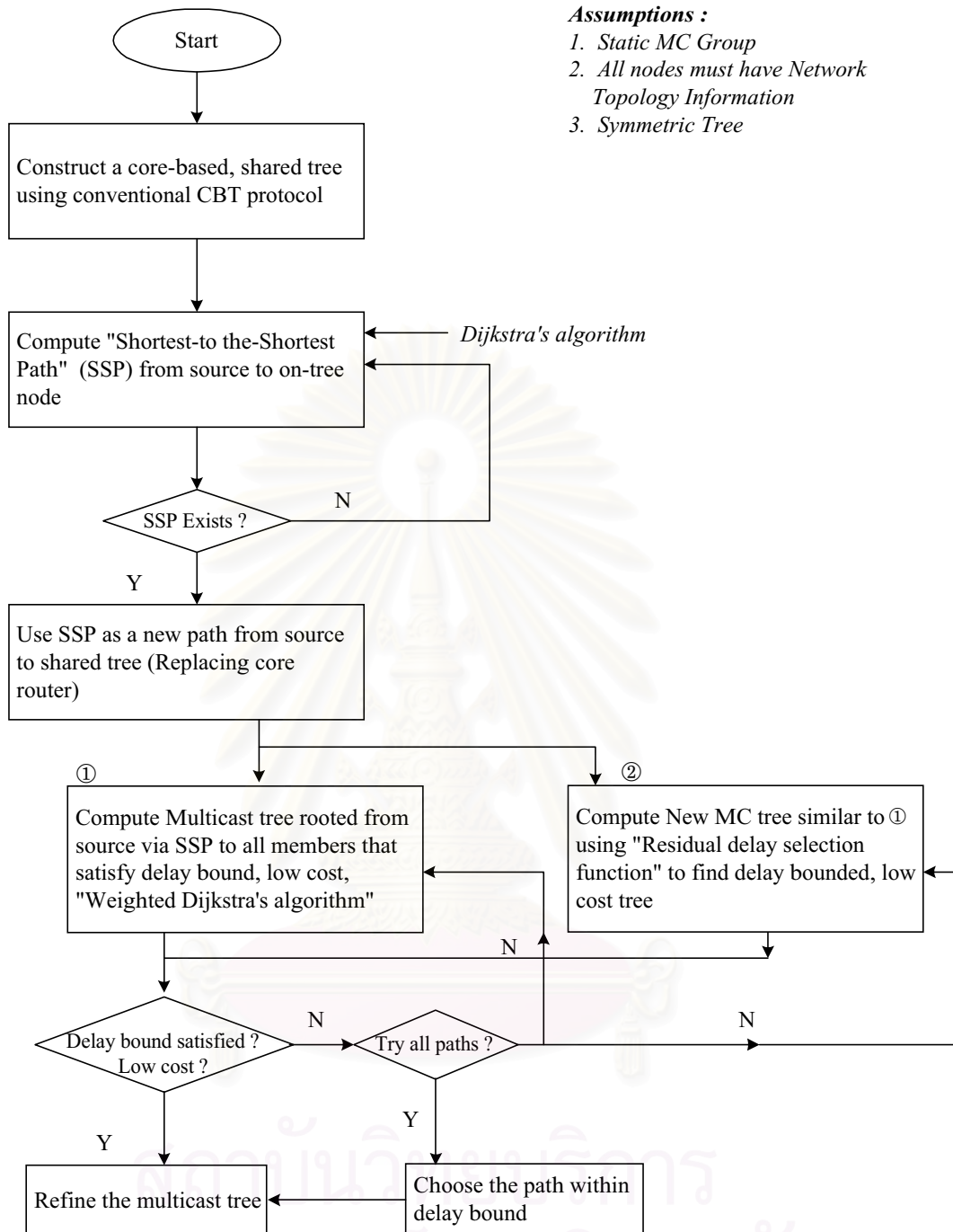


Figure 4.1: Flow chart of our proposed protocol with two path selection functions

The proposed method takes more off-line time to collect locations of nodes and compute the shortest paths. Once the route is determined, the runtime overhead is slightly higher than that in the CBT v2 [64].

In implementing our protocol, however, the network requires an additional capability. All nodes in the network should maintain the current tree information, since one of the on-tree nodes is selected as an access point to the shared tree in lieu of the core node in CBT. In the CBT approach, all nodes in the network must keep the information of the core node location. To do this, the bootstrap router (BSR), broadcasts the information of candidate core nodes onto all CBT routers. Similarly, in our formulation, the information of on-tree node is broadcasted to all routers. The additional overhead is the address of the designated on-tree node for each source and each multicast group. Note that the overhead required for the tree building and maintenance is nearly the same for both approaches.

To prove that our proposed protocol is loop-free, we assume that the existing tree does not contain a loop, and prove informally that a newly added branch does not create a loop [5]. In our proposed protocol, we assume that every on-tree node has the same information that the core node has. So all on-tree routers simulate the role of the core, i.e. the whole multicast tree shows the same input and output as a huge core, which has edges connected to all routers adjacent to the tree. Thus, the establishment of a new branch in our proposed protocol simulates building the first branch in a CBT. Thus, if CBT is loop-free, then ours is also loop free.

4.7 Simulation

In this section, we will describe the simulation results and discuss the performance of our protocol. We adopted the Waxman approach in [107] to construct our network graph. In our experiments, we used an average node degree of 4, which is close to the average node degree of the current Internet. We distributed n nodes randomly across a Cartesian coordinate grid of size 100 by 100. Nodes in the network graph represent the communication endpoints. The edges connecting the nodes represent links. Edges between nodes are added by considering the probability function

$$P(u,v) = \beta \exp(-d(u,v)/L\alpha) \quad (4.4)$$

for all possible pair (u,v) of node, where $d(u,v)$ is the Euclidean distance between the nodes u and v , L is the maximum possible distance between any two nodes, and β , α are parameters in the range $(0,1)$. Large values of β increases the number of edges from each node while smaller values of α increases the number of connections to nearer nodes compared to nodes further away. In our simulation, we set β to 0.25 and α to 0.2 to simulate a large network such as the Internet. To simulate the propagation delay, we assigned to each edge a network delay equivalent to the Euclidean distance between the two nodes. An example of a randomly generated network of 20 nodes and an average degree of 4 is shown in figure 4.2. In terms of network cost, we experimented with random cost value in the range 0 to 99 to each edge, and with unit cost equates to hop count for each edge. We found that using unit cost produces similar results to that of the random cost.

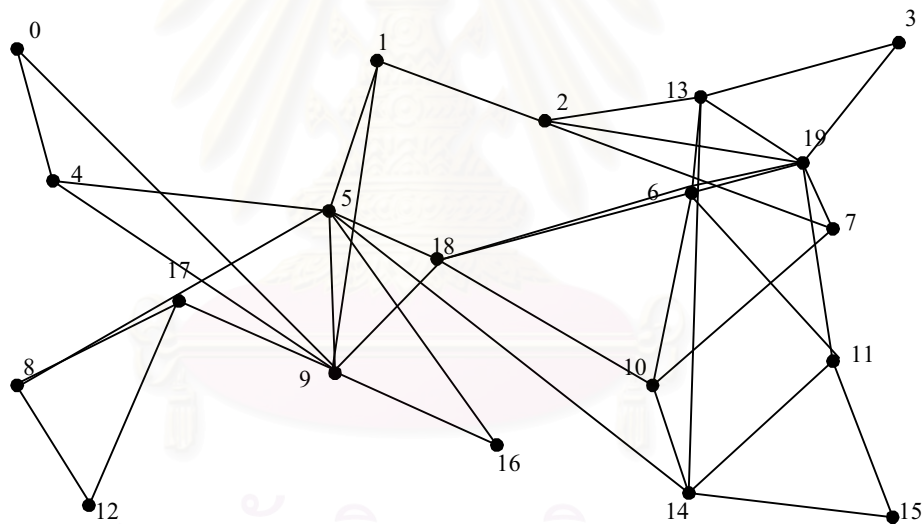


Figure 4.2: A randomly generated network, 20 nodes, average degree 4.

CBT version 2's specifications described in RFC 2189 is also simulated on ns-2 [75] as a baseline. The simulation environment is depicted in Table 4.1. At each simulation the multicast group size and the real-time constraint values Δ vary. Each time the source node and the destination set were randomly selected from the network graph. We used a static multicast model in our experiment. We assumed in our experiment that the network is symmetric, i.e., link cost and link delay in both directions are the same. This

can be represented as an undirected network in which $C(e) = C(\bar{e})$ and $D(e) = D(\bar{e})$ for all $e \in E$. The experiment was run repeatedly until the confidence interval for the number of all measured quantities are less than 5% using the 95% confidence level. On the average, 300 different networks were simulated in each experiment in order to reach such confidence levels. The label “Modified CBT_{WD}” denotes our proposed modified CBT protocol using Weighted Dijkstra’s path selection method, while label “Modified CBT_{RD}” corresponds to our proposed modified CBT protocol using the residual delay path selection function.

Simulation components	Description
CPU	Pentium III 733 Mhz, 256 MB RAM, 30 GB Hard Disk
Operating System	Linux Red Hat 6.2
ns2	ns-2.1b7

Table 4.1: Simulation environment.

The following performance metrics are considered:

- End-to-end delay bound: Time elapsed between the generation of a packet at a source and the reception of that packet by a group member. The average propagation delay is compared between our proposed protocol to its counterpart in CBT v2. Also, the results should prove that our protocol can route the packet within a time delay bound of the real-time application (maximum end-to-end delay) .
- Network resource usage: Total number of hops a multicast packet travels to reach all destination in the multicast groups. It is computed by dividing the total number of hops measured in a simulation by the number of packets received.
- Traffic concentration: Traffic concentration is measured by the maximum number of flows traversing a unidirectional link (the load of the most congested link). The distribution of traffic is observed and compared with that of the CBT v2, also the

change in the maximum link load of each multicast tree. This also shows link utilization of our proposed protocol against that of the CBT v2.

- **Loss Rate:** The loss rate measures the fraction of the transmitted packets that are not delivered at all or are delivered so late as to be useless for real-time applications. This also shows the effectiveness of our protocol in constructing the acceptable multicast tree for a given delay bound.
- **Execution time:** The execution time measures the running time of our algorithm from start until the time the multicast tree is completely formed.

4.8 Performance evaluation

The simulated traffic is with burst size 1 MB, and average rate of 1 Mb/s. The maximum length of the packet was set to 1000 bytes. The link capacity of all the links in the network was chosen randomly from the set of {2, 4 and 8} Mb/s. We repeated our experiment with different link capacity. The results have similar trends. We would like to mention here that the choice of our traffic is not to be emphasized in this study since our goal is to compare the performance produced by our proposed protocol under identical network conditions.

4.8.1 Average end-to-end delay

Average end-to-end delay is the average period for a data packet to be routed through the network from the application where it was created to a destination application. Different values of C_w and D_w in our Weighted Dijkstra's Shortest Path Tree algorithm were experimented in order to obtain the best total network cost performance. Then, we repeated the experiment using the residual delay path selection function to create the multicast tree. Figure 4.3 and figure 4.4 show the results of our proposed protocols to their counterparts in CBT v2 in terms of total network cost against maximum network end-to-end delay for multicast group sizes of 50 and 100. They reveal that our proposed protocols yield better cost and delay performance than those of the original CBT for both multicast group sizes. It is evident from figure 4.3 that CBT yields the worst network cost tree, which is up to three times the network cost of our modified protocols. At all delay constraint values, modified CBT_{RD} can consistently sustain a lower cost tree than that of

the modified CBT_{WD} . It can be seen from figure 4.3 that modified CBT_{RD} can produce up to 40 % less cost tree than that of the modified CBT_{WD} .

We are interested in comparing the overall network cost against real-time constraint Δ . For a smaller Δ , no routing tree which satisfies the real-time constraint exists. Δ is incremented, until the network cost almost levels off. Figure 4.5 and figure 4.6 show the histograms of the ratio of the average end-to-end delay in CBT, modified CBT_{WD} and modified CBT_{RD} respectively. It reveals that the both modified CBT_{WD} and modified CBT_{RD} have better delay performance than that of the CBT. On average, modified CBT_{RD} performs best in terms of utilizing a shorter delay paths, while maintaining lower cost tree.

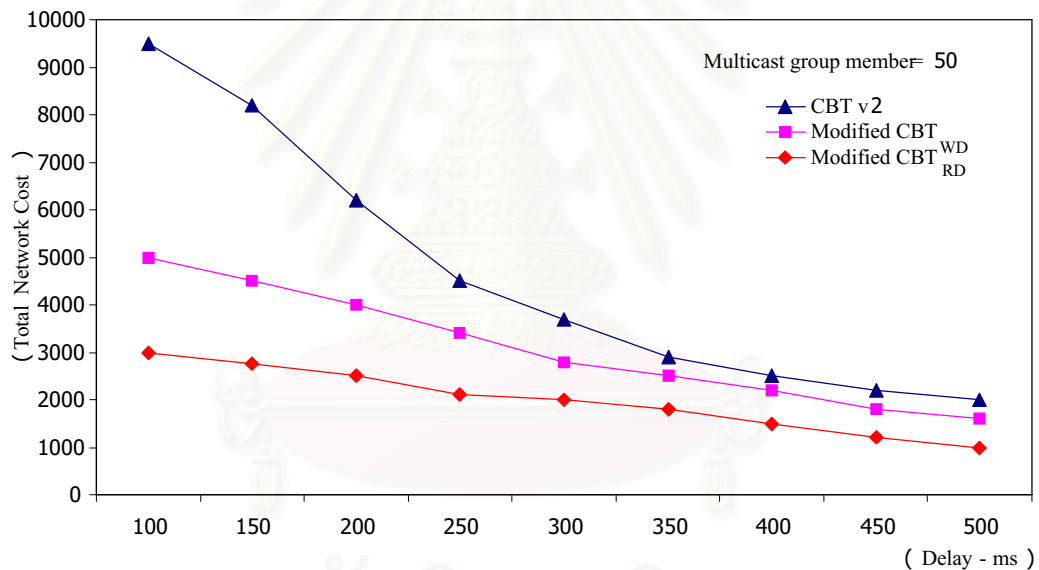


Figure 4.3: Total network cost against end-to-end delay constraint, multicast group member = 50.

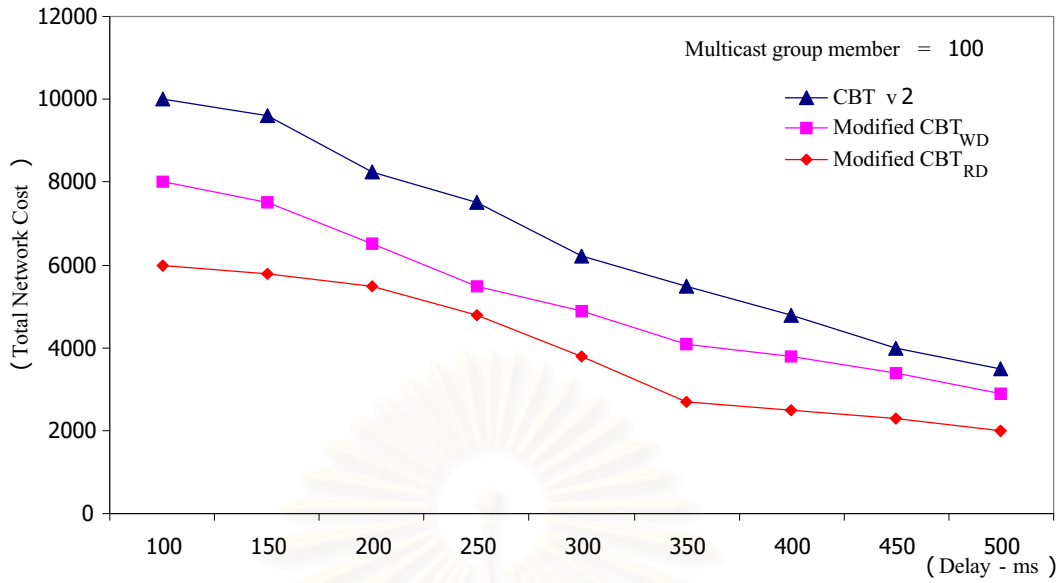


Figure 4.4: Total network cost against end-to-end delay constraint, multicast group member = 100.

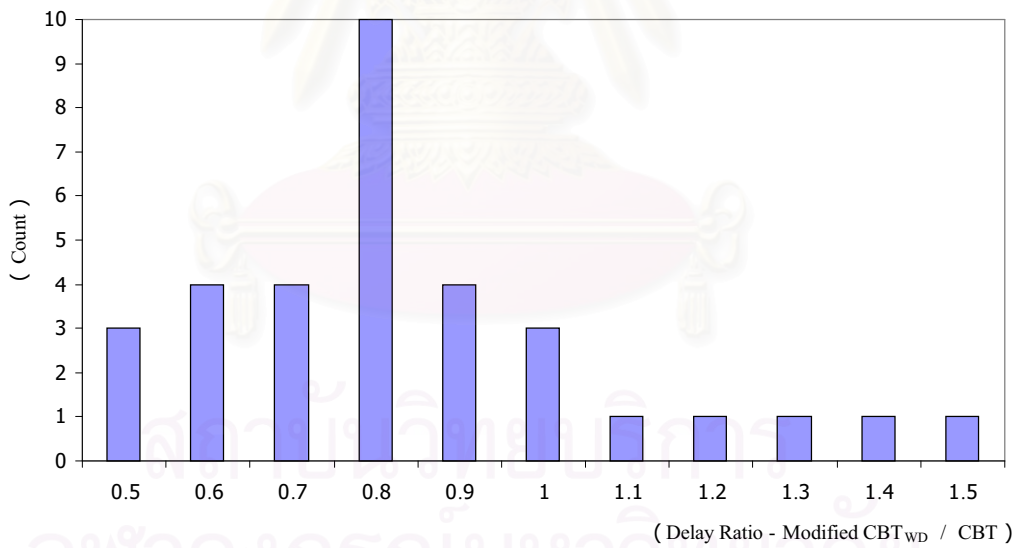


Figure 4.5: A Histogram of average delay in modified CBT_{WD} compared to its counterpart in CBT.

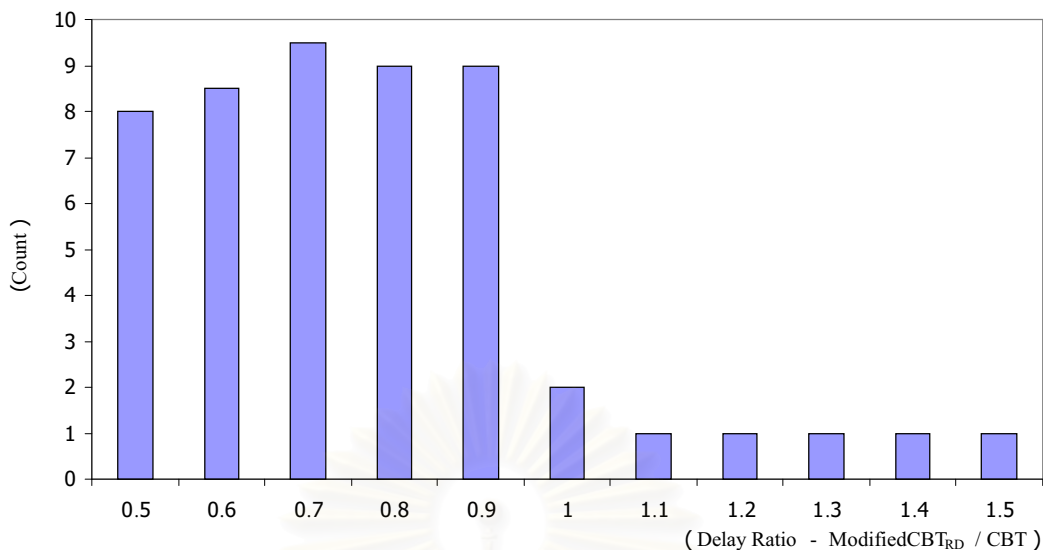


Figure 4.6: A Histogram of average delay in modified CBT_{RD} compared to its counterpart in CBT.

4.8.2 Network resource usage

A simple method to route a packet to all interested receivers is to unicast the packet to each receiver. However, unicasting is likely to route several copies of the same packet over links in the network. Multicast protocols send only a single copy of a packet over any link in the network and require fewer hops to deliver the packet than unicasting in most cases. Our simulation compares the modified CBT and CBT v2 in which protocol delivers a copy of a packet to all group members in the fewest number of hops. Figure 4.7 and figure 4.8 show the result of the network resource against Δ whereby multicast group size are set at 50 and 100. The simulation results show that the network resource usage of our proposed protocols are within 15% less than that of the CBT. With relaxed end-to-end delay values, all protocols can easily find less hop count path to each member. Hence, there are more alternate paths that minimize the number of hop counts which indirectly decrease the delay along that path. We conducted our experiment by varying different end-to-end delay time constraint values and found that our modified protocols, while complying to end-to-end time delay, use fewer hop counts than their counterpart. We performed another set of experiment to compare network cost at fixed Δ with different multicast group sizes. Figure 4.9 shows that as multicast group size

increases, our proposed protocols produces less costly network than that of the CBT. Our modified CBT_{RD} performs better than the modified CBT_{WD} in all cases in terms of network cost at all multicast group sizes. Modified CBT_{RD} can achieve up to 10% less network cost tree than that of the modified CBT_{WD} at a larger multicast group size.

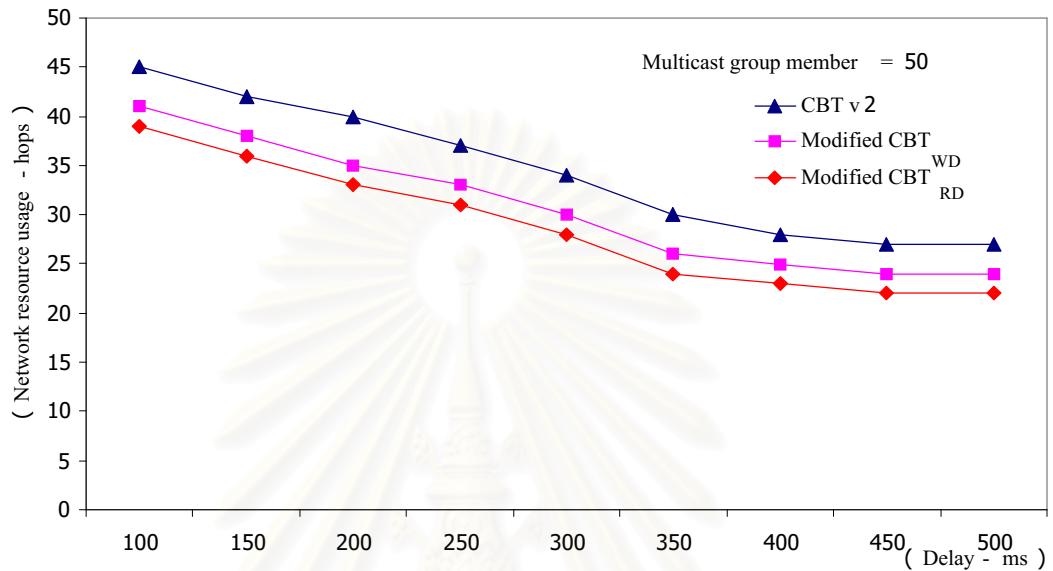


Figure 4.7: Network resource usage against end-to-end delay constraint values, multicast group member = 50.

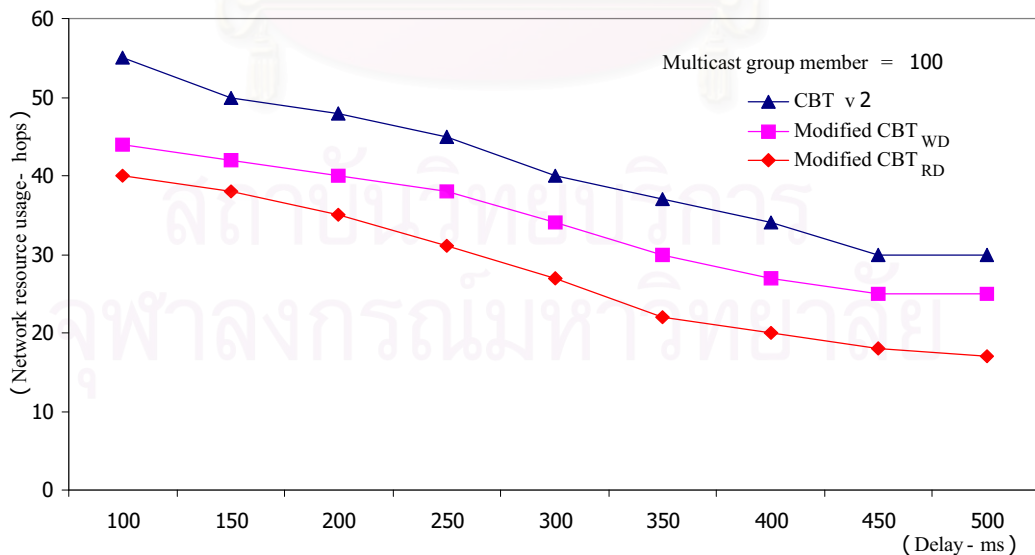


Figure 4.8: Network resource usage against end-to-end delay constraint values, multicast group member = 100.

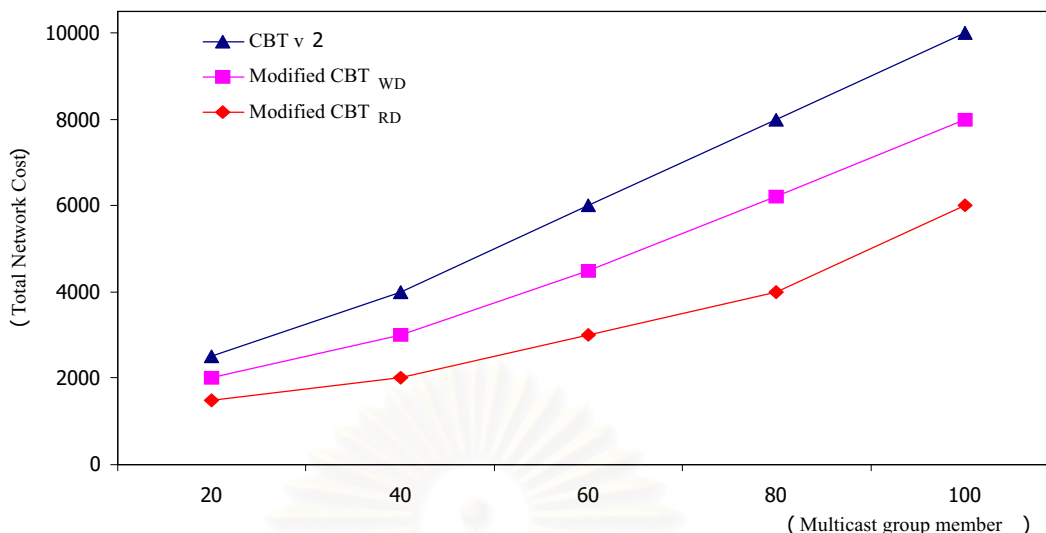


Figure 4.9: Network cost against number of multicast group size.

4.8.3 Traffic concentration

We define a flow to be the stream of packets from a particular sender to a multicast group. We assumed that each source of the multicast group generates traffic at constant unit rate. The total number of unit traffic flows that traverse a link is counted. Traffic concentration is measured by the maximum number of flows traversing a unidirectional link (the load of the most congested link). The overall rate of source traffic is at 1 Mb/s. We averaged out the simulation results of the multicast group size of 50 and 100. Figure 4.10, figure 4.11 and figure 4.12 show the traffic distribution in CBT, modified CBT_{WD} and modified CBT_{RD} respectively. Table 4.2 also summarizes the traffic flows in all cases. Note that in the CBT simulation, some link appears to be overloaded (450 flows) while others are underutilized (0 flows). On the other hand, the modified CBT_{WD} and modified CBT_{RD} show good link utilization. The maximum link load has decreased to more than half its value in CBT, which decreases the possibility of congestion. Modified CBT_{RD} performs well in distributing load to more links than that of the modified CBT_{WD}, which is the results of our residual delay path selection. It shows that the residual delay function can find better paths comparing to our Weighted Dijkstra's path selection.

CBT Max link load	Modified CBT _{WD} Max link load	Modified CBT _{RD} Max link load	Ratio CBT _{WD} /CBT	Ratio CBT _{RD} /CBT
432	180	174	0.42	0.40
448	189	180	0.42	0.40
431	170	170	0.39	0.39
438	185	181	0.42	0.41
444	180	170	0.41	0.38
445	179	169	0.40	0.38
450	200	190	0.44	0.42

Table 4.2: The Ratio of the Maximum link load in modified CBT_{WD} and modified CBT_{RD} to that in CBT

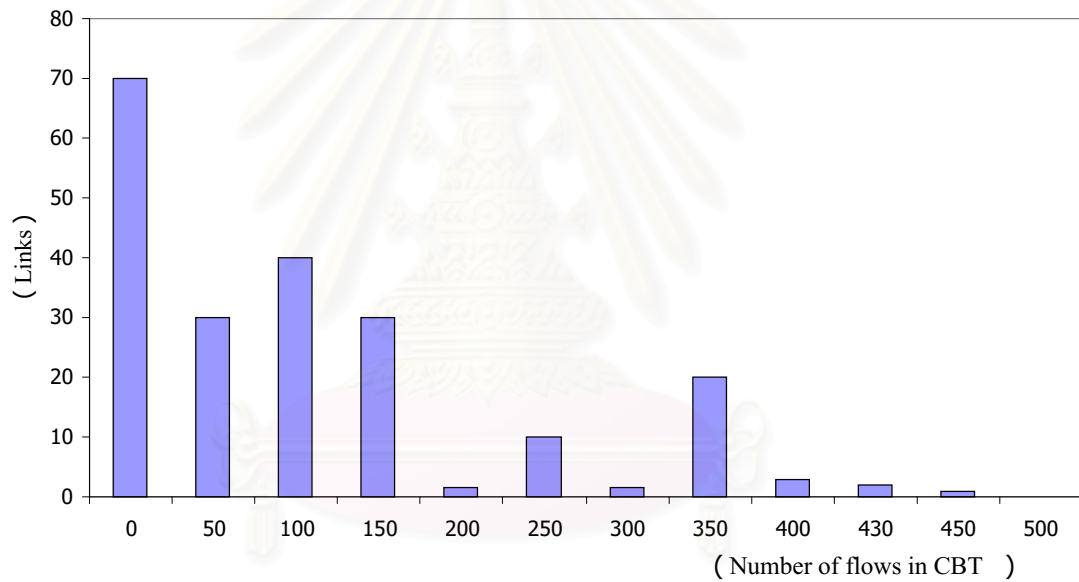


Figure 4.10: Traffic Concentration in CBT.

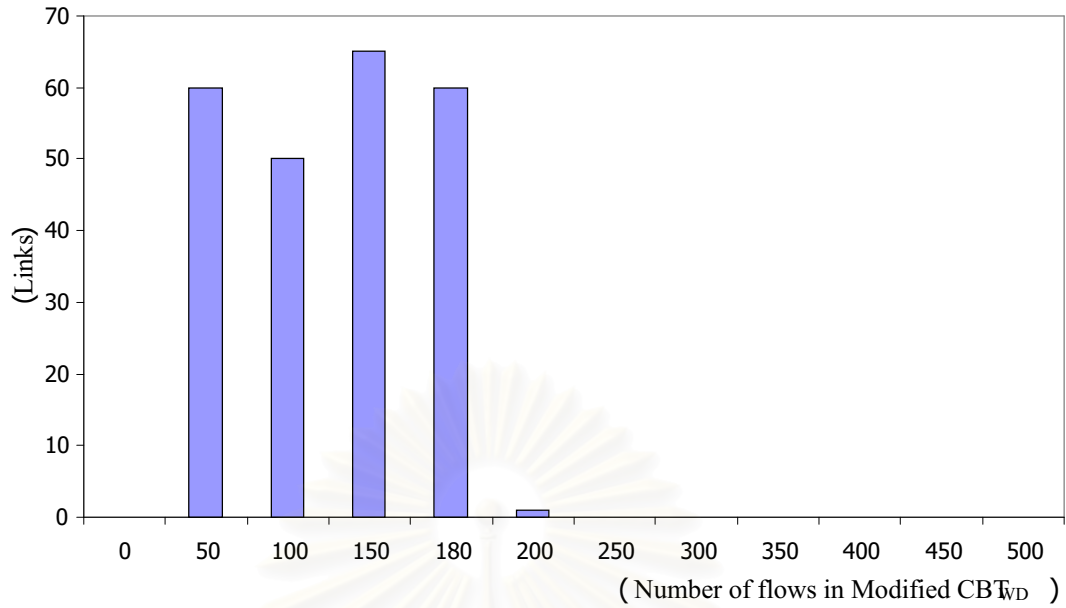


Figure 4.11: Traffic Concentration in Modified CBT_{WD}.

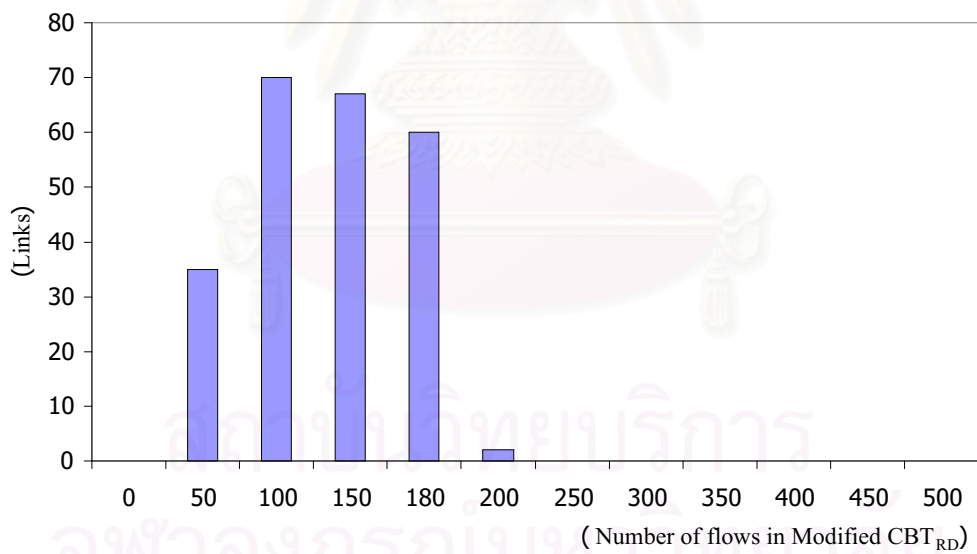


Figure 4.12: Traffic Concentration in Modified CBT_{RD}.

4.8.4 Loss Rate

We measured the loss rate of the packets by examining the packet arrival time at the destination node versus the time the packet was generated at the source node. The

number of packets arrived within the time bound, which is the real-time constraint was recorded. Since delay bound violation is one of the reasons to reject a multicast tree.

Therefore, loss rate is defined as the ratio of the total number of loss packets or late packets, to the total number of packets transmitted. The results, which are given in figure 4.13 and figure 4.14, show that our proposed protocols have up to 15% less loss rate compared with the loss rate in CBT. As the end-to-end delay becomes less stringent, the loss rate declines. We compare the loss rate performance of both protocols for different multicast group size of 50 and 100. We can see similar results in both cases.

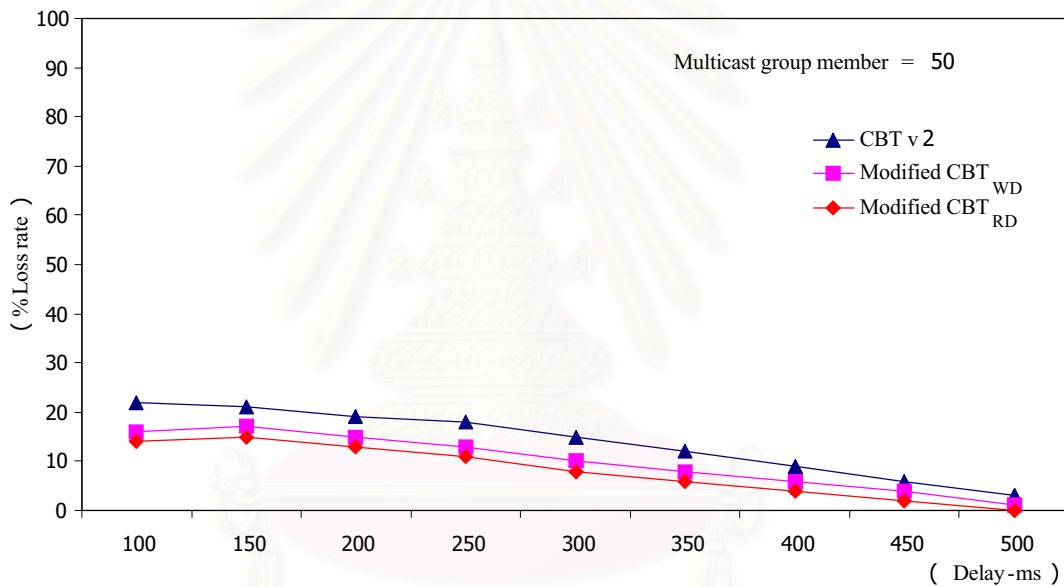


Figure 4.13: Loss rate against end-to-end delay constraint values, multicast group member = 50.

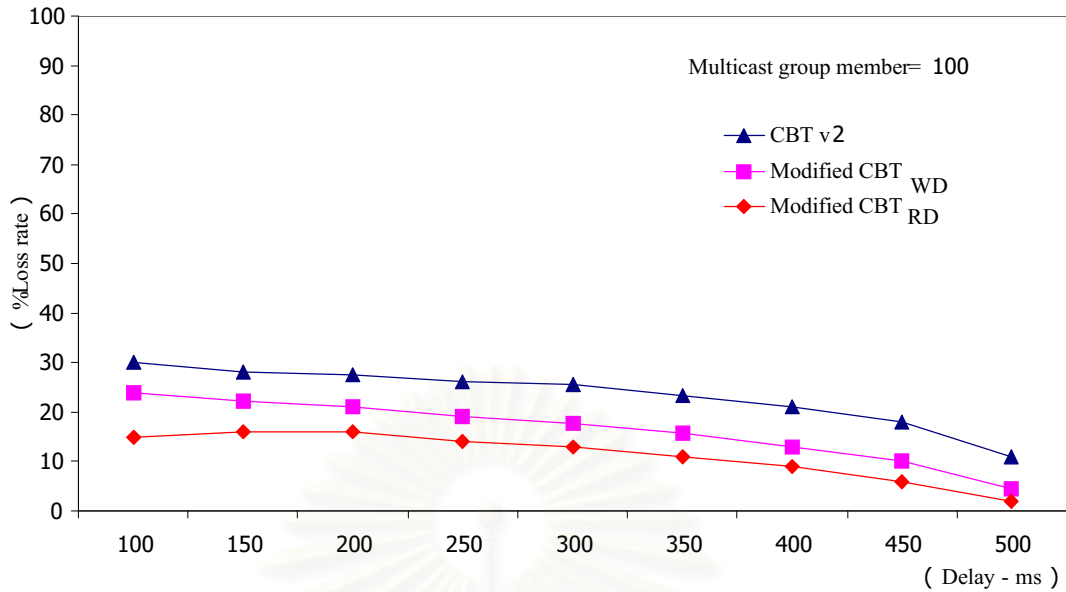


Figure 4.14: Loss rate against end-to-end delay constraint values, multicast group member = 100.

4.8.5 Execution Time

In figure 4.15, we present the average execution time of all algorithms, namely CBT, modified CBT_{WD} and modified CBT_{RD} with the multicast group sizes up to 100 members. Note, however, that our code for these algorithms was not optimized for speed. Thus, the improvement of the execution time can be further improved using the optimized code. We observed from the experiment that the average execution times of all heuristics grow at the same rate and are always within the same order of magnitude, with CBT being constantly slower than both modified CBT_{WD} and modified CBT_{RD}. Both our modified CBT heuristics are slower than CBT, which is the result of CBT tree refinement with real-time delay constraints. The running time of our modified CBT_{RD} is quite large and grows relatively faster than that of the modified CBT_{WD}, as the multicast group size increases. CBT is obviously quite efficient with larger multicast group in terms of its fast execution time.

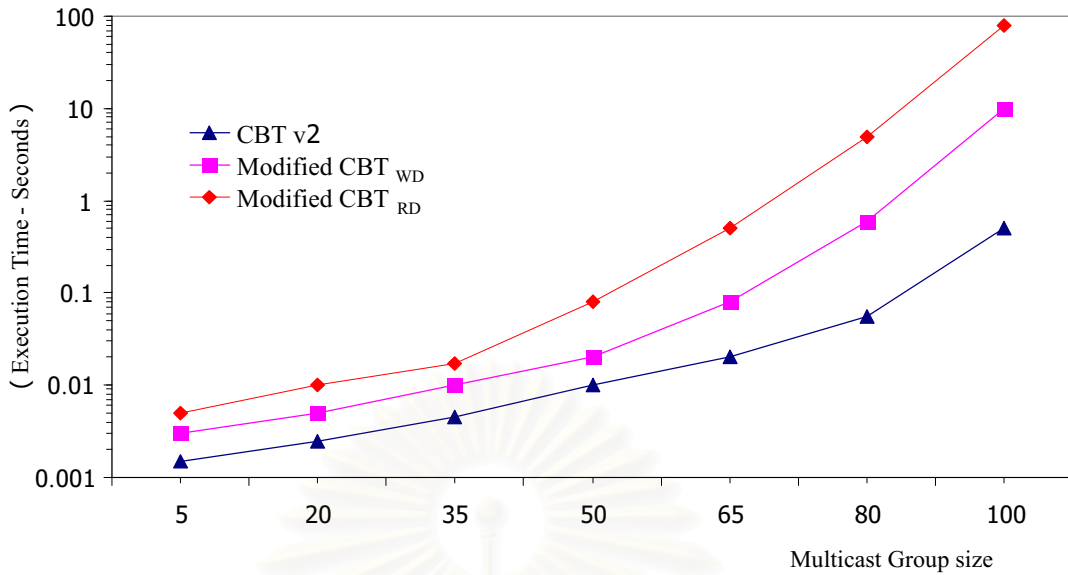


Figure 4.15: Execution time against multicast group size.

4.9 Conclusions

In this chapter, we proposed a new multicast routing protocol, based on the Core-Based Tree, whereby, shortest of the shortest paths or a “non-core” scheme has been incorporated. With time delay of real-time traffic imposed as one of the protocol constraint, our proposed approach which is based on the idea of Core-based trees version 2, are built with bounded time delay thus, sustaining the real-time traffic requirements of the applications. In this chapter, we also proposed two path selection methods which take into account of both the cost and the delay. These path selection functions can be applied to various multicast tree building algorithms to build the optimal multicast routing protocols. We incorporated the weighted path selection function into the Dijkstra’s shortest path algorithm. Then we proposed a residual delay function as another path selection function, and substitute this path selection function in the Dijkstra’s shortest path algorithm.

In comparison with the conventional CBT protocol version 2, a simulation result is presented to prove the efficacy of our protocol. Our protocol has a critical advantage to the original CBT version 2, in that the core node selection problem does not need to be addressed. Also, it is difficult to select an optimal core node since the locations of

group members are not known a priori for most real-time applications. In addition, the performance of the CBT strongly depends on the selected core node. However, with our proposed protocol, the network requires an additional capability whereby all nodes must maintain the current tree information, since one of the current on-tree nodes is selected as a distribution point for each new incoming member. This multicast node is selected such that the length of the path from the incoming user to a multicast node and the maximum end-to-end delay on the tree are jointly minimized. Thus, the proposed protocol can overcome the drawback of CBT. In the experiment, we considered a static set of multicast group members. Allowing nodes to join and leave an existing multicast group dynamically is another feature that should be considered in our future work, as well as the investigation of the network resources such as sizes of the routing tables maintained and the average number of messages exchanged between nodes for each topology changed.

We anticipate that additional work is needed to minimize the memory and processing power of each node maintaining the current tree information. The concept of hierarchical routing can be applied in this context because the underlying routing mechanisms used to disseminate topology data require the aggregation of information in order to cope with growing network size. Our proposed protocol can also be extended to have fault-tolerant capability, in case of a node failure or “core” router failure which is the subject of chapter 5. In the next chapter, we will investigate and study the fault-tolerant concept for real-time traffic, and find the strategy to couple the fault-tolerant features into our proposed protocol.

CHAPTER 5

FAULT-TOLERANT PROTOCOL

5.1 Introduction

Survivability has been an important topic of research in the context of telecommunication networks and data networks [61]. This interest has been fueled by the commercial importance of providing fault-free service to the users of the telephone network, and the military advantages of fault-tolerant distributed computing and data communications. However, real-time multicast applications makes it harder for the recovery of the network, which is subject to the time delay constraints imposed by the application between sender and many receivers belong to a multicast group.

The fault recovery mechanisms are intended to restore real-time connectivity with the original performance guarantees to the affected links. However, when a fault occurs, depending on the load in the network, all the affected links may not be successfully routed. Rerouting as much traffic as possible is an important objective, and this defines another metric for comparison of recovery schemes. The effect of the rerouting process on the ability of the network to accept more channels immediately after recovery process terminates is also a consideration. We would like the recovery process to used resources as efficiently as possible. The objectives of fault recovery are as follows:

- Use no extra resources during normal operation
- Restore original performance guarantees to channels or link between sender and receivers
- Recover as much traffic as possible within deadline (maximum end-to-end delay)
- Minimize switching time from primary path to the backup path
- Use resources efficiently during rerouting

The rest of the chapter is organized as follows. We investigate three rerouting approaches in case of faults in the next section. Section 5.2 describes the failure model and assumptions used in this dissertation. In section 5.3, we present the fault-tolerant concepts using dispersity, redundancy and disjointness. Section 5.4 and 5.5 present the proposed fault-tolerant multicast approaches, followed by the fault-tolerant real-time multicast techniques and some brief discussion on DCM routing algorithm [80], used to find disjoint backup paths under real-time constraints in section 5.6. We go on to present the approach to cope with multiple link faults in section 5.7. Section 5.8 concludes the chapter.

Approaches to rerouting are classified along three schemes: Centralized vs. Distributed Schemes, Pre-computed vs. Dynamic (on-demand) computation of routes, and Global vs. Local Knowledge Schemes.

5.1.1 Centralized vs. Distributed Schemes

A centralized scheme gathers all the important network state at one computer, where a new configuration for the network is computed. Since all the information is available, it is possible to compute a configuration that is optimal according to some target function. This optimal scheme might be able to reroute more channels successfully under heavy-load conditions. However, this gain comes at the expense of a possibly higher delay, since the time to gather the network state to one location and then distribute the network configuration to the routers must be added to the time to run the algorithm. The time to gather the network state has to be considered here, since, unlike the cross-connect networks, real-time networks have a very dynamic load, with connections coming and going as applications are started and ended.

Centralized algorithms are susceptible to single-point failures, since all recoveries would be stalled by a failure of the central computer, or of the control network which connects the central computer to the routers. In the telecommunication networks, this problem is mitigated by having more than one fault recovery computer and having redundancy in the control network, so that the control network itself is fault-tolerant. However, this increases the cost of the solution. We do not assume a separate control network; rather, the recovery action is carried out over the data network itself. However,

this makes the centralized approach much less desirable, since the failures in the network that we need to recover from, make collection and distribution process itself unreliable. The distributed algorithms deal explicitly with the problem of failed links. For example, the exchange of control messages with neighbors may be used as part of the process to determine the working topology of the network, since the inability to exchange messages indicates a failure of the corresponding link.

A distributed algorithm also scales better to larger networks, since the processing power available increases with the network size. The centralized scheme, on the other hand, must run an optimization algorithm of exponential complexity on a single computer. Thus, the major disadvantage of using a centralized solution (i.e., finding an optimal solution) is infeasible for all but the smaller networks. Thus, in this dissertation, we adopted the distributed fault recovery scheme, with local network topology information.

A distributed algorithm would find a non-optimal solution based on a limited exchange of information faster than a centralized scheme and without relying on a single central computer. This comes at the expense of possibly finding a solution considerably worse than the optimal. Since the distributed elements may have inconsistent views of the network, the overall behavior of the system may be hard to understand for an implementor, and may be far from the expected and desirable behavior, even though each individual element has a simple and understandable behavior. In spite of these disadvantages, this approach appears more feasible for the case of real-time networks, and will be explored here.

5.1.2 Pre-computed vs. Dynamic (on-demand) computation of routes

The task of rerouting may be partially pre-computed in order to decrease the response time of the recovery after a failure. This idea has been explored in the context of the cross-connect layer of telecommunication networks. But in the context of real-time communication networks, multimedia traffic is expected to be much more dynamic than the configuration presented by the cross-connect connections. A trunk is a relatively long-lived connection. As such, the requirements of a set of trunks, characterized by source, destination, and bandwidth, are fairly static. After the set of required trunks has

been decided, the current configuration, as well as the recovery configurations for all likely fault combinations, may be computed and stored. On the other hand for a real-time network, the set of active connections changes much more dynamically.

Most previous research on fault-tolerant, QoS routing has investigated on-demand policies that compute a path at connection arrival. Recent works considers pre-computation of backup resources that attempt to amortize the overheads of route computation by reusing the paths for multiple connection requests. Thus, backup path pre-computation introduces a trade-off between processing overheads and the quality of the routing decisions. However, in our dissertation, the concept of pre-computed backup links will be explored.

5.1.3 Global vs. Local Knowledge Schemes

The extent to which knowledge is universally shared among the elements of a distributed algorithm can have a tremendous impact on the performance and correctness of the algorithm. At one extreme, we might allow all nodes to gain complete information about any change in the network state before computing the next step in the algorithm. Since this would imply that all nodes have completely up-to-date knowledge of the network state and topology information throughout the reconfiguration process. This is equivalent to a centralized algorithm in terms of goodness or optimality of the solution. At the other extreme, we may choose to allow all nodes to run their reconfiguration algorithms with the view of the network state that they had at the moment of the fault, without allowing any additional communication. This would have the fastest response time of any algorithm, but would reroute fewer connections successfully, since the inconsistencies among the network views of the separate elements would lead to poor cooperation between the nodes. Thus, by allowing more time for communication, we can improve the success of the rerouting algorithm. In our formulation, we assumed that all on-tree nodes maintain global information of all multicast group information. However, at rerouting time, only the local routers involved in the failed link may be reconfiguration rather than reconfiguring all the routers. We then just reconfigure the routers that are on the involved backup paths.

5.2 Failure Model

We place certain restrictions on the failures being studied and investigated. We only focus our attention to a single link failure in this dissertation. Node failures can be considered more severe and treated as simultaneous multiple link failures of all links connecting to the failed node, which is not the subject of this dissertation. We assumed that a single link failure is a fail-stop failure, in which the failed component does not continue to transmit bad messages into the rest of the network. And after a single link fault period is over, it is possible for another (single) link fault to start, but not simultaneously. We further assumed that when a link fails, it simply stops transmitting data.

5.3 Fault-tolerant capability

There exists a body of graph-theoretic work that is useful in characterizing the fault-tolerant capability of the network. A network is said to be *k-node* connected if every source and destination is connected by *k node-disjoint paths* [8]. The edge-connectivity of a graph is the equivalent measure if we consider edge-disjoint paths. A minimal cutset of a graph is the smallest number of links, the removal of which breaks the graph into two disconnected components. Similarly, an articulation set is a set of nodes. The removal of which break the graph into two disconnected components. It can be shown that the cardinality of the minimal cutset of a graph is the same as its edge-connectivity, and the cardinality of the minimal articulation set is the same as the node-connectivity. The problem of designing minimal-cost networks with a given degree of connectivity is known to be NP-hard, but known heuristics and algorithms with good average case behavior can solve many real-world network design problems [42]. Tree structure of our fault-tolerant protocol is shown in figure 5.1 as follows.

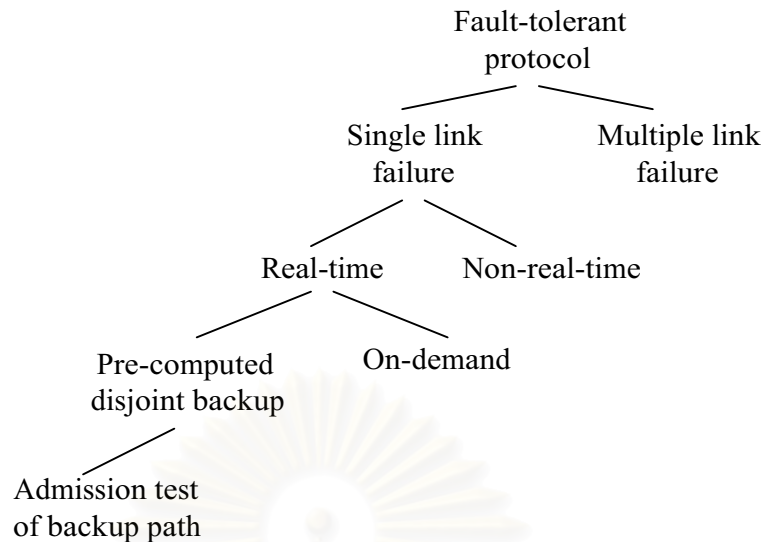


Figure 5.1: Tree structure of our fault-tolerant protocol

The schemes of fault tolerance can be characterized by three approaches: dispersity, redundancy, and disjointness.

5.3.1 Dispersity

Dispersity is the idea of sending information across a number of paths in the network. In a real-time network, we also make reservation on each of the paths to guarantee performance on the dispersity system. The information to be sent is divided uniformly across the paths, either by fragmenting each message, or by sending them round-robin on the paths. The advantages of spreading the information out on N paths are:

- The load on any one specific path is smaller and the effect of bursts is spread out over the network.
- In the event of a network failure, the transmission capacity of the aggregate system is only partially affected.
- If the message is fragmented, transmission time is reduced to $1/N$ th of its single-path value. This is less important in high-speed networks.

However, this system is not transparently fault-tolerant. It merely reduces the effect of the failure on the client. For example, a JPEG video stream, partitioned into multiple streams such that all the data for a frame follows the same path, but separate frames are

sent on different paths, would have this property. If one of the paths fails, every N th frame would be lost, but the resulting video stream would be usable. The number of paths chosen, N , is one of the variables that characterized a dispersity system.

The basic idea behind the dispersity schemes is to make reservations on multiple paths through the network so that in the event of failure the available bandwidth of the system is not reduced to zero. A range of fault tolerant services can be offered based on this idea. For example, Forward Error Correction (FEC) techniques can be used in conjunction with multi-path reservation to provide a transparently fault tolerant service.

5.3.2 Redundancy

Redundancy is the idea of sending more information than the message, in order to be able to reconstruct the message in the event of loss in the network. In a dispersity scheme, we can send the redundant information along a separate path from the rest of the data. For example, of the N paths, only K may carry the message stream. Thus, for a given message, the system could break it into K equal submessages and transmit them on the K paths. The rest of the paths may be used to transmit redundant information, which would be used to reconstruct the original message in the event of loss of some of the K original pieces of the message. A simple redundant system can be designed with $K = N - 1$. In this case, the single redundancy sub-channel carries a bitwise parity calculated over the $N - 1$ pieces of the message. If any one of the sub-messages is lost, the destination can compute its value from the remaining sub-messages and the parity sub-message. Error correction codes which work for arbitrary N and K exist. In the case of maximum distance separable codes, if any $N - K$ of the sub-messages are received, the message can be recovered.

The variable K in relation to N defines the degree of redundancy. $K = 1$ corresponds to duplicating the same information on N channels, uses N times the bandwidth of a non-fault-tolerant realtime channel, and has the largest fault tolerance. $K = N$ corresponds to a dispersity system without redundancy. A dispersity system with redundancy requires N/K times the bandwidth of a non-fault-tolerant realtime channel with the same traffic and performance requirements. When redundancy is used in

combination with dispersity, we get the following additional advantages as compare to a dispersity system without redundancy:

The system is tolerant to transmission errors. A certain number of the pieces of the message can be corrupted or lost without affecting the decoding of the message. The number depends on N , K , and the error-correcting code. It can be no larger than $N - K$.

The system is transparently fault-tolerant. A certain fraction of the paths can be affected by failure, without interrupting the flow of the information. Again, the specifics of the error-correcting code determine the number of failures that can be tolerated.

Moreover, since the service of the underlying real-time channels is realtime, we obtain performance bounds on the service provided by the dispersity system. The application sees fault tolerant realtime service, with guarantees on packet-delivery that continue to hold in the presence of a restricted number of faults. The restriction on the number of faults covered depend on the level of redundancy of the system, and the nature of the FEC code used.

5.3.3 Disjointness

In general, the routing algorithm in the network must be able to recognize channels belonging to a dispersity system, and place them on disjoint paths. If the paths are not disjointed, the failures of the paths are no longer independent, since, if a shared link fails, two or more paths can simultaneously stop transmitting data. However, disjointness is a very stringent restriction, especially as N approaches the degree of edge connectivity of the network topology. By allowing some links to be shared, we might be able to set up many more connections in the network. We would like to answer the question; can any of the advantages of dispersity routing still be provided after relaxing the disjointness criterion? If the constraint is completely relaxed, then it is possible for a link to be shared by all the paths in a dispersity system, leading to the undesirable characteristic that if that link fails, the capacity of the system is reduced to zero. Therefore, we must put in some constraint, which while looser than the strict disjointness constraint, should still allow the dispersity system to have good tolerance characteristics. If we allow a link to be used by at most two channel routes, then we

know that a single failure will not affect more than two of the paths. Then, by imposing the restriction $N-K \geq 2$, the system can continue to be tolerant to single faults, assuming maximum distance separable codes. In terms of our characterization, we can define a variable S , which places a limit on how many paths can share a link. The routing algorithm in the network must take the variable S into account while routing channels which belong to a dispersity system. Of course, even if the system allows links to be shared, the routing algorithm should try to find paths which do not share links, and only use paths with shared links if no disjoint paths meeting the delay constraint exist.

5.4 Fault-Tolerant Multicast Protocol

For any non-core router in CBT, if its parent router or path to the parent fails, that non-core router has one of two options for failure recovery. It can either attempt to re-join the tree by sending a JOIN-REQUEST to the highest-priority reachable core, thus keeping the failure transparent to the rest of the downstream branch. Alternatively the router subordinate to the failure can send a FLUSH-TREE message downstream, thus allowing each router to independently attempt to re-attach itself to the tree, possibly via a better route than previously. Routers must always attempt to join the highest priority reachable core. Fault-tolerant multicast protocol is responsible for detecting faults and reconfiguring the multicast network once faults are detected. Thus, it provides necessary infrastructure for the multicast protocol to deliver multicast packets. Consider that at runtime a component on the CBT becomes faulty. Alternative routes for the multicast packets, must be utilized. A predefined backup paths will be used to bypass the faulty components. This approach is considered “local approach” as opposed to the “global approach” which core-based trees are rebuilt and all routers are reconfigured. Then, all the packets that were supposed to be transmitted over the faulty link will be routed via the backup paths.

The basic idea of our fault tolerant protocol is as follows:

Each router (R) on the tree keeps monitoring the status of the upstream link. If a nonalive status is detected, then either the link and/or the father have failed. This means that the subtree rooted at R is disconnected from the rest of the tree. To repair the damaged tree, we connect R to the rest of the tree by its backup path. In summary, the

fault tolerant protocol performs the *Initialization* tasks (to select backup paths), *Fault Detection* tasks (to continuously monitor the status of upstream link router if it is in the faulty state), *Backup path invocation* (to start notifying the state of faults to all the routers on its backup paths) and *Router Configuration* tasks (to reroute all traffic via the backup paths).

5.4.1 Initiation tasks

To select the backup paths, this can be done with a global approach and a local approach. With global approach, all routers in the network will be informed of the faulty status. The core-based trees will be rebuilt and all the routers will be reconfigured. The runtime overhead including the notification of the faulty state and reconfiguration of routers may be too large to make this approach practical.

Rather than rebuilding the core-based tree and reconfiguring all the routers, (which is considered “global approach”) predefined backup paths approach, a local approach, will be used. Backup paths are identified off-line, using the DCM algorithms [78] which will be discussed in section 5.6.2. So at runtime, only routers on the backup paths are required to be reconfigured. The local approach is simple and involves small runtime overhead in comparison with the global approach. The routers on a backup path can be divided into three kinds, namely *owner*, *terminator*, and *on-path* routers. The *owner* is the first router on the backup path that will initiate the invocation of the backup path when the owner router detects a fault. The *terminator* router is the router at the other end of a backup path. The *on-path* router is the router in between the owner and the terminator.

5.4.2 Fault Detection tasks

When a network component, such as a router in the multicast network fails, we assume that the faulty state of the router can be detected by its neighboring routers. This can be achieved by a “keep-alive” mechanism operating between adjacent router. A keep-alive mechanism may be implemented by ICMP echo request/reply messages. We assume that each router is continuously monitoring the status of upstream link and router. For any non-core router, if its parent router or path to the parent fails, that non-core router

has one of two options for failure recovery: it can either attempt to re-join the tree by sending a JOIN_REQUEST to the highest-priority reachable core, thus keeping the failure transparent to the rest of the downstream branch. Alternatively, the router subordinate to the failure can send a FLUSH_TREE message downstream, thus allowing each router to independently attempt to re-attach itself to the tree. Routers must always attempt to join the highest priority reachable core.

5.4.3 Backup path invocation tasks

For the owner router, if a nonalive status is detected, the owner router starts the process of invoking the backup path. The owner waits until a positive confirmation message comes back. Once receiving such a message, it reconfigures its routing table to reflect the usage of the backup path. After it is done, it must go back to monitoring the faulty state again.

For the on-path router, the on-path router will wait for the message of setting up the backup path. Upon receiving such message, it still forwards it upstream and waits for the positive confirmation message. When this message is received, the on-path router forwards it downstream and reconfigures itself to reflect the usage of the backup path.

For the terminator router, after the initialization, it waits for the message of setting up the backup path. Upon receiving such message, it sends the positive confirmation message back downstream and reconfigures itself to reflect the usage of the backup path.

5.4.4 Router Configuration tasks

After all the routers on the backup path have confirmed their backup route, they will be configured to reroute the traffic on their backup paths. As new routes are established, the network state at the nodes change. So the routing tables on all nodes must be updated to reflect the changed network state. Since the success of a following establishment attempt depends on the accuracy of the routing tables on which the route computation is based, the mechanism to maintain the routing information is important.

5.5 Backup Path selections

We propose the real repair method in such a way that all the routing tables of routers on the backup path will be changed to reflect the new topology of the tree. Packets are routed in accordance with the newly adjusted routing table. More detail on real repair and virtual repair method can be found in section 5.5.2 and section 5.5.3.

5.5.1 Backup Core Router

For reasons of robustness, we need to consider what happens when a primary core fails. There are two approaches we can take, namely: single-core CBT trees and multiple-core CBT trees. For the case of single-core CBT trees, paths as well as cores themselves can fail, which may result in parts of the network being partitioned from others. In the presence of CBT trees, this means a single tree itself becomes partitioned. To cater for tree partitions, we have multiple “backup” cores to increase the probability that every network node can reach at least one of the cores of a CBT tree. At any one time, a non-core router is part of a single-core CBT tree. For multi-core CBT trees, each core is then strategically placed where the largest “pockets” of members are located so as to optimize the routes between those members. Each of the cores must be joined to at least one other, and a reachability/maintenance protocol must operate between them. There exists no ordering between the multiple cores, and senders who send multicast preferably to the nearest core. The essential difference between multi-core and single-core trees is that single-core trees have no explicit protocol operating between the “backup” cores and for any sender at any instant, there is one core to which it must attempt to send its multicast packets. We assumed in our formulation, single-core trees are more appropriate with less complex failure scenarios and less overhead of an explicit protocol operating between the cores. The basic idea of backup core selection is as follows:

For any two routers (R and R') on the core-based tree, R is a son of R' and R' is the father of R if there is a link between R and R' , and R' is closer to the core than R . R'' is the grandfather of R' if R'' is the father of R' , and R' is the father of R . Core router has neither father nor grandfather. The sons of the core router have a father but have no

grandfather. All other routers have both father and grandfather. One of the sons of the core is selected to be the backup core that will become core if the core fails. In practice, it may be selected from network administrative point of view, as suggested in selecting core [5].

5.5.2 Backup Paths with virtual repair

With our repair method, every router on the tree except the core, owns a predefined backup path. For a router that has a grandfather, its backup path is a path that connects itself to its grandfather. A constraint on the backup path of a router is that the path does not contain the father of the owner. For a router that has no grandfather, its backup path is a path that connects this router to the backup core. For the backup core router, its backup path is a path that connects itself to the core, by bypassing the link between itself and the core. We assume that for each router on the tree, at least one backup path exists. It is easy to verify that if a noncore router does not have any backup path, then the network is not single-fault tolerable. For a router, if multiple backup paths exist, we select the one with the shortest distance. With this method, no routing table is to be changed on the invoked backup path. Instead, a preprogrammed agent will be installed at the two end routers. The agent will encapsulate a multicast packet, which was supposed to be transmitted via the faulty component. The encapsulated packet will be source routed (via the backup path) to the other end of the backup path, once receiving the encapsulated packet, will unencapsulate it and transmit along the normal paths where the packet should be dispatched. Thus, the topology of the multicast tree is virtually unchanged, except that the faulty component is bypassed.

5.5.3 Backup Paths with real repair

With virtual repair method, the shortest path from a router to its grandfather is used as its backup path. However, if the backup path of a router transverses another partitioned subtree, a loop may occur [5]. Thus, the selection of backup path with the real repair method is not a trivial task. In particular, a loop in the network must be prevented. With this method, all the routing tables on the backup path will be changed to reflect the new

topology of the multicast tree. Packets are routed in accordance with the newly adjusted routing table.

Figure 5.2 shows an example of using these two methods. Figure 5.2a shows a portion of the network with the original core-based tree. Assume that there is a fault on the link between R4 and R6. Let the backup path that is used to reconnect the disjoint tree be $\langle R_6, R_5, R_3 \rangle$. Figure 5.2b shows the situation after virtual repair. In this case, the agent on R3 will encapsulate the multicast packets and source-route encapsulated packets to R6 via R5, and vice versa for the packets from R6 to R3. However, R3 still has to send multicast packets to R5. Hence, the load between R3 and R5 is doubled because the tree is virtually repaired. The situation improves when the real repair method is used as shown in figure 5.2c. In this case, R5 and R6 will be reconfigured to recognize that while R5 continues to be a son of R3, R6 is now a new son of R5. Hence, the packets between R3 and R5 will not be transmitted twice.

[46] observed that from their simulation, real repair method always performed better than that with the virtual repair method because the real repair method explicitly takes into account the new topology after a fault occurs and hence, better utilizes the system resources. Our formulation has adopted the real repair method over the virtual repair method as our reconfiguration strategy in selecting the backup paths.

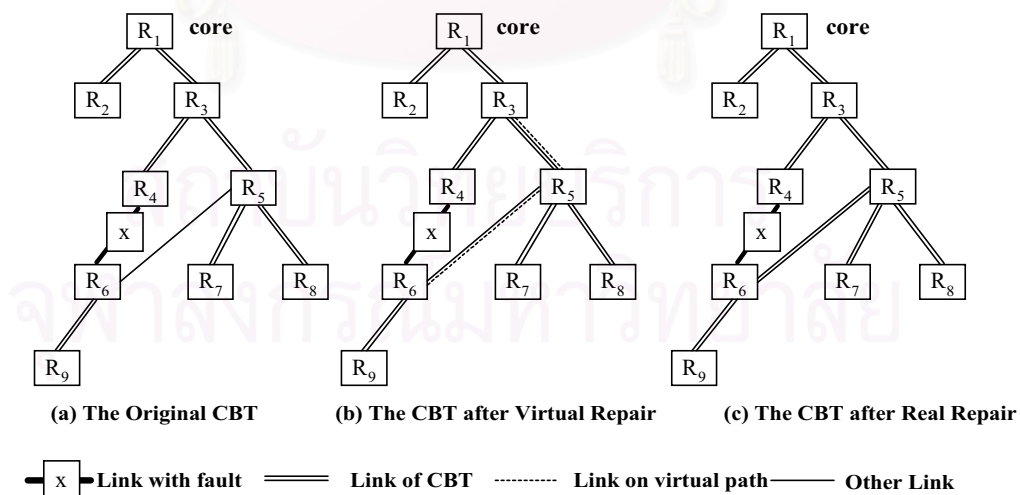


Figure 5.2: Reconfiguration method

5.6 Fault-tolerant in real-time communication

The delivery delay in a point-to-point switching network is difficult to control due to the contention among randomly-arriving packets at each nodes and multihops a packet must travel between its source and destination. It is even harder for the multicasted network to maintain consistent end-to-end delay for each source-destination pairs. Despite this difficulty, there are an increasing number of applications that require packets to be delivered reliably within specified delay bounds.

The approach of real-time channel, first proposed by Ferrari and Verma [38], is to provide delay bound guarantees to real-time connections. With the real-time channel approach, each packet is assigned a deadline over each link on its route and the transmission of packets over a link is scheduled according to their deadlines. Using a proper deadline assignment policy, the network will first serve those packets of the channels that require tight delivery delays and/or high link bandwidths. One important, yet under explored, problem is the fault tolerance associated with the concept of real-time channel. For ease in controlling the end-to-end packet delays, the static routing approach is used for real-time channels. All packets of a real-time channel are transmitted along the same path. This, unfortunately, is more susceptible to component failures than a dynamic routing approach, since a single component failure may disable the whole channel, while packets can be easily routed around the broken components with a dynamic routing approach. There have also been proposed forward-recovery approaches where multiple copies of a message are sent via disjoint paths to mask component failures. A variation of these approaches coupled with the error-correction coding scheme can be found in [30]. The methods proposed in [39] requires all failures to be broadcast to the entire network. When a source node is notified of the failure of its channel, it tries to establish a new channel from scratch. Since no resource is reserved in advance for the fault tolerance purpose, this method has a small overhead in the absence of faults. However, it does not give any guarantee on failure recovery. The channel reestablishment attempt for failure recovery can be rejected, even when there are sufficient resources, as a result of the contention among several simultaneous

recovery attempts. Moreover, successive attempts of channel reestablishment may extend the recovery delay significantly.

Most real-time communication schemes share three common properties: QoS contracted, connection-oriented, and reservation-based. A contract between a client and the network is established before messages are actually transferred. To this end, the client must first specify its input traffic behavior and required QoS. Then, the network computes the resource needs (e.g. link and CPU bandwidths, and buffer space) from this information, selects a path, and reserves necessary resources along the path. If there are not enough resources to meet the QoS requirement, the client's request is rejected. The client's messages are transported only via the selected path with the resources reserved, and this virtual circuit is often called a real-time channel. While this reservation-based approach has been successful in providing "hard" guarantees on timeliness QoS, it causes a serious difficulty in achieving fault tolerance (because the approach relies on static routing). Traditional failure-handling techniques for datagram services are inadequate, because a real-time message is allowed to traverse only the path on which resources are reserved a priori for it and, hence, cannot be detoured around failed components on the fly. Instead, a new channel which does not use the failed components should be established before resuming the data transfer. However, establishing a new channel is usually a time-consuming process, which can result in a long service disruption. Moreover, such an approach cannot make any guarantee on successful failure recovery, because there may not exist a proper detour.

To assure successful rerouting and avoid the time-consuming channel reestablishment process, one or more backup channels are set up *a priori*, in addition to each primary channel. That is, a dependable real-time connection consists of a primary channel and one or more backup channels. A backup channel remains a cold-standby until it is activated. In other words, it does not carry any data in a normal situation, so that the resources reserved for the backup channel may be used by other traffic. However, backups degrade the network's capability of accommodating real-time channels, because they reserve resources which can be used to accommodate other real-time channels otherwise.

5.6.1 Dependable Network

The idea of providing an efficient scheme to quickly restore real-time channels from link failures has been studied by many researchers. To assure successful rerouting and avoid the time-consuming link reestablishment process, one or more backup links are set up *a priori*, in addition to each primary link. This is called, a *dependable real-time network* or *dependable real-time connection*, consists of a primary link and one or more backup links. A backup link remains a cold-standby until it is activated. In other words, it does not carry any data in a normal situation, so that the resources reserved for the backup link may be used by other traffic. A dependable network is a network with at least two disjoint routing paths between any two nodes in the network G . In other words, a dependable network is a two-edge-connected according to the Menger's theorem [73] that the removal of any one edge does not disconnect G . We will investigate an algorithm that implements the concept of dependable network called "DCM routing algorithm".

5.6.2 DCM Routing Algorithm

In this dissertation, a routing algorithm used in determining and routing an alternate path is based on the DCM Routing Algorithm by Parris and Banerjea [79]. The first objective of DCM routing algorithm is to establish an alternate link, conforming to the specified QoS parameters, which in our case is the end-to-end delay constraints. This disjoint backup link is established in the presence of the primary link on which the host is currently active. When it encounter the failure situation, this backup resource will be invoked and go through admission test, to check if the network resource is readily available before the route changes (routing reconfiguration) from primary to backup link is established.

As mentioned earlier that, this algorithm determines a route from the source node to the destination node taking into consideration the traffic and the performance characteristics of the connection (i.e. end-to-end delay value). Another goal of the DCM routing algorithm is to balance the network load and obtain routes in a timely manner, also to maximize the probability that the route provided by the algorithm will be

successfully established. The routing algorithm calculates a minimal-cost route where the cost of the route is the sum of costs of the links comprising the route. The cost of a link is a delay value, which is the sum of the queuing delay, the transmission delay and the propagation delay.

The DCM routing algorithm proceeds in two steps. In the first step, a directed graph is created in which the nodes correspond to switches and hosts in the network and the edges to the links connecting these switches and hosts. The weights attributed to each edge represents the link cost. The link cost are computed just prior to applying the algorithm thereby using the most recent link information obtained from routing update messages. In the second step, a constrained, modified version of the Bellman-Ford algorithm is then applied to this graph to determine a possible disjoint backup links. In this algorithm, consecutive searches are performed on all $1, 2, \dots, N-2$ hop paths from the source to the destination nodes until the delay condition Δ is satisfied. The DCM routing algorithm maximizes throughput by minimizing the number of intermediate nodes encountered along the path from the source to destinations. Detail of DCM routing algorithms can be found in [80].

5.7 Multiple Link faults

The proposed fault-tolerant protocol was designed to handle a single link fault that occurs on CBT paths. We now discuss how to extend it to deal with the cases of multiple faults that will not only impact paths on the CBT, but also off-tree nodes. Let us first discuss the extension of our designed protocol so that even if multiple simultaneous link faults occur, the CBT can still function properly. The major extension has to deal with the methods of disjoint backup paths selection and backup path invocation. In order to tolerate m simultaneous faults, a trivial extension to backup paths selection is to select m disjointed backup paths for each router. These paths connect the router to its grandfather. However, this is insufficient, for example, if $m = 2$, then both father and grandfather can fail. In figure 5.3, assume that both R3 and R4 become faulty and that R7 has two backup paths from itself to its grandfather R3: $\langle R7, R10, R11, R5, R3 \rangle$ and $\langle R7, R6, R3 \rangle$. Since R3 is faulty, these two backup paths cannot be used to reconnect the multicast tree. That is, the CBT will be broken. Thus, in order to tolerate m faults, for

each router, we need additional backup paths to connect not only to its grandfather but also other ancestors. For example, consider the network in figure 5.3. In order to tolerate 2 faults, R7 needs not connect to only two paths from itself to its grandfather, but also a backup path to the father of grandfather R1: $\langle R_7, R_6, R_1 \rangle$. This backup path will be invoked in the case that both R3 and R4 are faulty. Note that the two paths from R7 to R3 should be disjoint. But a backup path from R7 to R1 does not have to be disjointed with ones from R7 to R3.

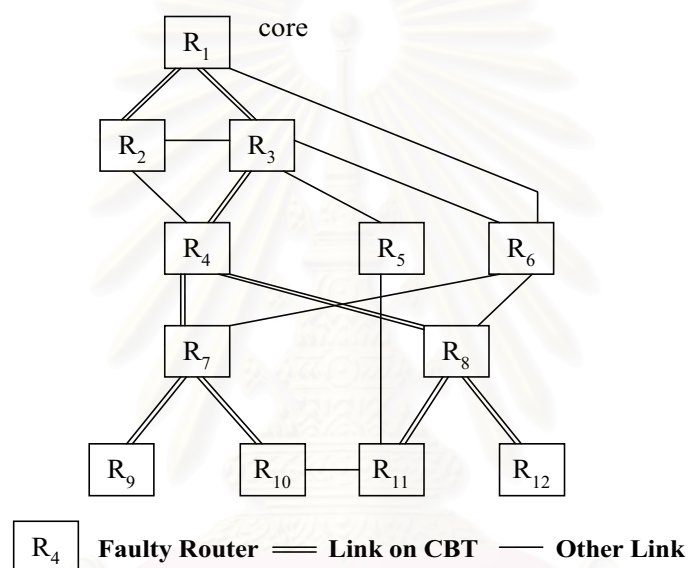


Figure 5.3: Backup path selection for multiple faults tolerance

Similarly, if $m = 3$, for each router on the CBT, we need one backup path to connect to the grandfather of grandfather two disjointed backup paths to connect to the father of the grandfather, and three disjointed backup paths to connect to the grandfather. Once again, note that the backup paths to the different ancestors do not have to be disjointed with each other. In general case, when $m = n$, for each router on CBT we need i ($i = 1, 2, 3, \dots, n$) disjointed backup paths to connect to the $(n+2-i)$ -th ancestor if it exists.

Now we discuss how to extend backup path invocation algorithms to handle the case of multiple simultaneous link faults. Since there are multiple faults and they may occur anywhere (including on some backup paths), invocation of a backup path may not always be successful. A timer should be used to time-out an invocation process.

This way, if indeed a fault happens on the path, the owner of the path can invoke another path.

The second issue of backup path invocation is loop elimination. Given that several routers may invoke several overlapped paths simultaneously, a loop could occur. This kind of loop can be prevented by properly trimming the involved backup paths during the invocation process. A simple rule for trimming is that whenever a path invocation message reaches an on-path router that has been configured due to the invocation of some other paths, the current invocation process terminates. That is, the rest of the path is trimmed. It can be easily proven that this method results in a correct CBT that is loop free.

Finally, we consider how to deal with faults that impact off-tree paths. To let an off-tree path recover from faults, we may take a similar approach as we did when we deal with faults that impact the CBT. This is because a simple path (from one router to another router) can be regarded as a special tree. Given that an off-tree path is the one used to route a message from its source to one node on the core-based tree, the selection of backup paths can take some advantages. That is, a backup path may not need to connect to the routers on the broken path. As long as the backup path leads to the core-based tree, the protocol will function correctly.

In summary, with these extensions our protocol can provide efficient and effective fault-tolerant communication in any faulty situation. However, our fault-tolerant protocol has to check if the end-to-end delay condition still holds. The more backup invocation, the less likely that the delay constraints can still be maintained.

5.8 Conclusions

In this chapter, we have studied the fault-tolerance framework and formulated the systematic study of recovery of real-time traffic. The fault tolerance framework was proposed which focus on rerouting traffic before the deadline expires, and minimizing the network resource used in the process. The approach to rerouting of the primary link to the backup link can be classified in: Centralized vs. Distributed Schemes, Link rerouting vs End-to-end rerouting, and Pre-computation vs dynamic (on-demand) computation of routes. We have proposed dispersity routing as a mechanism to provide

fault tolerance to real-time communication network. We investigated the rerouting of guaranteed performance service connections on the occurrence of link faults, focusing on the aspects of route selection, disjoint backup paths and routing table reconfiguration in the network. The fault-tolerance strategy of the original CBT was described. Our strategy to improve the fault-tolerance feature of CBT was studied. The concept of dependable network was discussed and studied. The DCM routing algorithm was proposed to find the disjoint backup path between each node pair, i.e., source and destination node in each multicast group. The algorithm determines an alternative path from the source node to the destination node taking into consideration the traffic and other performance characteristics of the connection. The DCM routing algorithm maximizes throughput by minimizing the number of intermediate nodes encountered along the path from the source to the destinations. When a single failure occurs, DCM algorithm also checks if the disjoint backup path, which was pre-computed, can meet the time delay constraint, and verifies if there is enough network resource, before rerouting all routers involved in the failure. Last, we analyse the multiple simultaneous fault situation, whereby multiple disjoint backup paths between the involved routers and to its grandfather must be specified. We conclude that to tolerate m simultaneous faults, for each router, we need additional backup paths to connect not only to its grandfather node, but also other ancestors.

We will implement the fault-tolerant protocol and principal mentioned throughout this chapter, and apply them to the proposed real-time optimal protocol based on CBT v2 mentioned in chapter 4, and proved by simulation the efficacy of our proposed approach which is the subject of chapter 6.

CHAPTER 6

FAULT-TOLERANT REAL-TIME MULTICAST ROUTING PROTOCOL

Overview

Development of efficient multicasting algorithms which support both real-time traffic and fault-tolerant capability are crucial to the successful deployment of multimedia applications. Emerging multimedia technologies introduce the prevalent multicast transmission, and the multicast tree is determined using the time-invariant network parameters. For reliable multicasting, some mechanism is required to recover from faults. If one of the group members fails or one of the nodes or links used in the multicast tree fails, the multicast tree can become disconnected. It is impossible for messages from the core to reach the members of the disconnected subtree. However, the recovery process which involves the rerouting of the existing traffic to a backup channel must be achieved before the applications' deadlines. In this chapter we propose the fault-tolerant multicast routing algorithm which is capable of determining an alternative path for real-time applications under a single link failure condition. We study the problem of core-based tree and propose an enhancement to guarantee end-to-end delay real-time constraints while sustaining the shared tree characteristics of scalability. Then, we propose a strategy to find the back up paths that meet the end-to-end delay guarantee and propose the algorithms to switch from the fault tree to the new tree so that the application end-to-end delay time is bound. The algorithm performs an admission test of the backup paths to verify that the end-to-end delay constraint of the real-time applications is not violated. Simulation results show that our proposed algorithm performed relatively well in terms of meeting the end-to-end delay constraints, network resource, traffic concentration, loss rate and execution times when there is a single link failure in the network.

The rest of this chapter is organized as follows. Section 6.1 and 6.2 give a brief introduction of the multicast routing algorithm. Real-time multicast routing is discussed with problem definition in section 6.3 and 6.4. Section 6.5 discusses our proposed fault-

tolerant protocol and the implementation of our fault-tolerant features to couple with the real-time multicast protocol previously proposed in chapter 4. We then describes the simulation setup to compare the efficacy of our proposed approaches in section 6.6. Section 6.7 gives the simulation results and discussion of the performance metrics. A conclusion to this chapter is presented in section 6.8.

6.1 Introduction

Multicasting refers to the transmission of data from one node (source node) to a selected group of nodes (member nodes or destination nodes) in a communication network. Instead of sending a separate copy of the data to each individual group member, a multicast source sends a single copy to all the members. Multicast routing requires the establishment of a multicast tree to allow group members to exchange data efficiently. In some case, the objective of constructing a multicast tree is to ensure that the QoS requirements of the underlying multicast traffic are met in addition to the data exchange to the group members. An underlying multicast routing algorithm determines, with respect to certain optimization objective, a multicast tree connecting the source(s) and the group members. Data generated by the source flows through the multicast tree, traversing each tree edge exactly once. As a result, multicast is more resource efficient, and is well suited for applications such as video distribution.

Multicast services have been increasingly used by various continuous media applications. For example, the multicast backbone (Mbone) of the Internet has been used to transport real-time audio and video for news, entertainment, and distance learning. With fast development of hardware technologies, commercialization of the Internet, as well as the increasing demand of quality-of-services (QoS) fueled by emerging continuous media applications, offering guaranteed and better than best effort services will add to the competitive edge of a successful service provider. The notion of QoS was proposed to capture the qualitatively or quantitatively defined performance contract between the service provider and the user applications. QoS provisioning entails the development of several essential techniques, i.e. definition and specification of QoS, design of QoS-driven (or termed elsewhere constrained-based) unicast/

multicast routing protocols, packet scheduling algorithms for link sharing, as well as resource reservation and management.

In general, a multicast communication session involves multiple sources transmitting to multiple destinations. This is the many-to-many multicasting problem. Videoconferencing is an obvious example of an application involving multiple sources and multiple receivers. The one-to-many multicasting problem is a special case of the many-to-many problem, in which the multicast session involves only one source. An example of one-to-many communication is real-time control application in which a sensor transmits its readings to more than one remote control stations. One approach to establish a many-to-many communication session is by setting up multiple one-to-many sessions. The host group model is defined as a multicast group that is a set of receivers, identified by a unique group address. The use of a unique group address allows logical addressing, i.e., a source needs only to know the group address in order to reach all receivers. It does not need to know the addresses of the individual receivers. In addition, the source itself need not be a member of the multicast group. Logical addressing is advantageous for applications with large numbers of sources and receivers, such as mailing lists or news groups, and for dynamic applications, such as computer-supported cooperative work, where receivers may join or leave the group at any time and sources may start or stop transmission to that group at any time.

Most real-time applications of computer networks, such as teleconferencing, remote collaboration and distance learning, rely on the ability of the network to provide multicast communication. These applications may require end-to-end delays, delay jitter, and loss rate which are expressed as QoS parameters which must be guaranteed by the underlying network. The upper bound on end-to-end delay from any source to any receiver in a real-time session is the main QoS parameter we consider during our investigation of various routing problems. In high-speed wide-area networks, the transmission delay is small and the queuing delay is also small, because small buffer sizes are used. Therefore, the propagation delay is the dominant component of the link delay. The propagation delay is proportional to the distance traversed by the link. It is fixed, irrespective of the link utilization. Therefore a route selection algorithm can guarantee an upper bound on the end-to-end delay by choosing the appropriated links

for the session being initiated, such that the delay from any source to any receiver does not exceed the delay bound.

Multicast trees can be classified into two categories: source-based trees and shared-based trees. Source-based tree is a tree composed of the shortest paths between the sender and each of the receivers in the multicast group. Multicast algorithms that use source-based tree require that a separate tree be constructed for each source in each multicast group. The share-based tree, on the other hand, uses a single shared tree for all senders in a given multicast group. This tree is a shortest path tree rooted at a specific node known as the Core. Share-based multicast routing provides a good mechanism for scalable multicasting since a single shared tree is created for all members of a given multicast group.

6.2 Multicast Routing Algorithm

Multicast routing requires some distribution tree rather than a simple point-to-point path through the network. The objective of multicast routing algorithms is to construct and maintain the distribution tree, called the multicast tree. The routing algorithms can broadly be classified as source routing and distributed routing. In the source routing, each router maintains the complete global state of the network. Based on the global state, the multicast tree is locally computed at the source router. In distributed routing, the tree is computed by an algorithm distributed over different routers in the network.

Most of the previous algorithms for constructing multicast tree generate a source-rooted tree for each (source, group) pair. These approaches are suitable for single sender/fixed recipient scenarios. However, for multiple senders/multiple recipient cases, it is more appropriate to use a single shared tree that can be used by all group members to send and receive the multicast packets. The Core Based Tree (CBT) algorithm is an example of this approach [6]. A single router (or a set of routers) is chosen to be the core router of the delivery tree. When a host wants to receive messages from and /or send messages to a multicast tree, it joins the cores of the multicast group. Since CBT constructs only one delivery tree for each multicast group, routers are required to keep less information as compared with other algorithms. CBT also conserves network bandwidth by forwarding packets only along the shared tree (it

does not use flooding). However, using a single tree for each group may lead to traffic concentration and bottlenecks around the core router. Selection and management (in case of the core failure) of the core router are additional problems. This approach has also been used in the PIM-SM protocol [29].

A multicast routing protocol uses one or more routing algorithms to construct and maintain the multicast tree. Some of the most widely used protocols are reviewed in this section. The Distance Vector Multicasting Protocol (DVMRP) is a distributed algorithm that dynamically generates a multicast delivery tree for each (source, group) pair using the RPM technique. It is an extension of the Routing Information Protocol (RIP) and can be summarized as a “broadcast and prune” multicast routing protocol. In this approach, the first datagram for any (source, group) pair is flooded across the entire Internet to create a spanning tree. The initial datagram is delivered to all leaf routers, which transmit prune messages back towards the source if there is no multicast group member on their directly attached leaf subnetwork. The prune messages remove all branches from the tree that do not lead to group members, thus creating a source-rooted shortest path (based on distance vector) tree with all leaves having group members. After a period of time, the pruned branches grow back (called grafting), and the next datagram for the (source, group) pair is forwarded across the entire Internet, resulting in a new set of prune messages. This method takes care of changes in the multicast group membership over time. DVMRP supports tunnel interfaces and is currently deployed in the majority of Mbone routers.

As discussed earlier, the CBT protocol builds a shared multicast distribution tree per multicast group. As all the routers connect to the core, the protocol may lead to traffic concentration and a performance bottleneck around the core router. Several core management approaches, e.g. core selection, core failure handling, and core migration [6] have been proposed to avoid these problems. Another solution to avoid the performance bottleneck is to use multiple cores. Unfortunately, when multi-core architecture is used, the CBT protocol can form loops and thus fail to build a connected multicast tree, even when the underlying routing is stable. The Ordered Core Based Tree (OCBT) protocol [92] eliminates these deficiencies. The performance bottleneck

around the core router can also be eliminated by allowing the new members to join any one of the on-tree routers instead of joining the core [57].

We summarize several multicast routing algorithms that can be used to solve the problems in previous section. A taxonomy of these multicast routing algorithms is given in table 6.1.

Algorithm		Centr./Dist.	Initiator	Tree Type	Complexity	Problem Solved
Shortest path tree	Dijkstra [30]	Centralized	Source	Source	$O(E \log V)$	Tree constrained
Minimum spanning tree	Prim [82]	Centralized	Source	Source	$O(E \log V)$	Tree optimization
	Gallager [41]	Distributed	Receiver	Source	$ V \log_2 V ^{(1)}$	Tree optimization
Steiner tree	Kou [69]	Centralized	Source	Source	$O(M V ^2)$	Tree optimization
	Takahashi [96]	Centralized	Source	Source	$O(M V ^2)$	Tree optimization
	Bauer [15]	Distributed	Receiver	Source	$O(D \cdot M)^{(2)}$	Tree optimization
	Maxemchuk [72]	Centralized	Source	Source	$O(M V ^2)$	Tree optimization
Constrained Steiner tree	Zhu [114]	Centralized	Source	Source	$O(k V ^3 \log V)$	Delay constrained tree optimization
	Kompella [66]	Centralized	Source	Source	$O(V ^3 \Delta)^{(3)}$	Delay constrained tree optimization
	Haberman [43]	Centralized	Source	Source	$O(k M V ^4)^{(4)}$	Delay/delay jitter constrained tree optimization
	Kompella [67]	Distributed	Source	Source	$O(V ^3)^{(5)}$	Delay constrained tree optimization
	Jia [47]	Distributed	Source Receiver	Source	$O(2 \cdot M)^{(6)}$	Delay constrained tree optimization
	Bauer [13]	Centralized	Source	Source	$O(M V ^2)$	Delay constrained tree optimization
Maximum bandwidth tree	Shacham [91]	Centralized	Source	Source	$O(E \log V)$	Link optimization
Misc.	Rouskas [87]	Centralized	Source	Source	$O(k M V ^4)^{(7)}$	Delay/delay jitter constrained tree optimization
	Chen [22]	Distributed	Source Receiver	Source	$O(M E)^{(8)}$	Bandwidth/delay constrained

(1) Message complexity: $O(|V| \log_2 |V| + |E|)$.

(2) D is the one-way trip time over the longest path between two nodes in the network or the diameter of the network. Message complexity: $O(M|V|)$.

(3) Δ is the delay requirement. The time complexity is polynomial if Δ is a bounded integer.

(4) k is the number of paths in the initial least cost path tree; l is the number paths tried when adding a path to a multicast group member.

(5) Message complexity: $O(|V|^3)$.

(6) Message complexity: $O(2 \cdot |M|)$.

(7) k and l are constants in the algorithm. A larger k or l results in a higher probability of finding a feasible tree and a higher overhead.

(8) Message complexity: $O(E)$.

Table 6.1: A taxonomy of multicast routing algorithms.

We start out with the review of some multicast routing algorithms which is relevant to the work in our dissertation, as follows. Some of the survey material was previously described in detail in chapter 2.

Shortest Path Tree

A shortest path algorithm minimizes the sum of the weights on the links along each individual path from the source to a receiver in the multicast group. If the unit weight is used, the resulting tree is a least hop tree. If the weight represents the link delay, then the resulting tree is a least delay tree. Bellman-Ford algorithm and Dijkstra algorithm [24] are the two well known shortest path algorithms, both of which are exact and run in polynomial time. Shortest path algorithms can be used to solve tree constrained (e.g. delay constrained) problems.

Minimum Spanning Tree

A minimum spanning tree is a tree that spans all the group members and minimizes the total weight of the tree. The well-known centralized minimum spanning tree algorithm is Prim's algorithm [24], and a distributed version was proposed by Gallager et al. [41]. In Prim's algorithm, the tree construction starts from an arbitrary root node and grows until the tree spans all the nodes in the network. In each step, a least cost edge connecting an off-tree node to the partial tree is added to the tree. The algorithm is greedy since the tree is augmented with an edge that contributes the minimum amount possible to the tree's total cost. Minimum spanning tree algorithms can be used to solve tree optimization problems.

Steiner Tree

The Steiner tree based problem aims to minimize the total cost of a multicast tree, and is known to be NP-complete [44, 112]. If the multicast group includes all nodes in the network, the Steiner tree problem reduces to the minimum spanning tree problem. Unconstrained Steiner tree algorithms can be used to solve tree optimization problems. However, they do not attempt to fulfill the tree constraints on an end-to-end basis, hence may not be well-suited for applications with such requirements. We summarize the KMB algorithms to the Steiner tree problem, next.

The KMB heuristic was proposed by Kou, Markowsky, and Berman. KMB applies Prim's minimum spanning tree algorithm to the complete distance graph, where the complete distance graph is a graph that contains all the nodes in the network and

has an edge between every pair of nodes representing the shortest path between them. The heuristic works as follows: (1) it creates a complete distance graph H from the original network topology G ; (2) it finds the minimum spanning tree U for the graph H ; (3) it builds a connected subgraph V by converting every node of U into its equivalent shortest path; (4) it applies the minimum spanning tree algorithm to subgraph V to create a spanning tree T ; and (5) it prunes T of non-multicast leaves until no non-multicast leaves remain. It has been shown in [69] that the KMB heuristic finds a tree whose cost is within twice the cost of the corresponding Steiner tree. In addition to KMB, Takahashi et al. [96] proposed a heuristic that constructs a tree whose cost is also within twice that of the Steiner tree, and Bauer and Verma [15] proposed a distributed algorithm for solving the Steiner tree problem.

Constrained Steiner Tree

The Steiner tree problem has been extended to include other side constraints, for example, delay, delay jitter, or a combination thereof. These problems are also NP-complete, and heuristic algorithms are sought for. Previous researches on some heuristic algorithm in the context of constrained Steiner Tree are presented as follow:

Zhu's algorithm: Zhu *et al* [114] proposed a heuristic algorithm, called the *bounded shortest multicast algorithm* (BSMA), to solve the delay constrained tree optimization problem. They defined the link cost as a function of the link utilization. They also defined a superedge of a tree as the longest simple path whose internal nodes (i.e., excluding the end nodes on the path) are relay nodes and each relay node connects exactly two tree edges. The algorithm starts by computing a least delay tree rooted at a given source and spanning all the group members. It then iteratively replaces superedges in the tree with cheaper superedges not in the tree while not violating the delay constraint, until the total cost of the tree cannot be further reduced. Cheaper superedges are located by using a k -th shortest path algorithm. BSMA always finds a delay constrained multicast tree if one exists because it starts with a least delay spanning tree.

Kompella's centralized algorithm: Kompella *et al.* [66] proposed a heuristic algorithm, called the *KPP heuristic*. They assumed that the link delay, $d(u, v)$, of link (u, v) and the delay constraint D are integers, while the link cost, $C(u, v)$, of link (u, v) may take any positive real value. They defined (i) a *constrained cheapest path* between two nodes u and v as the least cost path from node u to node v that has delay less than D (the cost and delay on such a path are denoted as $P_c(u, v)$ and $P_d(u, v)$, respectively); and (ii) a *closure graph* G' of a graph $G = (V, E)$ as a complete graph over the nodes in V , with edges representing constrained cheapest paths.

Given the source s and a multicast group M , KPP first computes a delay constrained closure graph G' over $\{s\} \cup M$ using dynamic programming. The constrained cheapest path from node u to node v is then located. Then, KPP uses Prim's algorithm [24] to obtain a minimum spanning tree of the closure graph G' . Starting with the source node, the tree is incrementally expanded by adding an edge one at a time until all the receiver nodes are included. The edge selected each time is the one which (1) connects an on-tree node and an off-tree node, (2) does not violate the delay constraint, and (3) minimizes a selection function. KPP proposed two selection functions: one is the link cost, and the other strikes a balance between cost minimization and delay minimization. Finally, KPP replaces the edges in the minimum spanning tree with paths in the original graph G . Loops if any are removed.

Haberman's algorithm: Haberman *et al.* [43] considered the Steiner tree problem under the delay and delay jitter constraints. The algorithm first constructs a *reference tree*, T_R , of least cost paths from the source node s to all receiver nodes. Second, for each receiver node $d_i \in M$, the algorithm attempts to construct a tree, T_i , that initially contains the path in T_R from node s to node d_i . Then, the algorithm augments T_i by adding "good" paths from on-tree nodes to off-tree receiver nodes, until all the receiver nodes are included. If more than one feasible tree is eventually constructed, the one with the least cost is selected.

Kompella's distributed algorithm: Kompella *et al.* [67] proposed a distributed heuristic algorithm to construct delay constrained Steiner trees. The algorithm requires that every node maintain a distance vector of the minimum delay to every other node in the

network. It starts with a tree that initially contains the source node, and augments the tree by adding receivers one at a time, until all the receivers are included in the tree. The approach used to select a receiver for inclusion is as follows: the source node s multicasts a find message via the partial tree. Upon receipt of a find message, a node locates an outgoing link that connects to an off-tree receiver, does not violate the delay constraint, and minimizes a selection function. The node then sends back to the source a response message that contains the identify of the candidate link. Upon receipt of all the responses, node s decides the best link l to be added to the tree. The algorithm requires multiple passes of control messages.

Jia's distributed algorithm: Jia [47] presented another distributed algorithm to solve the delay constrained tree optimization problem. It is assumed (perhaps unrealistically) that the least cost path between two nodes is always the shortest delay path between them.

To facilitate distributed implementation, a table is used to keep track of the following information for each receiver d : (i) whether or not node d is currently on tree, (ii) the on-tree node, n , at which which a tree branch should be grafted to connect to node d ; the tree branch is a shortest path P from node n to node d that incurs the least cost and fulfills the delay constraint, and (iii) the least cost incurred in P .

The algorithm starts with a tree that contains only the source node. For each receiver node d , the source node s constructs a least delay path P from itself to node d . If such a least delay path P satisfies the delay constraint, node s records in the table itself as the on-tree node at which a tree branch (i.e., P) is grafted to connect to node d . Node s then selects a receiver whose least delay path incurs the least cost (let the receiver be denoted receiver j), composes a setup message that carries this table and the cumulative delay from node s , D (which is initialized to 0), and sends the message to receiver j . The message is sent, hop by hop, along the shortest path to receiver j .

Upon receipt of a setup message, an intermediate node u updates and remembers the parameter D . In addition, node u checks for each currently off-tree receiver, i , if a constraint-satisfying path with a smaller cost (than the one currently recorded in the table) exists. If so, for each such receiver i , node u updates the table to reflect that a tree branch to receiver i should be grafted from node u . After the setup

message reaches receiver j , receiver j selects the next receiver to join the multicast tree as the one whose least-delay path to an on-tree node incurs the least cost.

Bauer's algorithm: By imposing constraints on the number of outgoing links that can be used for a multicast group (which was termed as the *copying ability*) at each individual node, Bauer and Varma [13] proposed a node degree constrained Steiner tree algorithm. They proposed to modify six existing unconstrained Steiner tree heuristics. All the heuristics have a common property: a multicast tree is constructed by connecting different components. Each heuristic merges two components of a graph by the shortest path between two components. In the degree constrained case, one or more such shortest paths may exhaust the allowable degree of a node. Thus, the heuristics are modified as follows: when a node's degree constraint is violated by a partial tree, the node and its remaining edges are eliminated from further consideration. This modified topology may alter the shortest path information for the remaining algorithm steps. As a consequence, modified heuristics must re-evaluate the shortest paths between nodes when nodes and/or edges are eliminated. The authors also proposed an alternative heuristic, called *shortest path heuristic with iteration* (SPH-R). Construction of a tree begins with an arbitrary starting point, and an edge that is closest to the partial tree is added, one at a time. The shortest path heuristic is repeatedly applied to the network graph for different starting points. SPH-R terminates when it generates a solution.

Maximum Bandwidth Tree

Shacham [90] proposed a maximum bandwidth tree algorithm for distributing hierarchically-encoded data. It uses a Dijkstra-like algorithm to compute the maximum single-path bandwidth to all destinations. Their algorithm works as follows. First, they compute the maximum available bandwidth paths to all receivers from the source. The set of links connecting the nodes on the paths to the receivers form a maximum bandwidth tree by construction. Second, receivers are classified into different categories according to their receiving capabilities. A quality value is assigned for each layer of data. The satisfactory level of a receiver is measured by summing up the quality value over all intended layers that are received. The rate at which each receiver will

receive is then determined to maximize the sum of the satisfactory level of all receivers. This optimization procedure gives for each individual receiver the intended rate at which it will receive from the source. The link bandwidth will then be allocated appropriately on the maximum bandwidth tree. The maximum bandwidth tree algorithm solves the link optimization problem.

Miscellaneous Trees

Rouskas *et al.* [87] studied and proposed a heuristic algorithm to the problem of constructing source-based multicast trees to meet the delay and inter-receiver delay jitter constraints. Chen *et al.* [22] proposed a distributed receiver initiated probe-based multicast routing algorithm to construct a multicast routing tree with certain QoS requirements.

Tree Rearrangement in Response to Member Join/Leave

A multicast group member may join or leave a multicast session dynamically. It is thus important to ensure that member join/leave will not disrupt the on-going multicast session, and the multicast tree after member join/leave still remains near-optimal and/or satisfies the QoS requirements of all on-tree receivers. If a multicast tree is reconstructed each time a member joins or leaves, on-tree nodes may not switch to the new tree simultaneously and a seamless transition may not be possible. One may handle dynamic member join/leave by incrementally changing the multicast tree. When a new member intends to join the distribution tree, a tree branch connects the new member to the nearest tree node. When a member leaves the multicast group, only the corresponding tree branch is torn down. This incremental change approach suffers from that the quality of the multicast tree maintained may deteriorate over time in terms of, for example, the total tree cost.

Several researchers addressed the multicast tree rearrangement issue, among which the *edge-bounded algorithm* (EBA) [45], Bauer and Varma's algorithm [11], Narvaez's algorithm [74], and Sriram's algorithm [94] may have received the most attention. The main idea is to define and monitor certain damage index to the multicast

tree as members join/leave, and trigger tree rearrangement when the index exceeds certain threshold.

6.3 Real-time Applications

Real-time applications impose stringent delay and throughput constraints on the network, as compared with traditional data applications. When real-time applications communicate across a network, data must traverse the network in time for the application to use it. Many in the research community have investigated solutions to the real-time multicast routing within the context of QoS routing. A typical QoS routing scheme globally distributes topology, link resource availability, group membership and per-flow resource usage. A source's first-hop router then uses this information to compute a multicast tree that is known *a priori* to have available resources. Because these QoS routing approaches require global distribution and synchronization of such rapidly varying quantities, we do not believe they are applicable to interdomain routing, where issues of scale are paramount. A global database of topology alone scales linearly with the size of the network; neither group membership nor the number of flows is limited by the size of the network.

6.3.1 Real-time Multicast routing Model

The real-time multicast routing is defined as the following:

Given a network graph $G(V, E)$, a source node $s \in V$, a set of destination nodes $D \subseteq V$, and a real-time constraint Δ , a real-time routing tree for multicast connections is a subtree of the graph $G(V, E)$ rooted from s , that contains all of the nodes of D and an arbitrary subset of $(V - D)$, whose leaf set consists only of a subset of nodes of D , and the delay from s to any node in D is within the time constraint Δ . However, the multicast trees obtained from such conditions may have a high cost.

So, the optimal real-time multicast was proposed. The optimal real-time multicast satisfies (1) the overall network cost is reduced; and (2) the maximum delay from the source to any destinations is within a specified real-time constraint Δ . The network cost of multicasting a message to a group of destinations is proportional to the sum of the cost of all links in the tree. We define the network cost of a multicast routing tree as:

$$\text{Network Cost } (T) = \sum_{l \in T} c(l). \quad (6.1)$$

The bounded delay requirement can be expressed as the following, where $P(s, u)$, $u \in D$, is the path from s to u along the routing tree D and Δ is the delay bound.

$$\text{Network Delay } (T) = \sum_{l \in P(s, u)} d(l) \leq \Delta \quad (6.2)$$

The three major difficulties in routing for real-time multicast connections are: fully distributed routing, delay-bounded suboptimal routing, and integration of routing with connection configurations. We now discuss some existing solutions to those problems.

Distributed Steiner Tree Heuristics: Some widely adopted distributed heuristics are based on minimum spanning tree (MST) heuristics. An MST heuristic is to generate an MST of the network graph $G(V, E)$, spanning all nodes in V . Then an approximate Steiner tree is obtained by removing, from the MST, subtrees containing no nodes in $\{s\} \cup D$. Basically, there are two types of distributed MST algorithms [47]. One type is based on the Prim's MST algorithm. Prim's algorithm [24] initializes the tree as the source node and then grows the tree by successively adding the next closest node to the tree, until all nodes are in the tree. The other type is based on Kruskal's MST algorithm [24]. Kruskal's algorithm initializes each of the nodes as a subtree and joins subtrees pairwise repeatedly until all the nodes are in a single tree. There are two disadvantages to these algorithms. First, all the nodes in the network are involved in the execution of the MST algorithms, which is very costly in large networks. Second, it takes two steps to produce a routing tree; 1) generate an MST of the whole network and 2) prune the MST to the routing tree. It costs more network messages and takes a longer time to establish a connection.

Delay-Bounded Suboptimal Routing: The cost requirement often conflicts with the delay requirement in multicast routing. A Steiner tree with optimal network cost may have a long delay to the farthest destination. Whereas a shortest path tree (SPT), in which each path from the source to a destination is a shortest path (in terms of delay), has the

shortest delay, it may incur a high network cost. A tradeoff algorithm between optimal network cost and minimum average delay was proposed by Kumar *et al* [10]. It generates two routing trees, an SPT T and a Steiner tree T' . A k -degree tradeoff (k is a positive integer specified by the user) is achieved by first identifying out k destinations to whom the difference between the delay in T and the delay in T' is the largest. Then the paths to k destinations in T' are replaced by the corresponding shortest paths in T , obtaining a less optimal routing T' , but with a shorter average delay. This idea can be extended to delay bounded routing by replacing those paths in the Steiner tree whose delay exceeds the bound by their shortest paths. In multimedia applications, there is actually no need to minimize the average delay to all destinations. It often requires that the delay to any destinations shall be within a bound. A recent survey of delay-bounded routing algorithms and the evaluations of them can be found in [88]. Kompella *et al.* proposed two *centralized* heuristics in [66]: the *cost-delay* heuristic and the *cost* heuristic. Both of them are based on Prim's MST algorithm. A routing tree grows up from the source s . Each time when selecting the next nontree node v to add to the tree, the *cost-delay* heuristic uses the following function to convert the cost and delay of a link into the weight

$$\text{If } (D[u] + t[u, v]) < \Delta$$

$$\text{Then } w[u, v] = c[u, v] / (\Delta - (D[u] + t[u, v]))$$

Otherwise

$$w[u, v] = \infty$$

Where c and t are the cost matrix and delay matrix, respectively; u is a tree node and $D[u]$ is the delay from s to u along the tree. Then it selects node v which has the smallest $w[u, v]$ and adds it to the tree. The *cost* heuristic simply selects node v whose cost to the tree is minimal under the condition $D[v] \leq \Delta$ and adds it to the tree.

These two heuristics are later extended to distributed versions in [48]. In their distributed versions, the two heuristics of selecting edges remain the same. But each time of selecting an edge, adding to the tree, it takes one round trip along the tree

formed so far: a *FIND* message is first sent from s down to the tree leaves, node by node, to determine the best out-going edge at each node; then the best outgoing edge is propagated from the leaves back to s , replaced by a better choice along the way. This is a time-consuming and message-costly procedure.

Jia *et al.* [50] proposed a *centralized* algorithm which improves the performance of Kompella's cost heuristic. When selecting node v to add to the tree, if $D[v] > \Delta$, it backtracks the tree formed so far to find a structure of the tree which can link v to the tree with the least cost under the condition $D[v] \leq \Delta$. It then restructures the tree to include v into the tree. Shu *et al.* proposed another *centralized* algorithm called the bounded shortest multicast algorithm (BSMA) in [114]. BSMA starts with an SPT to all destinations. It then iteratively replaces super-edges in the tree with lower cost paths not in the tree without violating the delay bound. A super-edge is a path in the tree between two branching nodes, two destinations, or a branching node and a destination. The operation continues until the total cost of the tree cannot be reduced any further. The computation cost of searching the best replacement of a super-edge under the delay constraint is very high. The computing complexity is $O(Kn^3 \log n)$, where K is the convergence number in searching a delay-bounded shortest path (shortest in terms of cost) for a super-edge, and n is $|V|$.

Integration of Routing with connection configuration: In a distributed environment, routing operation must be integrated with the operation of connection configuration so that a connection can be established faster. This is because connection configuration needs to be done node by node along the direction from the root to all leaf nodes. If the routing operation is separate from the root to all leaf nodes. If the routing operation is separate from the configuration, it would take an extra traverse of the whole tree to configure the connection, which is costly in both time and network messages. Centralized algorithms compute routing trees at a central node, so they need another phase for connection configuration. Some distributed heuristics, such as those based on Kruskal's MST algorithm [14, 41], construct routing trees in parallel. The input output directions at a tree node are not known during the tree construction until the whole tree

is generated. Therefore, they also need a separate phase to configure the connection after the routing tree is generated.

6.4 Real-time optimal multicast routing

We adopted the two path selection methods from chapter 4. and applied the same modification to the CBT v2, using the modified Dijkstra's shortest path algorithms to obtain the optimum paths that satisfied the real-time delay constraints with reduced cost. We later enhance the real-time optimal multicast routing algorithm with the fault-tolerant protocol which we discuss in section 6.5. Figure 6.1 shows the flow chart of our proposed protocol with Weighted Dijkstra's path selection method and Residual delay path selection function. The flow chart looks similar to the one in chapter 4 (figure 4.1.)



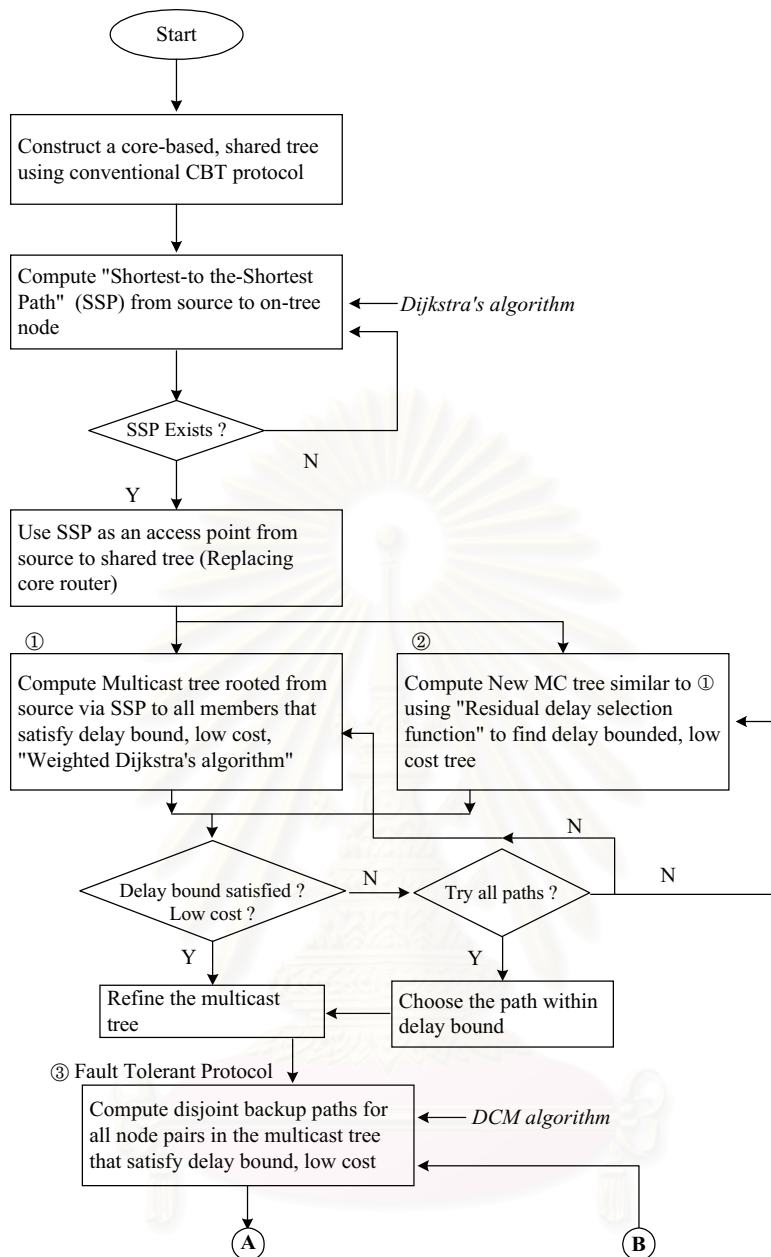


Figure 6.1: Flow chart of our proposed real-time multicast protocol

Figure 6.1 illustrates the real-time optimal protocol part of our proposed approaches which is similar to the one presented in chapter 4. We then extend our approach to immune our proposed protocol with the fault-tolerance capability which is shown in figure 6.2

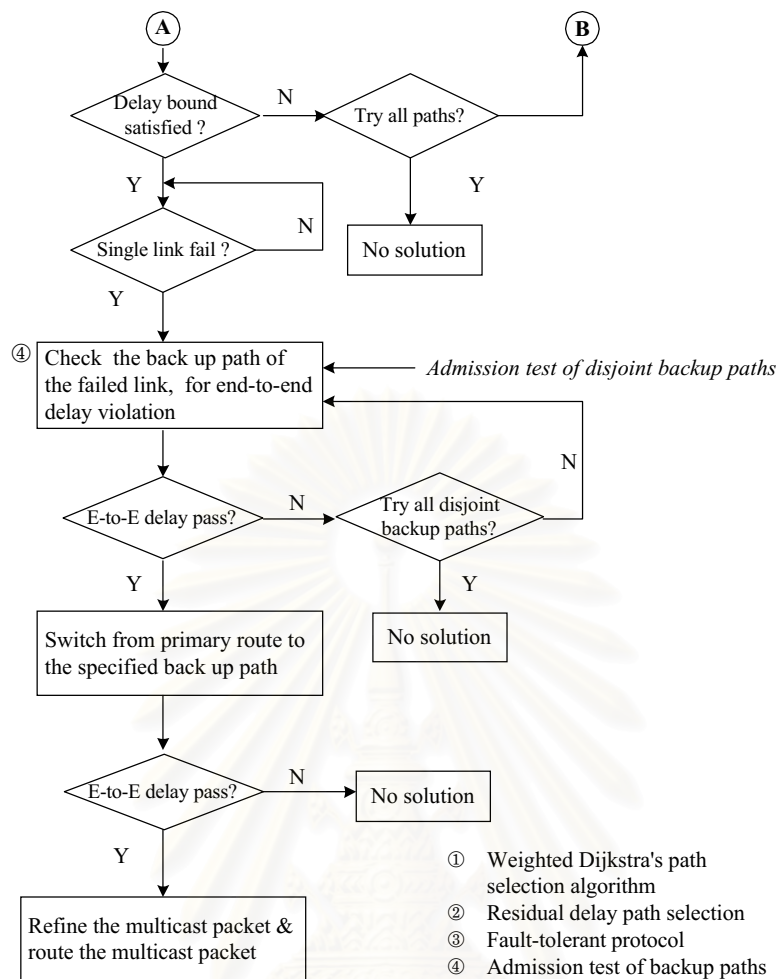


Figure 6.2: Flow chart of our proposed fault-tolerance protocol

6.5 Fault-tolerant Multicast Protocol

For reliable multicasting, some mechanism is required for a network to recover from faults. If one of the group members fails or one of the nodes or links used in the multicast tree fails, the multicast tree can become disconnected. It is then impossible for messages from the core to reach the members of the disconnected subtree.

The main schemes to provide fault tolerance in routing are dispersity routing or multipath routing. While the idea of dispersity routing has been around for many years, the traditional uses of dispersity routing have been at the physical layer of network communication. Many dispersity schemes were proposed for fault-tolerant real-time channels in [3, 21]. Some others use dispersity schemes to provide multiple paths between a pair of source and destination nodes; one of the paths is used as the primary

path under normal conditions, and others are used as backup paths if the primary path fails [3, 21]. Dispersity routing is also used to split the traffic into multiple paths, in order to increase network throughput and to decrease message delay [3, 21].

6.5.1 Backup Paths with real-time constraints

The main schemes to provide fault tolerance in routing are dispersity routing or multipath routing. One important, yet under explored, problem is the fault tolerance associated with the real-time constraint of meeting end-to-end delay guarantee. As mentioned earlier that, we can enhance the multicast routing protocol with fault tolerance capability by finding the node-disjoint paths between each pair of nodes in the network. However, finding two node-disjoint paths between any pair of nodes were proposed in [40]. Given a graph $G = (V, E)$ and a set of k pairs of vertices in V , we are interested in finding for each pair, a path connecting each node pair such that the set of k paths so found is edge disjoint. For arbitrary graphs the problem is NP-complete, although the problem can be solved in polynomial time if k is fixed [40].

In our formulation, we are interested in finding a backup path for every node pair that satisfies the end-to-end delay constraint. First, we assume that at least one disjoint backup path exists between any two nodes (except the core node) in a shared tree. This problem has been studied in the context of *dependable network* G whereby, G is dependable if at least two disjoint routing paths between any nodes in G exist. In other words, a dependable network is 2-edge connected according to the Menger's theorem that the removal of any one edge does not disconnect G [73]. This condition is very important since if a noncore node does not have such disjoint backup path, then the network is not single-fault tolerable. If multiple backup paths exist, we select the one with the shortest distance (less hop counts).

In our future work, in order to tolerate multiple simultaneous faults, an extension to backup paths selection is needed to select multiple disjointed backup paths (e.g. 2 backup paths for each node pair). These paths connect the router to its grandfather. However, this is not sufficient and requires additional backup paths to connect not only to its grandfather but also other ancestors. For multiple faults tolerance, this will be explored in our future work. Then we run the backup paths through the admission tests

by using the Dynamic Connection Management (DCM) Routing algorithm proposed by Parris and Ferrari in [80] to verify if the backup paths fulfill the end-to-end delay constraints. DCM utilizes a constrained, modified version of the Bellman-Ford algorithm to guarantee the time delay constraint conformance on the backup paths with minimum cost paths. That is, the path will have the lowest cost possible without violating the delay constraint. If the path fails the admission test, the algorithm will conduct search until the path under the delay constraint value is found, if such path exists. If such path does not exist, then it is not possible to form the multicast tree, resulting in loss traffic.

Since providing end-to-end delay guarantee involves firstly, fault detection, node rerouting and router configuration. Our problem in this case is to find the backup paths that guarantee not only the propagation delay from source to destination but the excess delay value from node rerouting and routing reconfiguration time mentioned earlier, has to be considered, as well. To define the excess delay value, E , it is associated with each edge (x,y) , in addition to cost C and delay Δ . The excess delay value is apportioned to the edge during the backup paths admission phase. Its value is calculated in a such a way that along any backup path from the source to a destination node, the sum of delays and excess delays is bounded by Δ ,

$$D(x, y) + E(x, y) \leq \Delta \quad (6.3)$$

The results will be the backup paths that immune the network from any single failure in the shared tree. We assume that the link between source and the designated on-tree node does not fail. It should be observed that if no fault is detected in the network, our proposed protocol will try to optimize the path with both delay and cost constraint by using the weighted Dijkstra's algorithm and residual path selection function. In case of a single fault detection, the algorithm will attempt to admit the pre-defined backup paths, hence, to verify that the time delay constraints condition of the multicast tree still holds.

6.6 Simulation results

In this section, we will describe the simulation results and discuss the performance of our protocol. We used the same network topologies, cost and delay functions and traffic

source which we used in the experiments of chapter 4. Therefore, a brief summary of the experimental setup suffices. We simulated a random network with an average node degree of 4, which is close to the Internet environment. We distributed n nodes randomly across a Cartesian coordinate grid of size 100 by 100. Nodes in the network graph represent the communication endpoints. The edges connecting the nodes represent links. Edges between nodes are added by considering the probability function

$$P(u,v) = \beta \exp (-d(u,v)/ L\alpha) \quad (6.4)$$

for all possible pair (u,v) of node, where $d(u,v)$ is the Euclidean distance between the nodes u and v , L is the maximum possible distance between any two nodes, and β , α are parameters in the range $(0,1)$. Large values of β increases the number of edges from each node while smaller values of α increases the number of connections to nearer nodes compared to nodes further away. In our simulation, we set β to 0.25 and α to 0.2 to simulate a large network such as the Internet. To simulate the propagation delay, we assigned to each edge a network delay equivalent to the Euclidean distance between the two nodes. In terms of network cost, we assign the cost value similar to that in our experiment in chapter 4.

Faults are also randomly generated with X being the average life-time of a single link fault, and Y being the average interarrival time of faults, as follows.

$$P_f = \Pr (\text{The system is in the fault state}) = X/Y \quad (6.5)$$

That is, P_f is the probability that the system is in a faulty state. The network performance will be measured as a function of P_f . We assume that a single link failure uses a fail-stop link failure model, resulting in a sudden stop of a functioning link, and therefore resulting in a single link failure. We assume that after a single link fault period is over, it is possible for another (single) link fault to start, but not at the same time. The same link can fail more than once but with average failure time of X . The case of multiple simultaneous link faults will be studied in our future work.

CBT version 2's specifications described in RFC 2189 is also simulated on ns-2 [75] as a baseline. We also modify the existing CBT version 2 with the feature described in section 6.4 and section 6.5. Backup paths were calculated off-line and each backup

path has to go through the admission tests at run time to verify the end-to-end delay acceptance before the switching of the paths take place.

At each simulation the multicast group size and the real-time constraint Δ vary. Each time the source node and the destination set are randomly selected from the network graph. We used a static multicast model in our experiment. The experiment was run repeatedly until the confidence interval for the number of all measured quantities are less than 5% using the 95% confidence level. On the average, 300 different networks were simulated in each experiment in order to reach such confidence levels. The following performance metrics are considered as follows:

- End-to-end delay bound: The delay of a packet is defined as the summation of the routing delay, transmission delay and queuing delay. Ratio of the average and maximum propagation delay in our proposed protocol to its counterpart in CBT v2. Also, the result should prove that our protocol can route the packet within time delay constraint (maximum end-to-end delay) of the real-time application under fault or no fault scenario.
- Network resource usage: Total number of hops a multicast packet travels to reach all destination in the multicast groups.
- Traffic concentration: Traffic concentration is measured by the maximum number of flows traversing a unidirectional link (the load of the most congested link.). Link distribution is also observed and compared with the original CBT v2. This also shows link utilization of our proposed protocol against the CBT v2.
- Loss Rate: The loss rate measures the fraction of the transmitted packets that are not delivered at all or are delivered so late as to be useless for real-time applications. The loss rate can be seen as the failure rate of our proposed protocol to construct the delay bound multicast tree.
- Execution time: The execution time measures the running time of our algorithm from start until the time the multicast tree is completely formed.

6.7 Performance evaluation

The simulated traffic is with burst size 1 MB, and average rate of 1 Mb/s. The maximum length of the packet was set to 1000 bytes. The link capacity of all the links in the network was randomly chosen from the set of {2, 4, 6, 8} Mb/s. This traffic is the same set that we used in chapter 3.

6.7.1 Average end-to-end delay

Average end-to-end delay is the average period for a data packet to be routed through the network from the application where it was created to a destination application. With real-time end-to-end delay constraint, the results should show that our protocol can route the application within such constraint, that is, within the maximum delay bound Δ . Figure 6.3 and figure 6.4 show the results of our proposed protocol to their counterparts in CBT v2 in terms of total network cost against maximum network end-to-end delay with multicast group size of 50 and 100. They reveal that our proposed protocol has better delay performance than that of the CBT and yet sustaining better cost and delay performance than the original CBT in both cases. Figure 6.3 shows that at a tight delay constraint value, our modified CBT_{RD} can produce a multicast tree three times less cost than that of the CBT. Figure 6.5 and figure 6.6 show the histograms of the ratio of the average end-to-end delay in CBT and modified CBT_{WD} and modified CBT_{RD} respectively. It reveals that the both modified CBT_{WD} and modified CBT_{RD} have better delay performance than that of the CBT. On average, modified CBT_{RD} performs best in terms of utilizing a shorter delay paths, while maintaining lower cost tree.

We conducted another experiment to verify the effect of a single link failure in our random network. The fault probability was set to 0.1 and 0.5 respectively while similar parameters in previous experiment were applied.

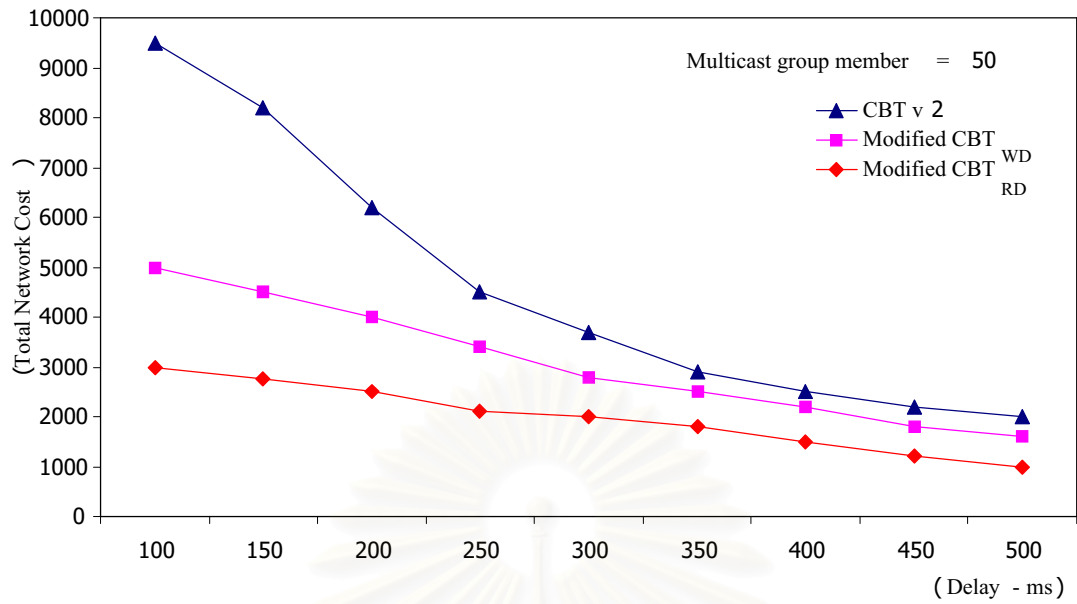


Figure 6.3: Total cost of a multicast tree, multicast group member = 50

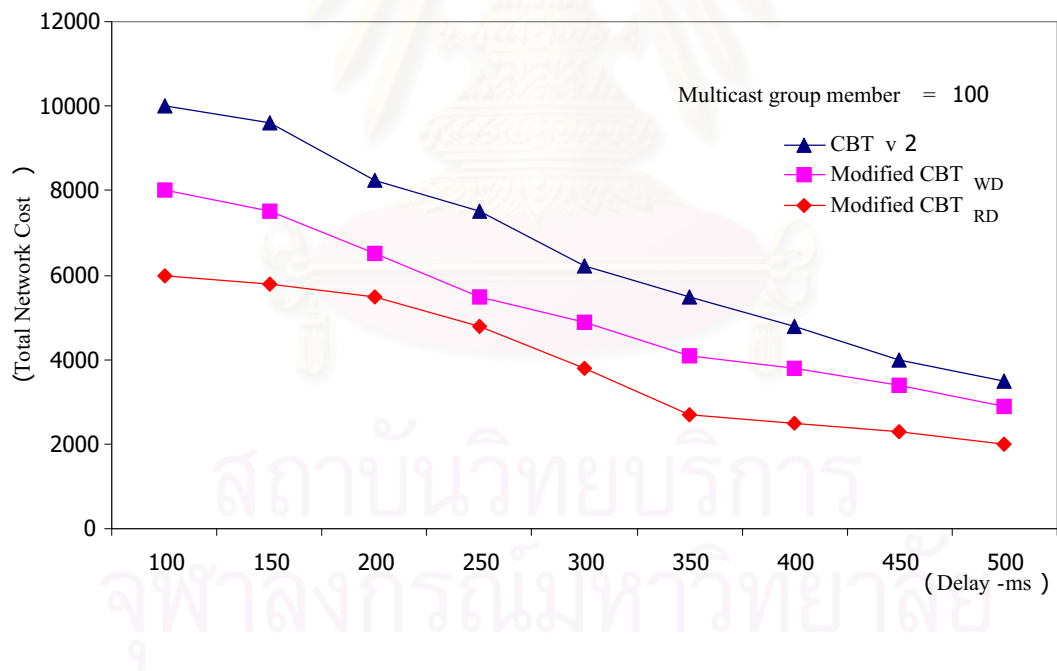


Figure 6.4: Total cost of a multicast tree, multicast group member = 100

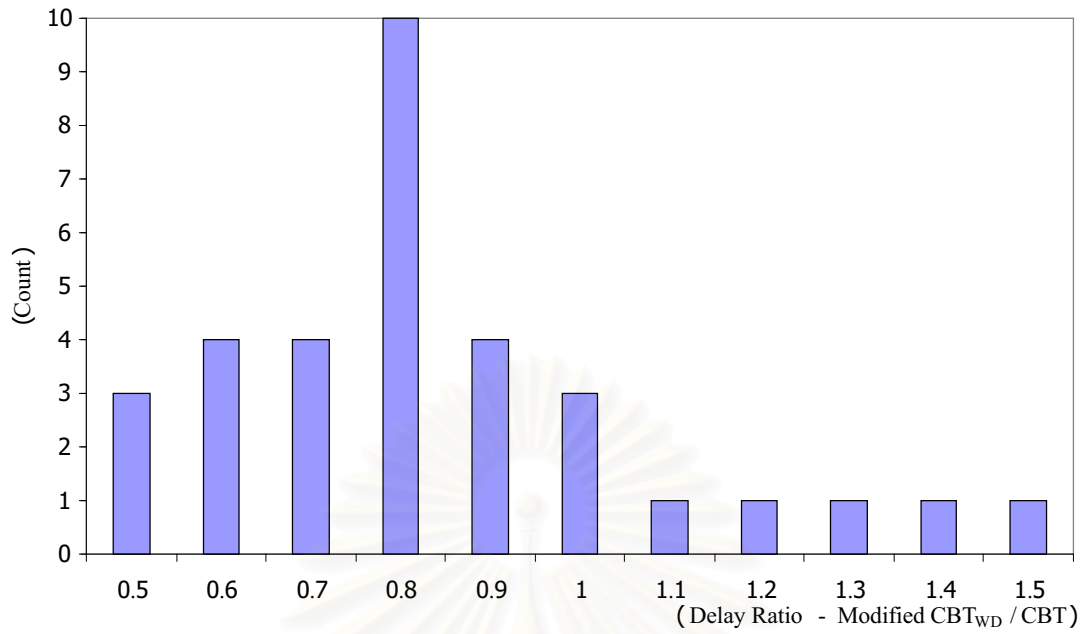


Figure 6.5: A Histogram of the average end-to-end delay in modified CBT_{WD} compared to its counterpart in CBT.

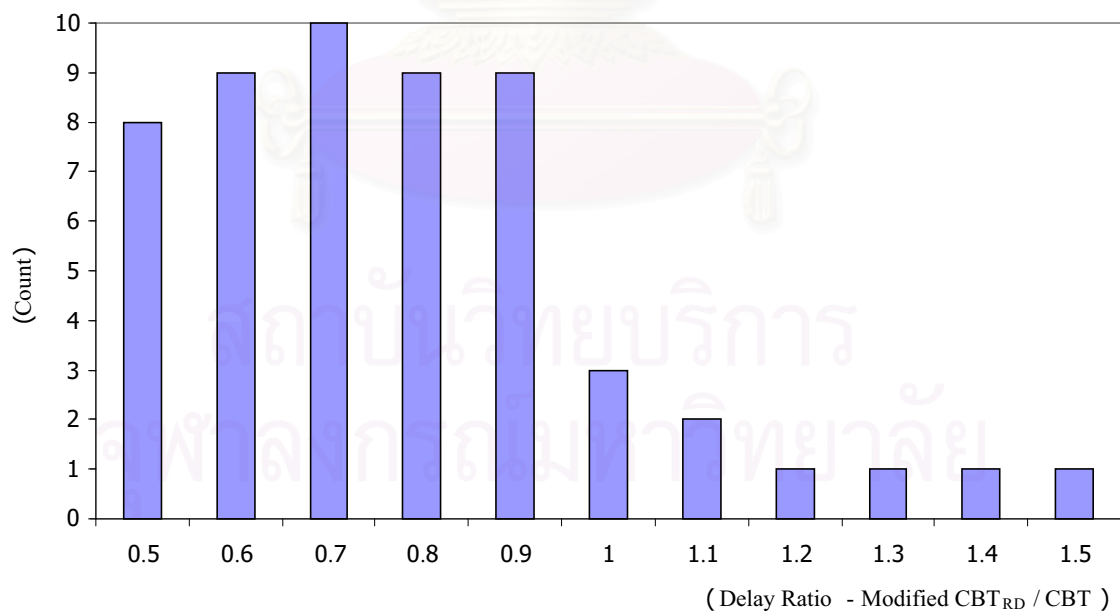


Figure 6.6: A Histogram of the average end-to-end delay in modified CBT_{RD} compared to its counterpart in CBT.

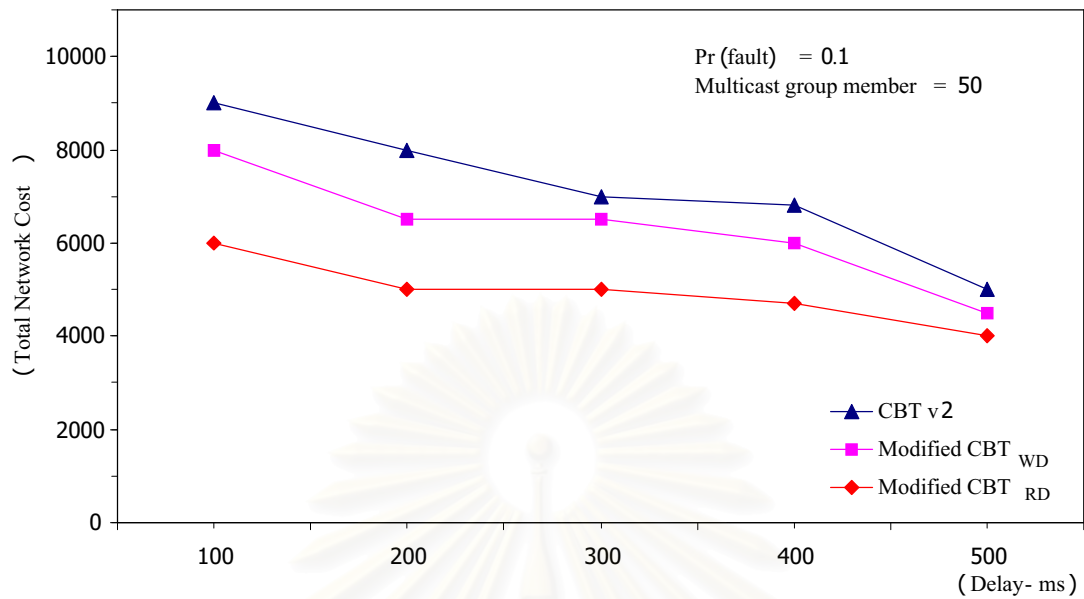


Figure 6.7: Total cost of a multicast tree, Pr (fault) = 0.1

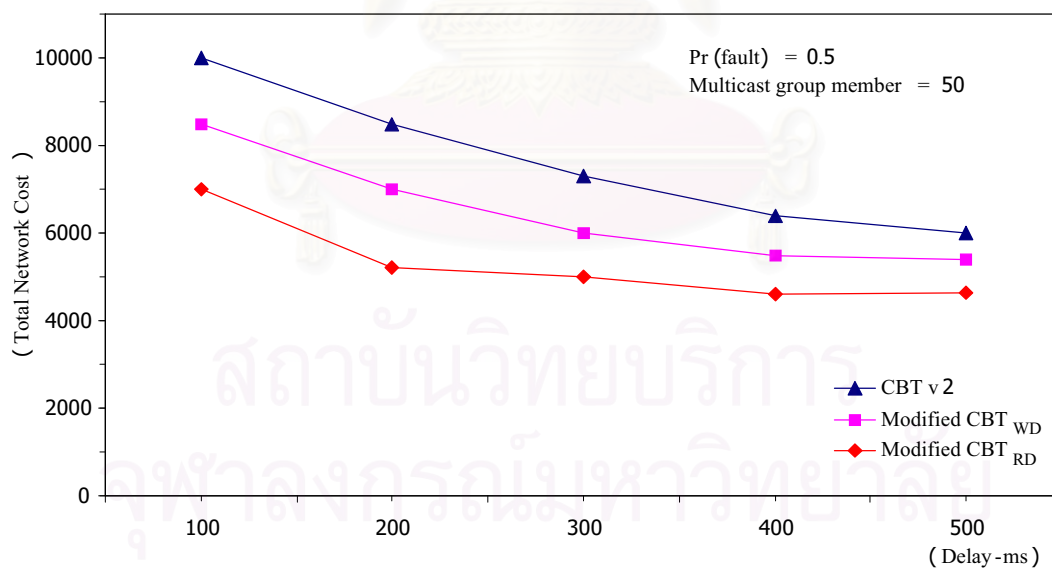


Figure 6.8: Total cost of a multicast tree, Pr (fault) = 0.5

Figure 6.7 and figure 6.8 show that as the delay constraint is relaxed, the total cost of network decreases which is consistent with the previous experiment when there is no

fault. When the delay constraint is tight, it limits the algorithms' ability to construct cheap trees, because there are not many possible solutions for the problem. As the delay constraint value increases, its effect on restricting the algorithms' efficiency in constructing cheap tree diminishes, and the algorithms are capable of constructing cheaper trees.

We also found that with higher fault probability, the total network cost also increases. It is obvious that our modified protocol incurs up to 30% less costly tree than the CBT protocol. Since our algorithm uses weighted Dijkstra's algorithm and residual delay path function to optimize the multicast trees, as we expected, it is obvious that both of our proposed protocols obtain better cost tree than that of the CBT under fault condition.

We repeated the experiment with a variable fault probability with multicast group size fixed at 50 and then 100. In the next experiment we compare total network cost of CBT v2, modified CBT with Jia's algorithm [47], which is also suitable for delay-sensitive applications. Jia's algorithm is based on distributed algorithm which is suitable for large network. The algorithm's performance is stable and it generates good quality of routing trees with low network cost. Jia's algorithm constructs multicast routing trees using Prim's Minimum Spanning Tree algorithm in combination with a distributed shortest path algorithm [24].

We simulated variable fault probability values but with fixed multicast group size at 50 and fixed delay constraint value of 100 ms. The results are shown in figure 6.9 and figure 6.10 for different group sizes. We show the percentage excess cost of each algorithm relative to Jia's algorithm. The percentage excess cost of a tree generated by our proposed protocol relative to Jia's algorithm is calculated as follows:

$$\frac{\text{Cost of tree under modified CBT} - \text{Cost of tree under Jia's algorithm}}{\text{Cost of tree under Jia's algorithm}} \times 100 \quad (6.6)$$

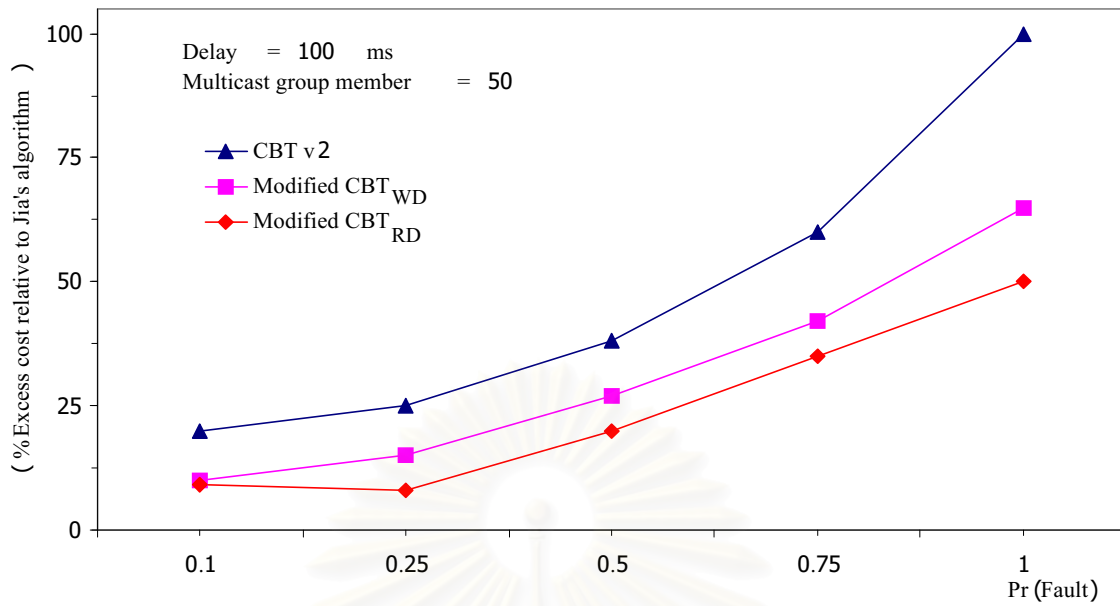


Figure 6.9: Total cost of a multicast tree relative to Jia's algorithm, multicast group size = 50, Delay constraint value = 100 ms

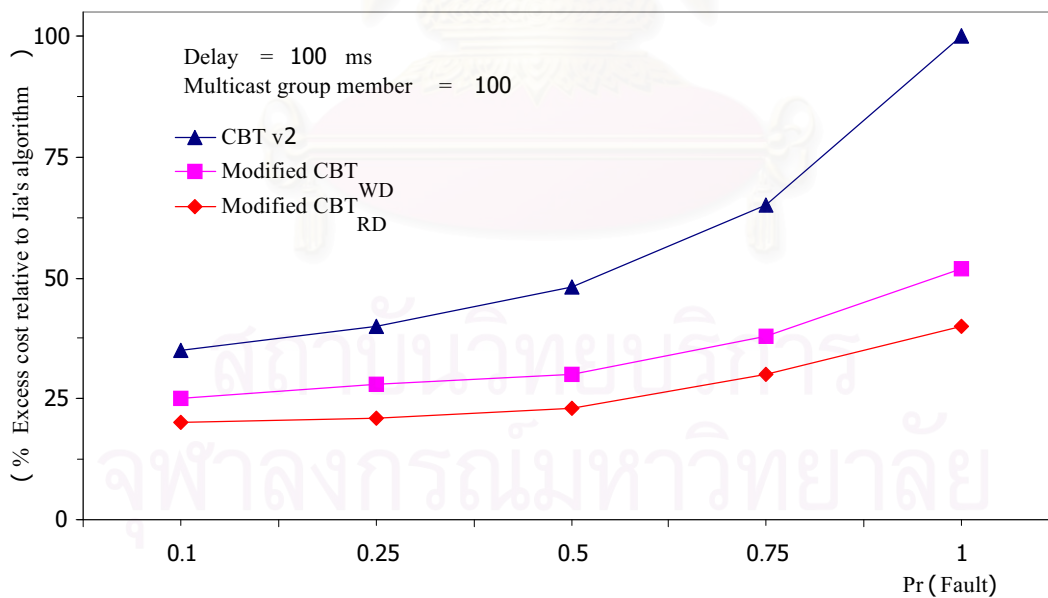


Figure 6.10: Total cost of a multicast tree relative to Jia's algorithm, multicast group size = 100, Delay constraint value = 100 ms

With higher fault probability, the cost of network increases, but the rate of increase of our proposed protocol is less compared with that of the CBT. We can see that with fault probability of less than 0.5, the cost of our modified CBT_{RD} protocol is around 25% worse than Jia's algorithm, as shown in figure 6.10. The experiment was repeated with different multicast group sizes and different delay constraint values. The results are shown in figure 6.11 and figure 6.12. Figure 6.11 illustrates the cost effect of our proposed protocols compared with Jia's algorithm in terms of different delay constraint values of the modified CBT_{WD} while figure 6.12 demonstrates the cost effect on the modified CBT_{RD} .

We measured the relative cost of network compared with Jia's algorithm and found that it is consistent with the previous experiment where smaller group size of 20, modified CBT_{RD} produces within 25% higher cost tree compared with Jia's algorithm. Our proposed protocol performs worse as the multicast group size increases.

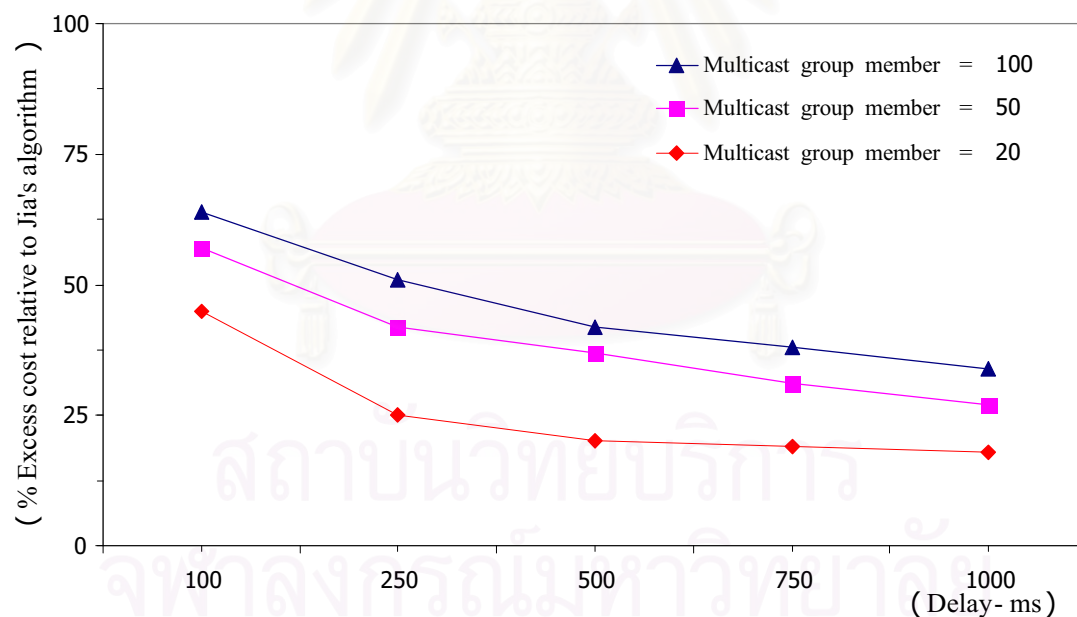


Figure 6.11: Total cost of a modified CBT_{WD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, at $Pr(\text{fault}) = 0.1$

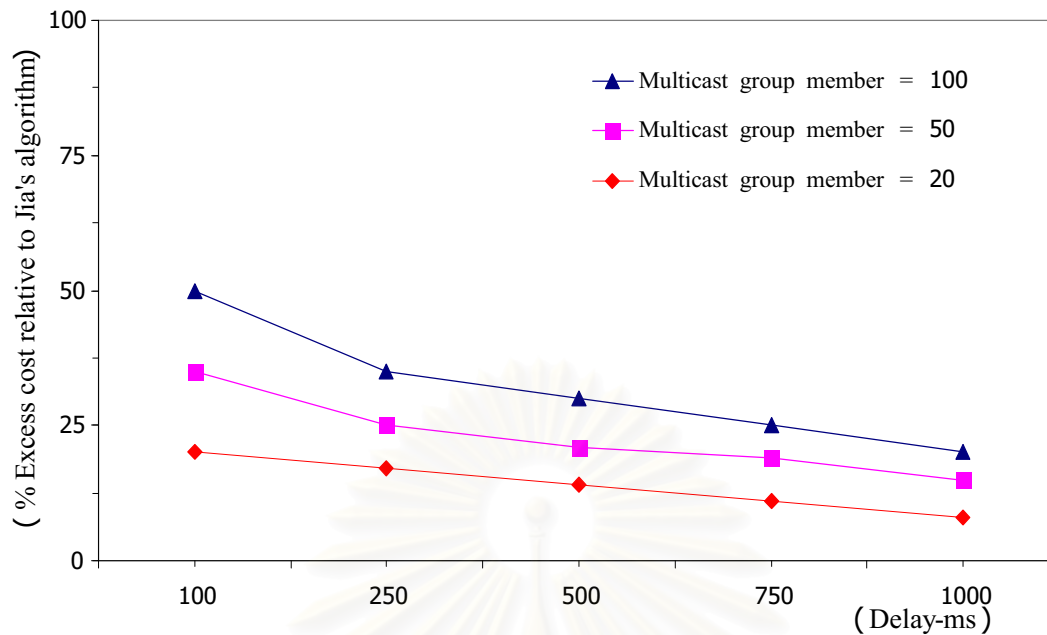


Figure 6.12: Total cost of a modified CBT_{RD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, at $Pr(\text{fault}) = 0.1$

6.7.2 Network resource usage

A simple method to route a packet to all interested receivers is to unicast the packet to each receiver. However, unicasting is likely to route several copies of the same packet over links in the network. Multicast protocols send only a single copy of a packet over any link in the network and require fewer hops to deliver the packet than unicasting in most cases. Our simulation compares the modified CBT and CBT v2 in which protocol delivers a copy of a packet to all group members in the fewest number of hops.

The experiment we ran, compares the total network resource (hop counts) of CBT and our proposed protocol under different fault probability. Figure 6.13 and figure 6.14 show the results for different fault probability of 0.1 and 0.5 respectively. The experiment was repeated with different delay constraint values to verify the effect on the network resource under fault condition, while multicast group size is fixed at 50. It is shown that the network resource improves as the delay constraint becomes looser which is similar to CBT. Our protocol shows some improvement over the CBT. It should

be noted, however that, with higher fault probability, the network resource remains almost constant at any end-to-end delay constraint value.

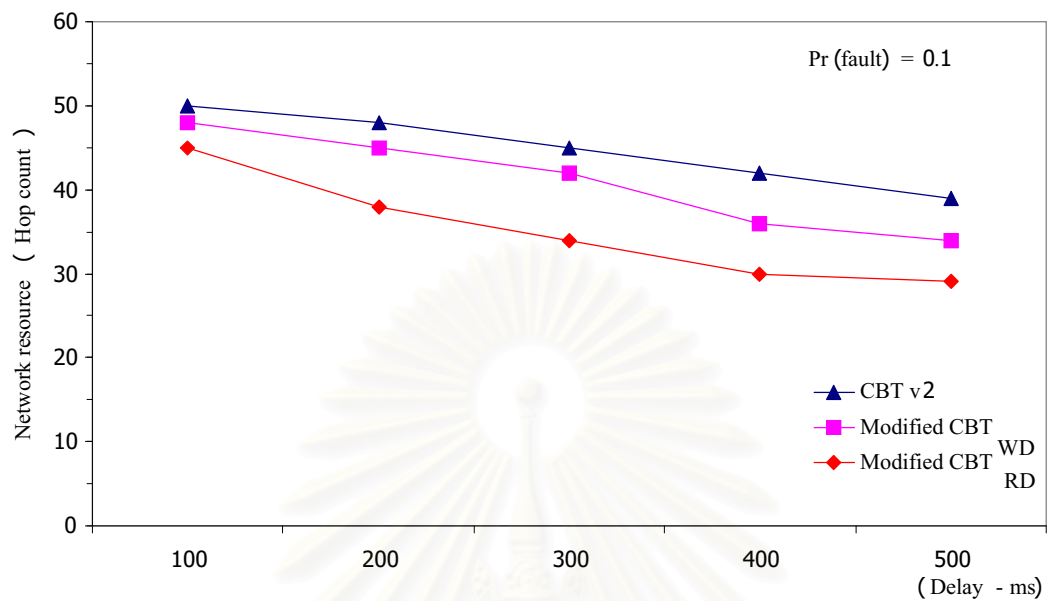


Figure 6.13: Total network resource of the modified CBT_{WD}, Pr (fault) = 0.1, Multicast group size = 50

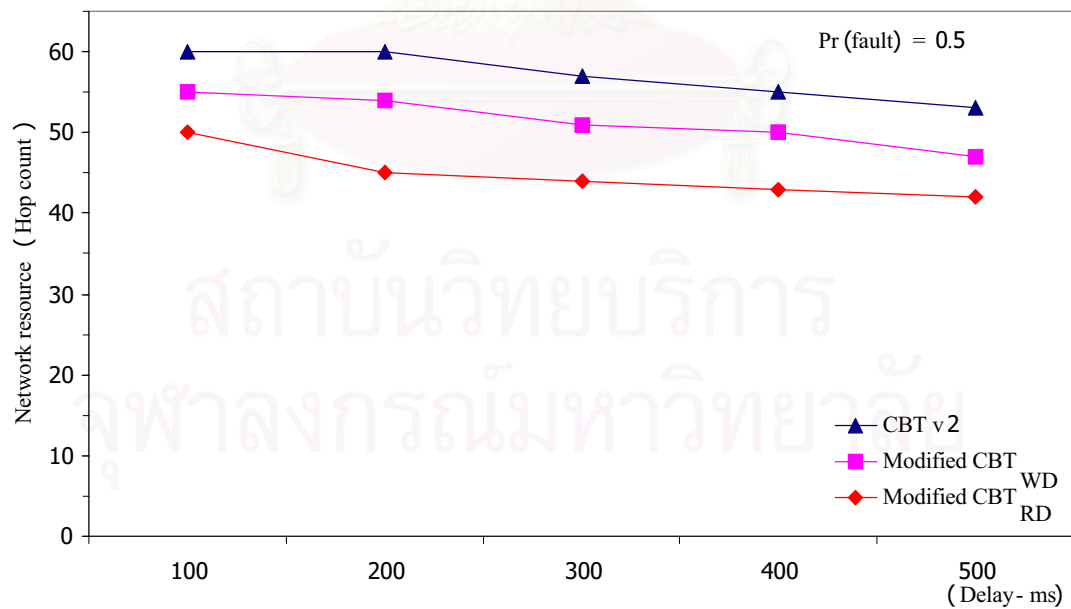


Figure 6.14: Total network resource of the modified CBT_{RD}, Pr (fault) = 0.5, Multicast group size = 50

We then study the effect of faults on CBT v2, our proposed protocol and Jia's algorithm is used as a baseline. The percentage excess network resource (hop count) of a tree generated by our proposed protocol relative to Jia's algorithm is calculated as follows:

$$\frac{\text{Network resource under modified CBT} - \text{Network resource under Jia's algorithm}}{\text{Network resource under Jia's algorithm}} \times 100 \quad (6.7)$$

Figure 6.15 shows that our protocol consumes network resources at 10% worse than Jia's algorithm at smaller fault probability and can get up to almost 60% with higher fault probability. However, we can see some improvement over the CBT v2 from figure 6.15 and figure 6.16. We observe that with larger multicast group size results in slightly higher network resource consumed, as shown in figure 6.16.

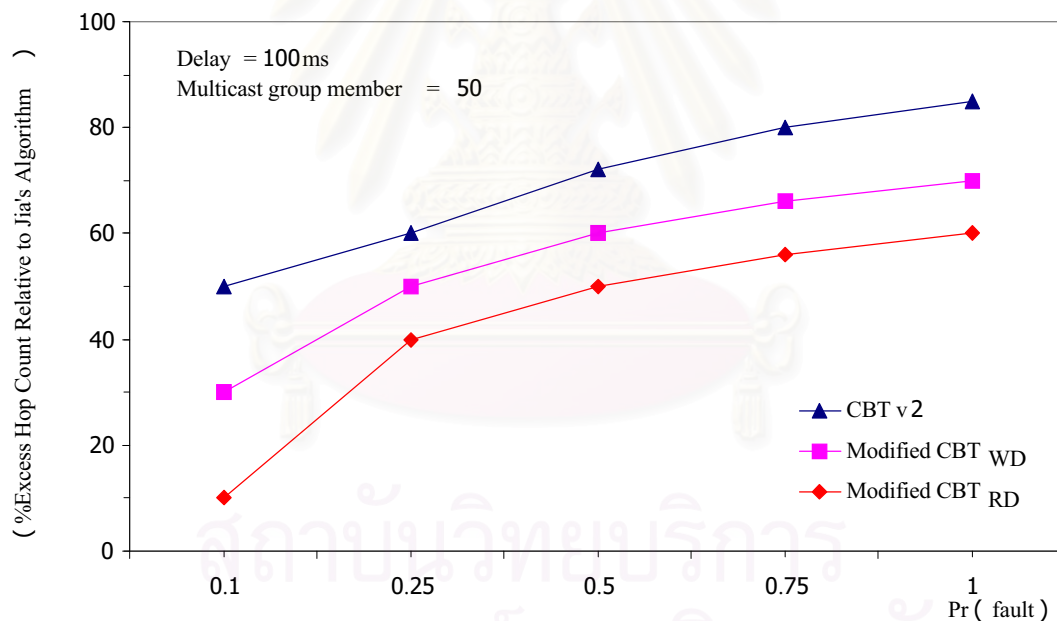


Figure 6.15: Total network resource of modified CBT_{WD} and modified CBT_{RD} relative to Jia's algorithm, multicast group size = 50, Delay constraint value = 100 ms

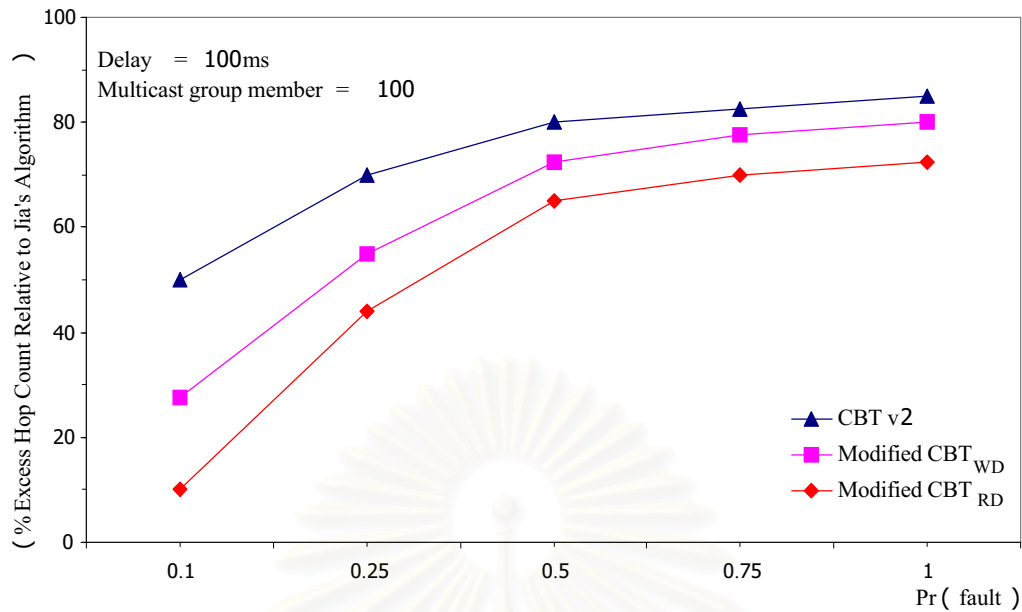


Figure 6.16: Total network resource of modified CBT_{WD} and modified CBT_{RD} relative to Jia's algorithm, multicast group size = 100, Delay constraint value = 100 ms

Figure 6.17 and figure 6.18 show the effects of many different multicast group sizes 20, 50 and 100 on relative network resource at different delay constraint values of our modified CBT_{WD} and modified CBT_{RD}. We found that as the multicast group size increases, our proposed protocol consumed more network resource. The required network resources decreases as the delay constraint value increases. Our modified CBT_{WD} protocol's performance is almost 30% worse than Jia's algorithm when the group size is small and its performance deteriorates fast as the group size increases. Our modified CBT_{RD} performance is slightly better than that of the modified CBT_{WD} in all cases.

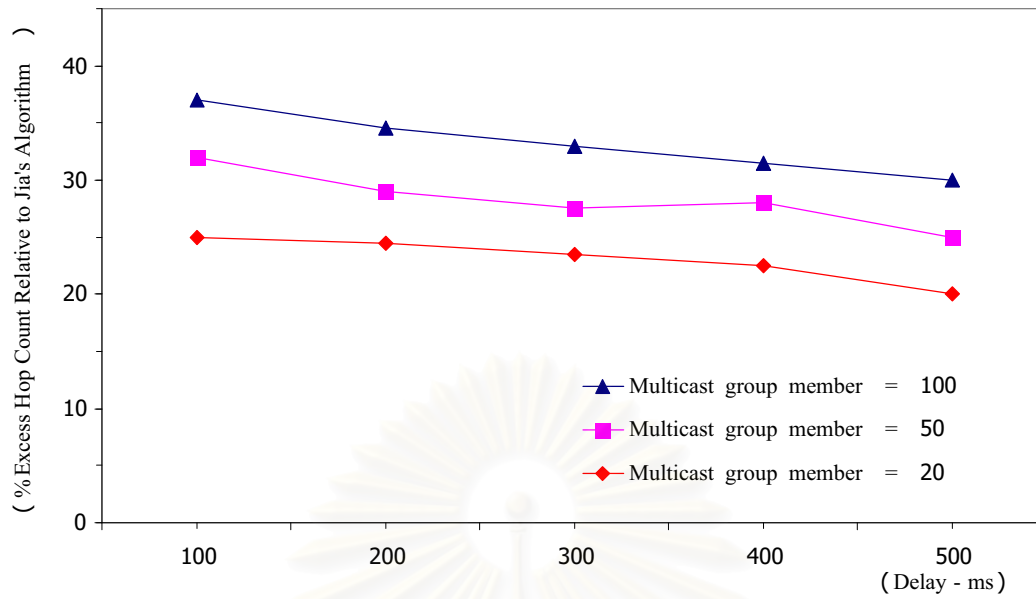


Figure 6.17: Total network resource of a modified CBT_{WD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, $Pr(\text{fault}) = 0.1$

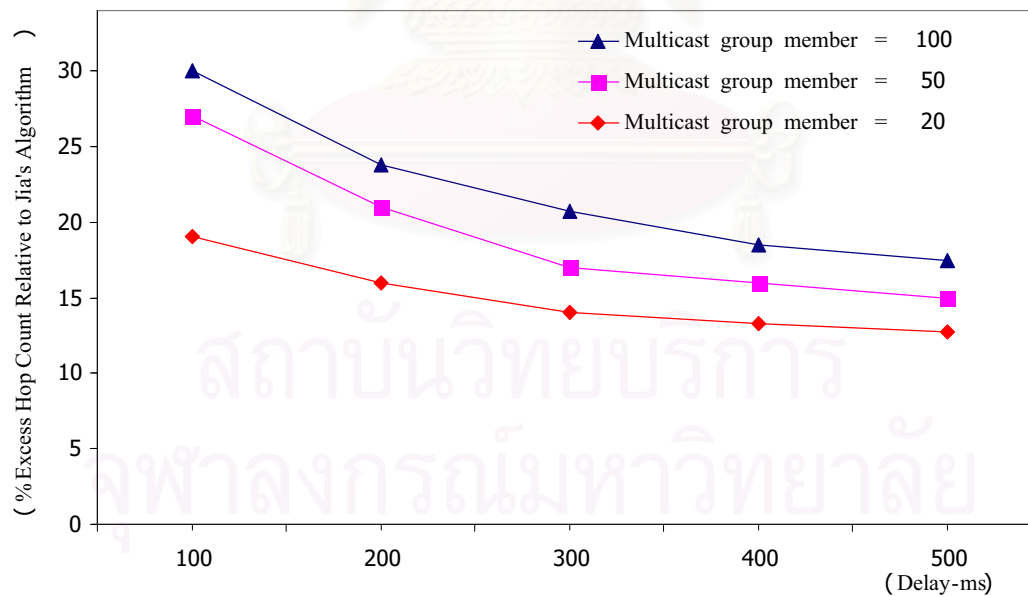


Figure 6.18: Total network resource of a modified CBT_{RD} multicast tree relative to Jia's algorithm with multicast group size 20, 50 and 100, $Pr(\text{fault}) = 0.1$

6.7.3 Traffic concentration

We define a flow to be the stream of packets from a particular sender to a multicast group. Assuming that each source of the group generates traffic at constant rate (1 Mb/s), the total number of traffic flows that traverse a link is counted. Therefore, traffic concentration is measured by the maximum number of flows traversing a unidirectional link (the load of the most congested link). For the same graph and groups, the simulation on CBT and modified CBT were conducted.

We show the result of the experiment with both CBT and our modified protocol with fault probability of 0.1 and 0.5. Figure 6.19, figure 6.20 and figure 6.21 show the results of traffic concentration of CBT, modified CBT_{WD} and modified CBT_{RD} respectively at the fault probability of 0.1. While figure 6.22, figure 6.23, and figure 6.24 show the traffic concentration of the same set of protocols at the higher fault probability of 0.5. We can see that many links are underutilized with 0 flow, while some of the links are overutilized. However, there is no significant change in traffic concentration with larger fault probability. We observe that the distribution of traffic is more evenly with our modified protocols, also results in the decrease of the maximum link load.

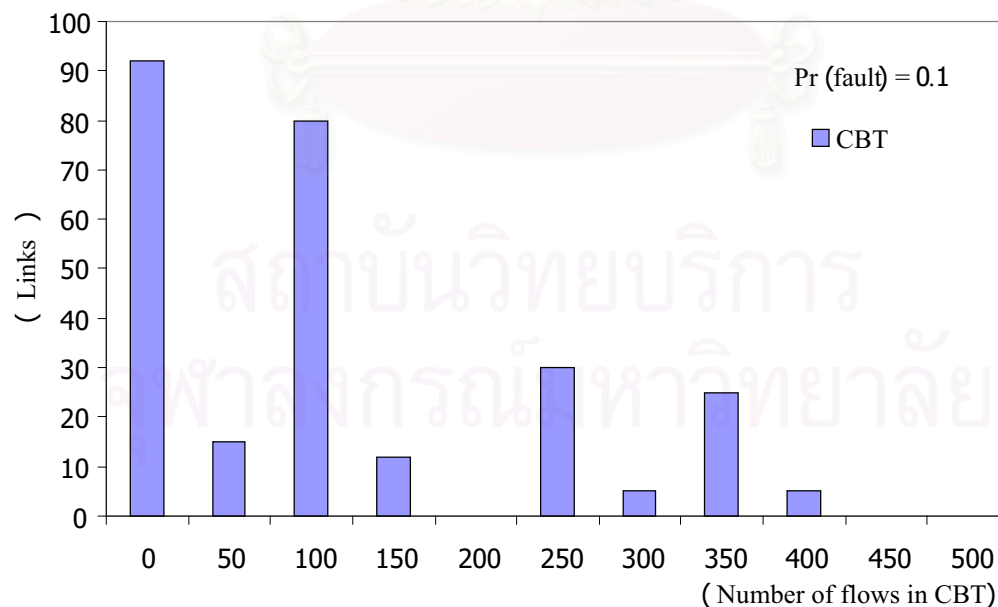


Figure 6.19: Traffic concentration in CBT, Pr (fault) = 0.1

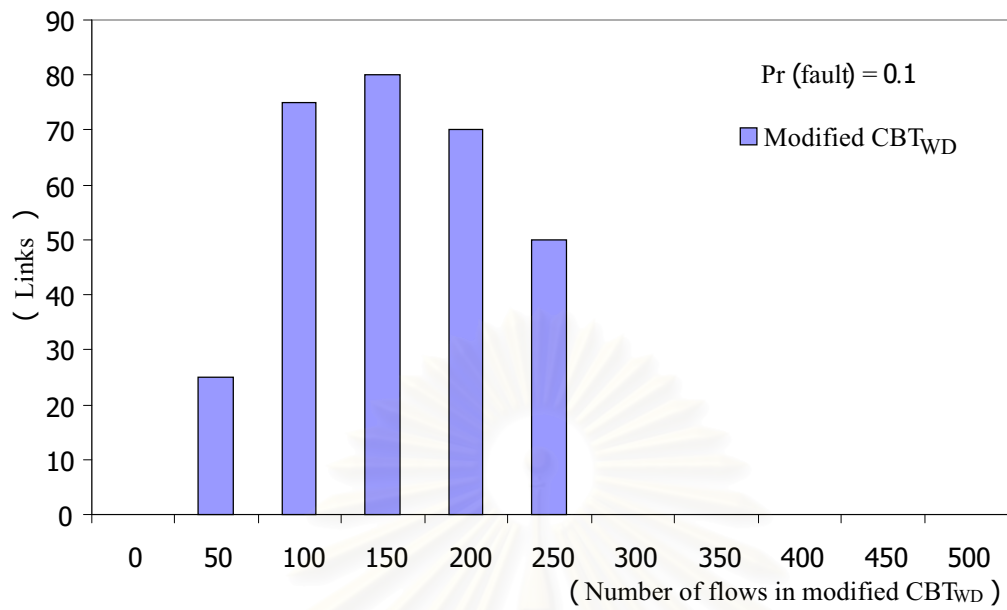


Figure 6.20: Traffic concentration in modified CBT_{WD}, Pr (fault) = 0.1

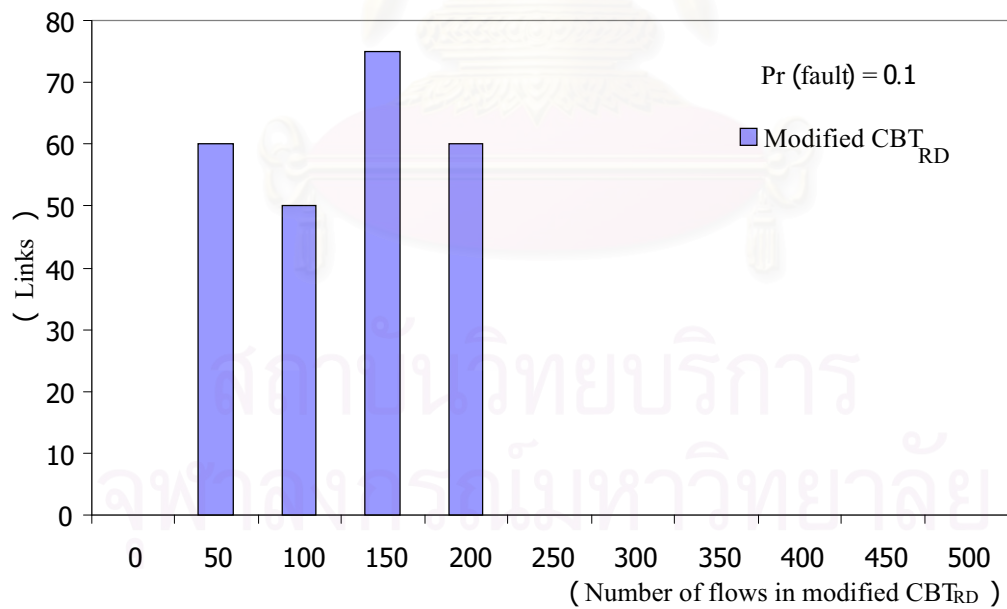


Figure 6.21: Traffic concentration in modified CBT_{RD}, Pr (fault) = 0.1

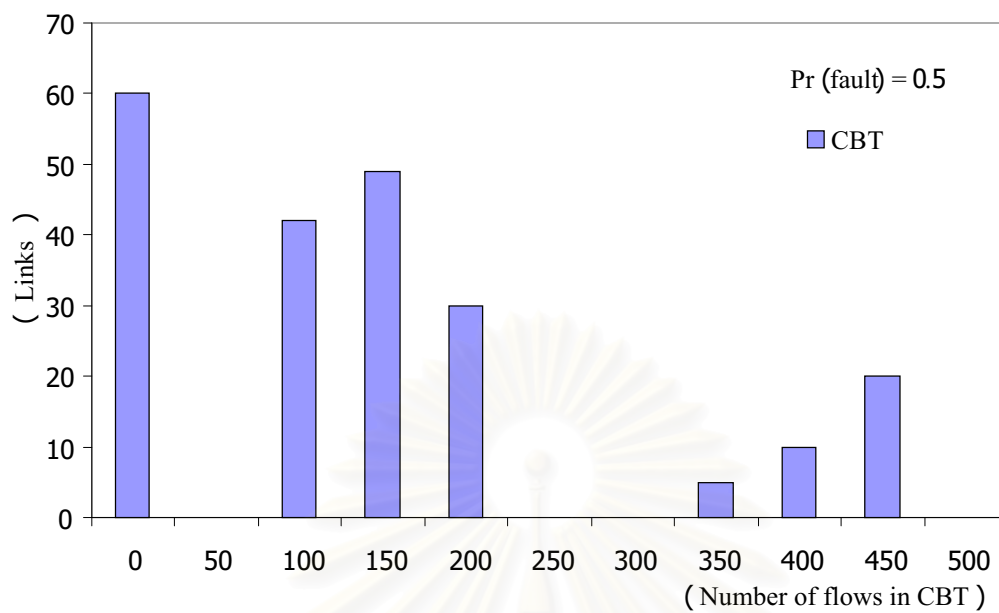


Figure 6.22: Traffic concentration in CBT, $\text{Pr}(\text{fault}) = 0.5$

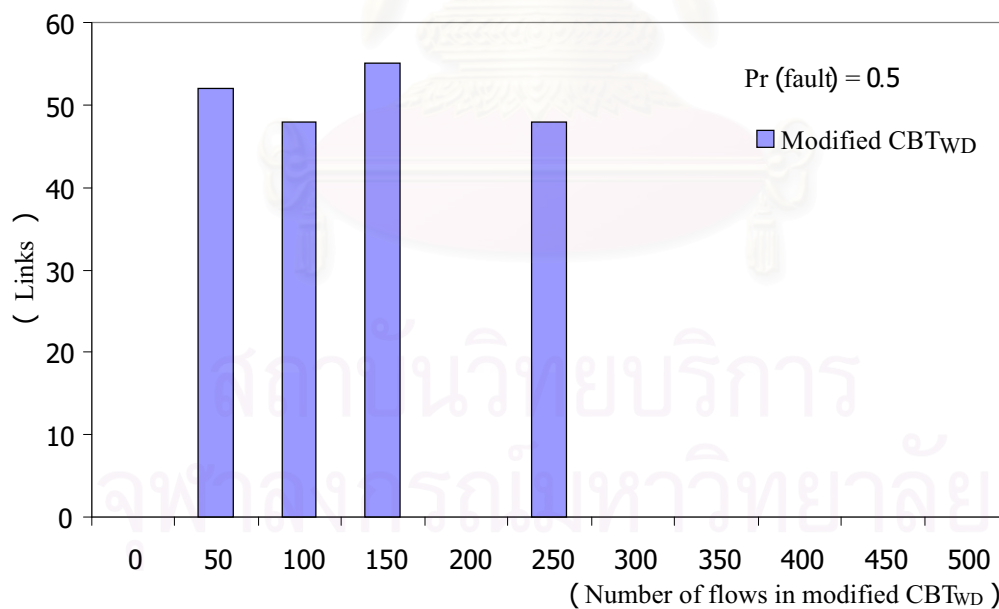


Figure 6.23: Traffic concentration in modified CBT_{WD}, $\text{Pr}(\text{fault}) = 0.5$

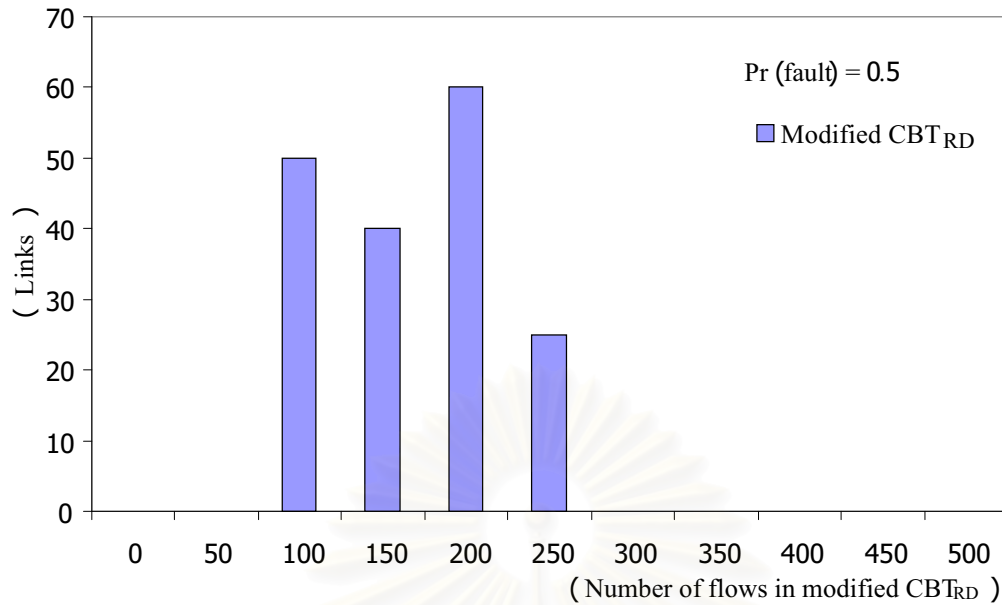


Figure 6.24: Traffic concentration in modified CBT_{RD}, Pr (fault) = 0.5

We conducted another experiment to verify the effect of fault probability and the relative maximum link load of CBT and modified CBT. The experiment was conducted repeatedly with different group sizes of 20, 50 and 100. Figure 6.25 and figure 6.26 show that our modified protocol maintained relative constant maximum link load compared to that of CBT.

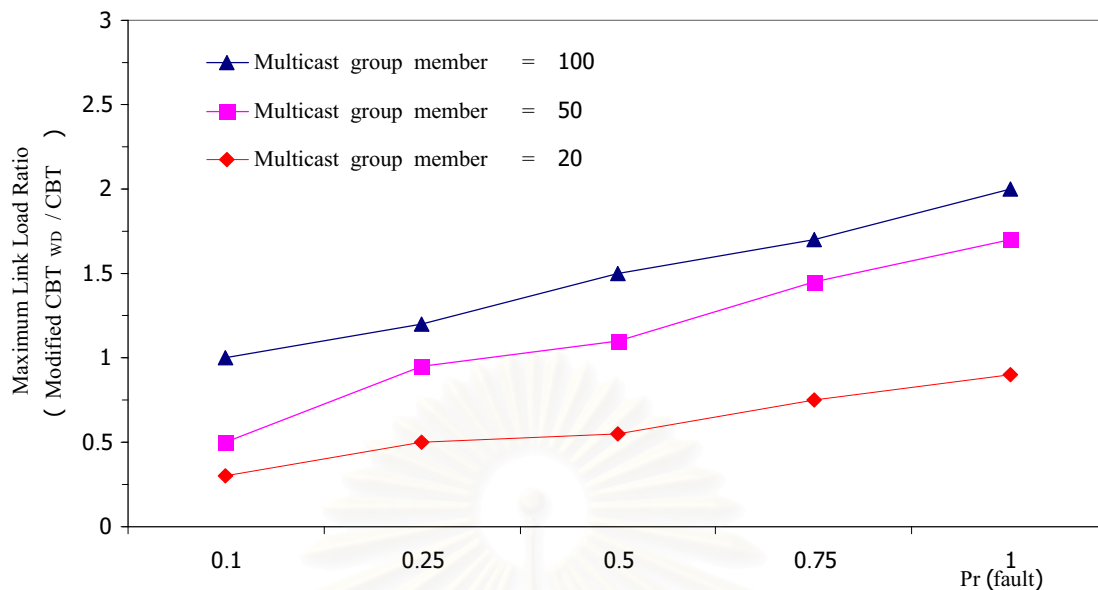


Figure 6.25: Relative Maximum link load between CBT and modified CBT_{WD} with multicast group size = 20, 50 and 100

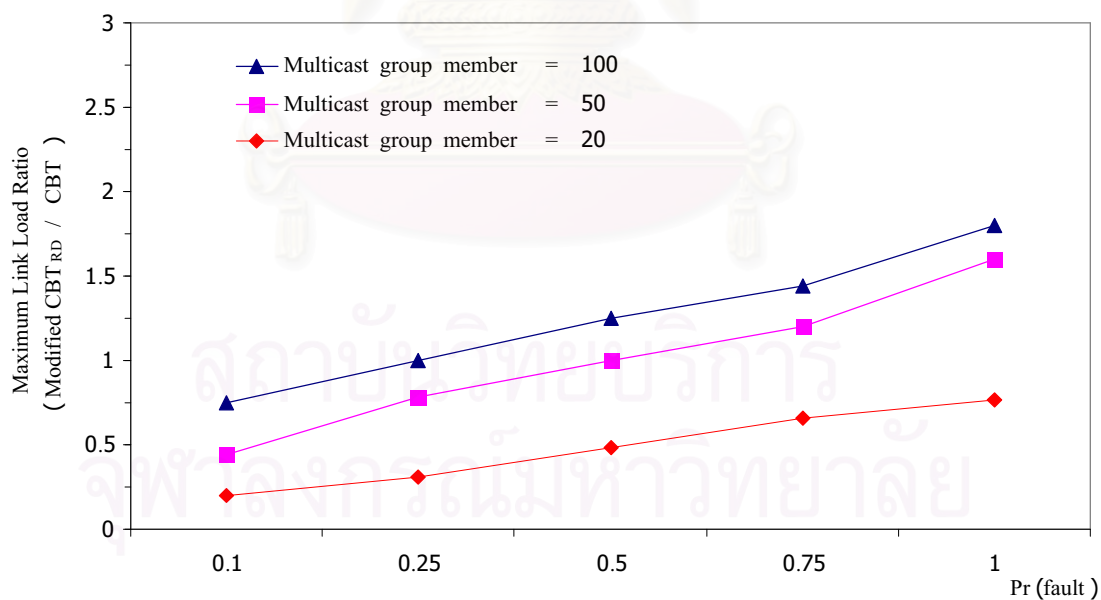


Figure 6.26: Relative Maximum link load between CBT and modified CBT_{RD} with multicast group size = 20, 50 and 100

6.7.4 Loss Rate

We measured the loss rate of the packets by examining the packet arrival time at the destination node versus the time the packet was generated at the source node. The number of packets arrived within the time bound, which is the real-time constraint was recorded. Since delay bound violation is one of the reasons to reject a multicast tree. Therefore, loss rate is defined as the ratio of the total number of loss packets or late packets, to the total number of packets transmitted.

We conduct the experiment to compare the relative loss rate with Jia's algorithm to verify the effectiveness of our protocol at 0.1 fault probability and under different delay constraint values. The results, which are given in figure 6.27 and figure 6.28, show that our proposed protocols have less loss rate compared with that of CBT. As the end-to-end delay becomes less stringent, the loss rate declines.

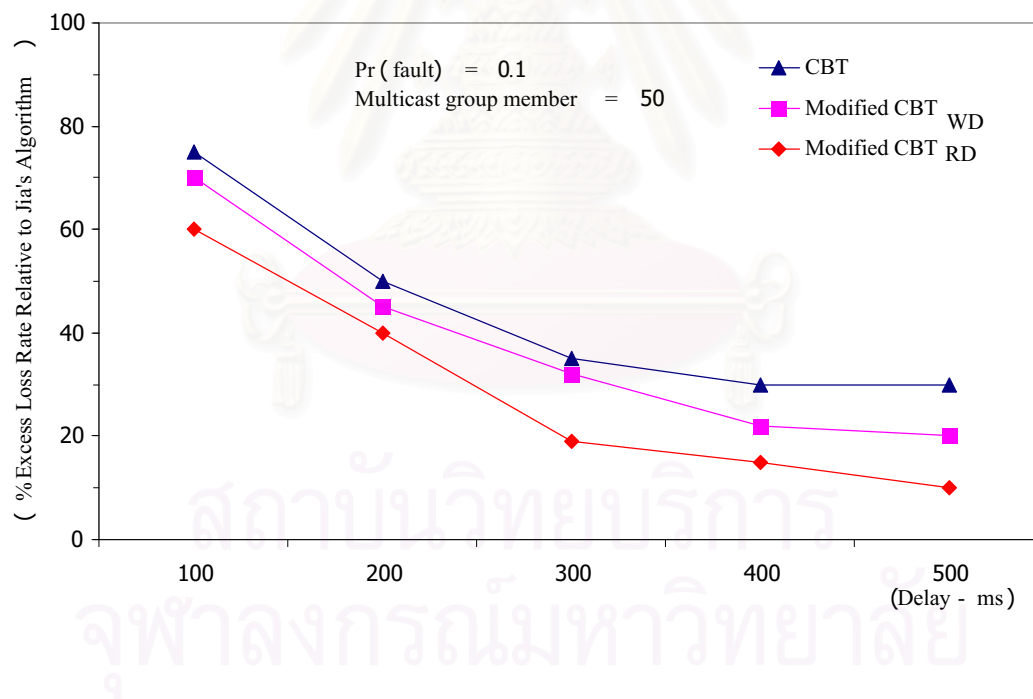


Figure 6.27: Total loss rate relative to Jia's algorithm, Pr (fault) = 0.1, multicast group size = 50

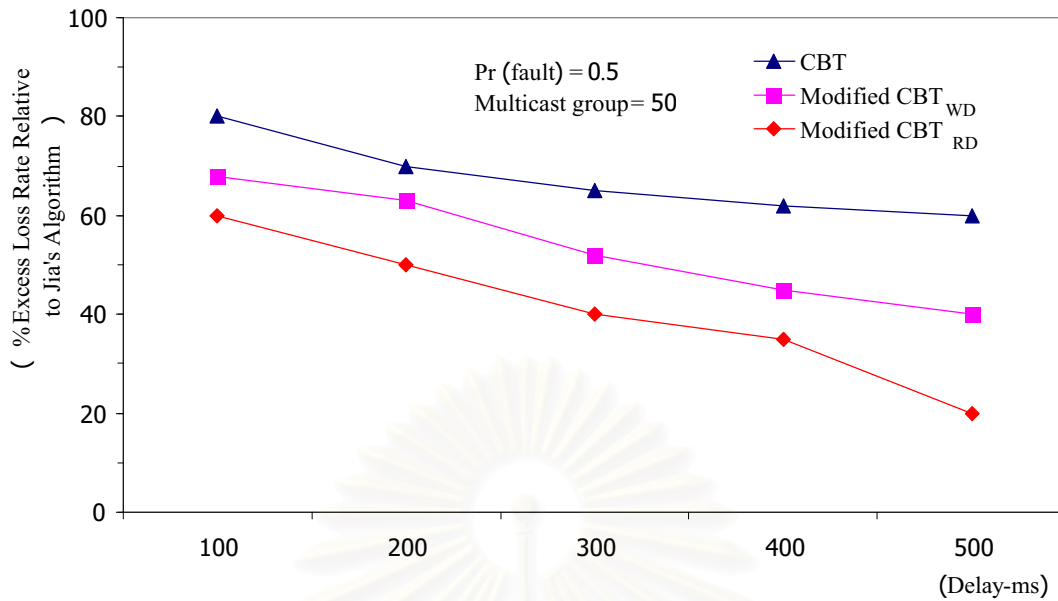


Figure 6.28: Total loss rate relative to Jia's algorithm, $\text{Pr}(\text{fault}) = 0.5$, multicast group size = 50

Our protocol performs relatively better with larger fault probability shown in figure 6.21. We compare the loss rate performance of both protocols in different multicast group sizes of 20, 50 and 100 which is shown in figure 6.29 and figure 6.30. We can see similar results in all cases. We then repeated the experiment to compare the effect of fault probability on our protocol relative to Jia's algorithm. The percentage excess loss rate of a tree generated by our proposed protocol relative to Jia's algorithm is calculated as follows:

$$\frac{\text{Loss rate under modified CBT} - \text{Loss rate under Jia's algorithm}}{\text{Loss rate under Jia's algorithm}} \times 100 \quad (6.8)$$

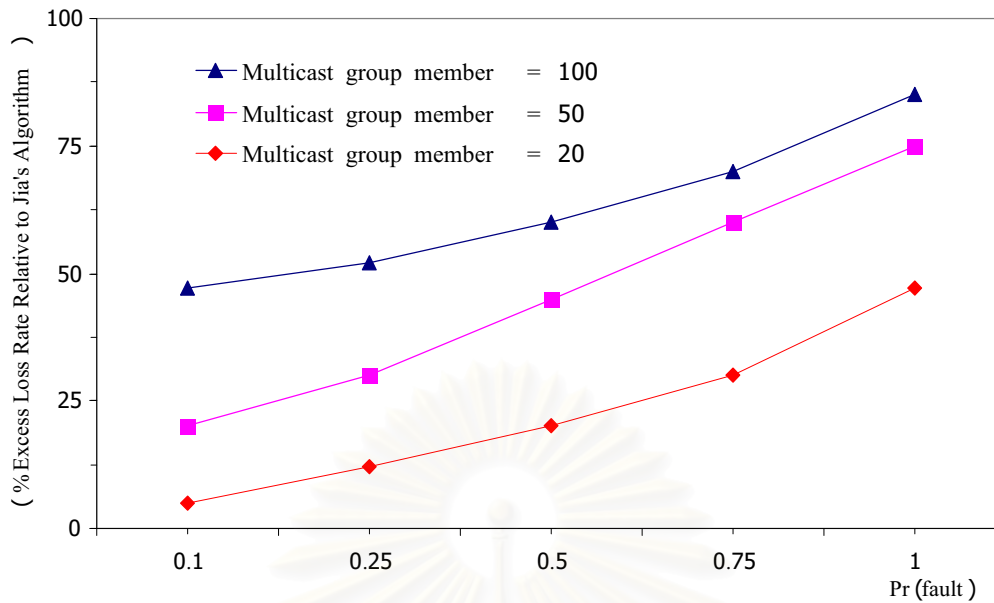


Figure 6.29: Total loss rate of a modified CBT_{WD} multicast tree relative to Jia's algorithm with multicast group size = 20, 50 and 100, end-to-end delay = 100 ms.

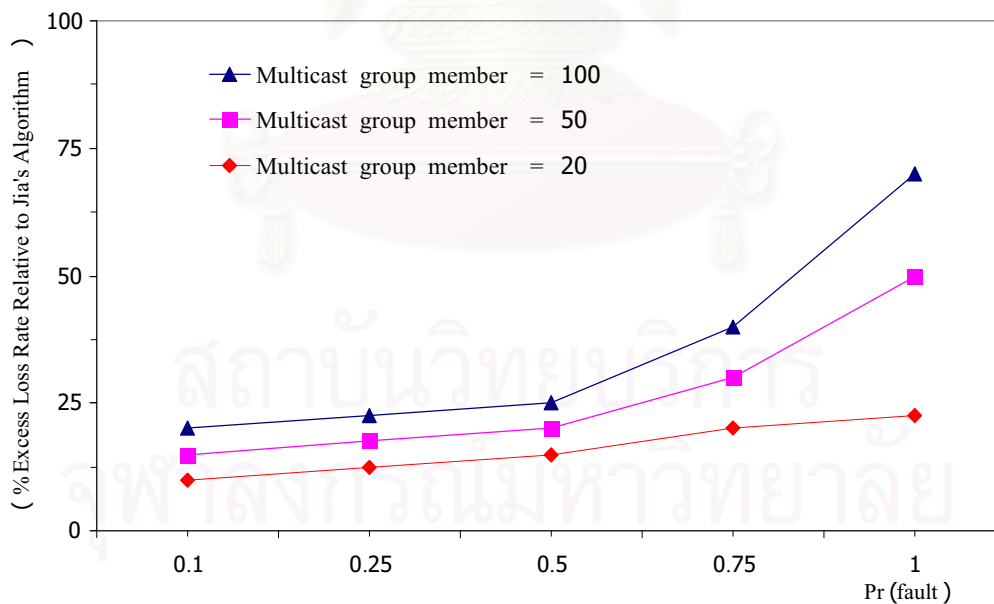


Figure 6.30: Total loss rate of a modified CBT_{RD} multicast tree relative to Jia's algorithm with multicast group size = 20, 50 and 100, end-to-end delay = 100 ms.

We found that the loss rate of our proposed protocol is around 10% worse than Jia's algorithm and its performance deteriorates within 20% when the multicast group size increases to 100. With tighter end-to-end delay constraint values, our proposed protocol may not be able to find the backup paths that fulfill such conditions which results in high loss rate. At a fixed fault probability, the relative loss rate of our proposed protocol increases as multicast group size increases. As expected, the relative loss rate is worse with higher fault probability. We found that Jia's algorithm performance deteriorates when the fault probability is relatively high because it has to recompute the multicast tree every time a fault is encountered, thus resulting in high loss rate.

6.7.5 Execution Time

Figure 6.31 and figure 6.32 show the growth of the execution time with the multicast group sizes up to 100 members at fault probability of 0.1 and 0.5 respectively. The running time of Jia's algorithm is quite large and grows relatively fast as the multicast group size increases. Our proposed protocol did relatively well but took more time to build the complete multicast tree than the original CBT, since all disjoint backup paths must be verified by the admission tests. Hence, it takes more time than that of the CBT. With higher fault probability, our proposed protocol did very well and shows the best performance in terms of execution times. Jia's algorithm and CBT are less efficient with larger multicast group at the higher fault rate. This shows the effectiveness of our protocol in handling the fault-tolerant scenario.

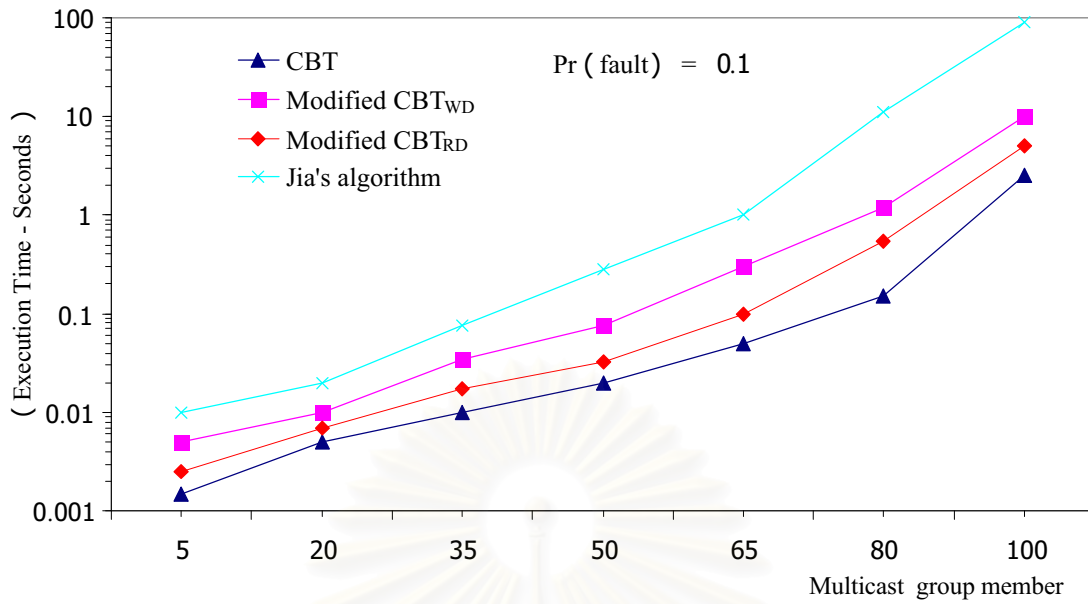


Figure 6.31: Execution Time, Variable multicast group size, Pr (fault) = 0.1

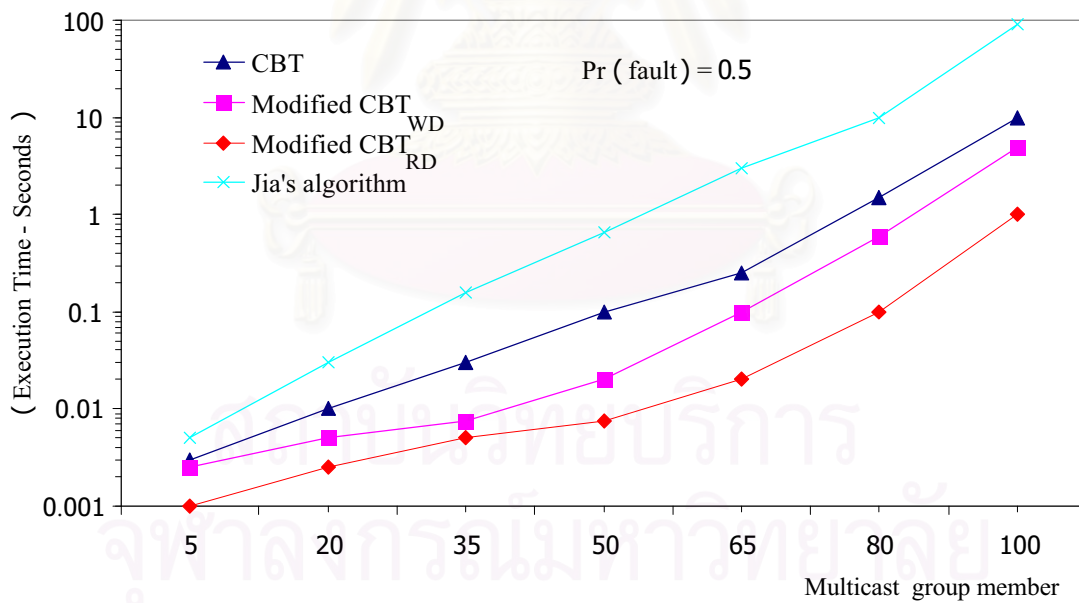


Figure 6.32: Execution Time, Variable multicast group size, Pr (fault) = 0.5

6.7 Conclusions

We proposed the fault-tolerant features on top of our real-time protocol in chapter 4, to cope with a single link failure in the network. The pre-defined backup paths were computed as alternate paths for all node pair in the multicast group. Then the alternate paths are tested to verify that they passed the end-to-end delay constraint. Our protocol is responsible for invoking the backup paths and rerouting the network such that the excess delay value along with the delay value does not violate the real-time constraint Δ .

We have proposed dispersity routing as a mechanism to provide fault tolerance to real-time communication network. We investigated the rerouting of guaranteed performance service connections on the occurrence of link faults, focusing on the aspects of route selection, disjoint backup paths and routing table reconfiguration in the network. The fault-tolerance strategy of the original CBT was described. Our strategy to improve the fault-tolerance feature of CBT was studied. The concept of dependable network was discussed and studied. The DCM routing algorithm was proposed to find the disjoint backup path between each node pair, i.e., source and destination node in each multicast group. The algorithm determines an alternative path from the source node to the destination node taking into consideration the traffic and other performance characteristics of the connection. The DCM routing algorithm maximizes throughput by minimizing the number of intermediate nodes encountered along the path from the source to the destinations. When a single failure occurs, DCM algorithm also checks if the disjoint backup path, which was pre-computed, can meet the time delay constraint, and verifies if there is enough network resource, before rerouting all routers involved in the failure.

Last, we analyzed the multiple simultaneous fault situation, whereby multiple disjoint backup paths between the involved routers and to its grandfather must be specified. We conclude that to tolerate m simultaneous faults, for each router, we need additional backup paths to connect not only to its grandfather node, but also other ancestors.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this dissertation, we studied a number of routing problems for real-time communication on the Internet. Real-time applications, e.g. multimedia and real-time control, have QoS service requirements which must be guaranteed by the underlying network. A *guarantee* is a contract between an application and the service provider to satisfy a requested level of QoS requirements. However, the “best-effort paradigm” offered by the current datagram service, which proved to be very successful in the realization of a universal network in a heterogeneous environment, is not adequate to support real-time traffic. The network protocol offers no guarantee about timely, reliable, and ordered delivery of packets.

One QoS requirement, the end-to-end delay constraint, can be guaranteed in wide-area networks by using the appropriate routing algorithms. However, another approach to support applications’ real-time requirements, is to use the combination of scheduling algorithms, admission control, and a reservation protocol, which is not the subject of this dissertation. Support for multicast communication will play a prominent role in the future as the deployment of real-time multicast-based applications will continue to grow. It is, therefore, important that network developers be aware of the design issues related to real-time multicast communication in order to avoid impacting the performance of the multimedia applications. It is absolutely necessary to enhance the reliability of applications in such nature.

To further increase the reliability of the real-time multicast applications, we investigate and study the failure handling and recovery of the multicast routing protocols. We believe that to increase the tolerance to failure, it is possible to incorporate useful fault-recovery and failure handling in designing a real-time multicast routing protocol. The objective is to obtain the desired QoS even when the network faces a failure situation.

In chapter 2, we surveyed previous work on multicast routing with no constraint and under time delay constraint. Chapter 3 starts with the descriptions of the core-based tree (CBT), its architecture, protocol overview, protocol format and its functions.

Chapter 4 presents a formal definition of the real-time multicast problems, previous approaches of the real-time multicast problems, and our proposed approaches. We also described how the previous approaches work and discuss advantages and disadvantages of each approach. We introduce a technique of optimizing the path previously used by a source reaching the shared multicast tree by using the shortest of the shortest path, to a specific on-tree node. Then, we proposed two new path selection methods, bases on Dijkstra's algorithm and Residual delay concept by Kompella's algorithm, to find optimal paths for the new multicast trees whereby the end-to-end delay condition is not violated. With our approaches, the cost of the multicast tree diminished substantially. We then described our simulation to prove the efficacy of our protocol which bases on CBT v2. We compared our proposed protocol with the CBT v2 and show its improvement in terms of average end-to-end delay, network cost, network resource usage, traffic concentration, loss rate and execution time.

Chapter 5 presents the concept of fault-tolerance framework, failure model and solution to recover the network from failure. We study the concept of depedable network whereby disjoint backup paths are identified off-line. The resource to the disjoint backup path will be allocated at runtime. The runtime overhead will be reduced significantly without compromising the network performance.

In chapter 6, we study the fault-tolerance aspect of the real-time traffic, and approaches to sustain the reliability of the network under a single link failure condition. We discuss many approaches to improve the network redundancies and introduce our fault-tolerant approaches to the algorithms proposed in chapter 4. Our approach is based on first, finding the pre-computed disjoint backup paths and secondly, proposing an algorithm to switch from the primary link to secondary link within end-to-end delay constraints imposed by the applications. This involves the admission test of the pre-computed backup paths prior to switching of the failed path to the backup path. We show that our proposed approaches can reliably switch the multicast traffic along the

failed link to the backup link within time bound and still performs better than the CBT v2 in terms of the performance metrics, described in previous chapter.

7.1 Future Work

Although we have made important progress in studying and implementing the fault-tolerant real-time multicast protocols based on the core-based tree, a number of issues still need to be explored further, as follows.

- CBT v3: In this dissertation, we based our experiment on CBT v2 which is well-known and resource rich. CBT v3 which improves the loop removal in the protocol should be explored in the future research. By distributing cores throughout the network and by maintaining logical level topological information, CBT v3 allows for a flexible multicast group in which the core structure does not have to be fixed in advance. We anticipate that more overhead in detecting the tree loop will be introduced, if our enhancement is implemented using CBT v3 protocol.
- Resource reservation and admission control: Another approach to improving the performance of real-time applications is to integrate resource reservation and admission control into the routing protocol. Applications use the reservation protocol to request resources from routers and are either accepted or rejected by admission control at each router. We suggest that future work on real-time multicast routing should take this aspect into consideration.
- Hierarchical structure of state information exchange: We anticipate that additional work is needed to minimize the memory and processing power of each node maintaining the current tree information. The concept of hierarchical routing can be applied to the routing protocol because the underlying routing mechanisms used to disseminate topology data require the aggregation of information in order to cope with growing network size.
- Dynamic join and leave/Tree arrangement: In the dissertation, we considered a static set of multicast group members. One may handle

dynamic member join/leave by incrementally changing the multicast tree. When a new member intends to join the distribution tree, a tree branch connects the new member to the nearest tree node. When a member leaves the multicast group, only the corresponding tree branch is torn down. The main idea of the multicast tree rearrangement issue, is to define and monitor certain damage index to the multicast tree as members join/leave, and trigger tree rearrangement when the index exceeds certain threshold. Allowing nodes to join and leave an existing multicast group dynamically is another feature that should be considered in our future work.

- Multiple simultaneous faults: The proposed fault-tolerant protocol was designed to handle a single link fault that occurs on CBT paths. In order to tolerate m simultaneous faults, a trivial extension to backup paths selection is to select m disjointed backup paths for each router. These paths connect the router to its grandfather. This should be explored in the future research work on fault-tolerant multicast.
- Implementation of our modified CBT protocols: In this dissertation, we have presented the modified CBT protocols that can sustain the maximum delay bound condition imposed by the applications and tolerate a single fault in the network by simulation. However, in order to prove the usefulness of our approach, it is important to investigate the performance of our proposed protocols through experiments in a real network such as the Internet.

7.2 Summary of Main Contributions

Our contributions in this dissertation are the following:

- We proposed a shortest to the shortest path to optimize the path from source to the multicast tree thus bypassing the core router. The conventional CBT finds the shortest path tree from the source node to the core node and uses this path to route the traffic to all multicast members, resulting in traffic concentration around the core. With this modification, CBT has more chance to meet the end-to-end delay constraints, while traffic concentration around core router decreases significantly.

- We proposed two new path selection methods to ensure that the paths from source to all members in the shared multicast tree do not violate the end-to-end delay constraint condition, while cost of the multicast tree is reduced substantially. The first path selection method is based on weighted dijkstra's path selection algorithm. The Dijkstra's Shortest Path Tree algorithm was modified by substituting the original path selection with the weighted path selection function in order to create optimal solution balancing cost and delay parameters. The second path selection method is based on Kompella's selection function which considers residual delay in addition to low cost tree. The new selection function explicitly uses both cost and delay in its functional form. It tries to choose low cost paths, but modulates the choice by trying to pick edges that maximize the residual delay. The idea is to reduce the cost of the tree through path sharing. We used simulation to prove the efficacy of our protocols against the original CBT v2.
- We proposed a new fault-tolerant protocol to enhance the proposed optimal real-time protocol. The idea is to pre-compute disjoint backup paths between each node pair for all links in the multicast tree. When a certain link on the multicast tree fails, our protocol will check if there exists a disjoint backup path for that link. The protocol will also check if the new disjoint backup paths can sustain end-to-end delay constraints. Then, the protocol verifies if the switching of routes from the failed link to the backup link takes no more than the maximum time delay allowed.
- We used simulation to evaluate the algorithms we proposed. In our simulation, we imitated the realistic Internet network environment. We conducted the experiment using ns-2 simulator to simulate a random network. The simulation shows that our proposed protocol performs well under a single link failure condition with moderate execution time and lower cost tree compared to that of the original CBT, while traffic concentration and network resource usage decreased substantially. Our proposed protocol verifies if the end-to-end delay condition is not violated in the simulated scenarios.

REFERENCES

1. Ammar, M., Cheung, S., and Scoglio, C. Routing multipoint connections using virtual paths in an ATM networks. In proceedings of IEEE INFOCOM '93 (1993) : 98-105.
2. Awerbuch, B., Bar-Noy, A., and Gopal, M. Approximate distributed bellman-ford algorithms. In proceedings of IEEE INFOCOM '91 (1991) :1206-1213.
3. Bahk, S., and Zarik, M. A dynamic multi-path routing algorithm for ATM networks. J. High Speed Networks Vol. 1, No. 3 (1992).
4. Ballardie, A. CBT border router specification for interconnecting a CBT stub region to a DVMRP backbone. ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-cbt-dm-interop-*.txt. Working draft, March 1997.
5. Ballardie, A. Core based trees (CBT version 2) multicast routing - protocol specification. RFC2189 (September 1997).
6. Ballardie, T., Francis, P., and Crowcroft, J. Core based trees (CBT): An architecture for scalable inter-domain multicast routing. Proceedings of the ACM SIGCOMM, San Francisco (September 1993).
7. Banerjea, A. A taxonomy of dispersity routing schemes for fault tolerant realtime channels. In proceedings of European Conf. Multimedia Application, Services and Techniques (ECMAST'96) Louvainla-Neuve, Belium (May 1996).
8. Banerjea, A. Fault recovery for guaranteed performance communications connections. IEEE/ACM transactions on networking Vol. 7, No. 5 (October 1999).
9. Banerjea, A. Simulation study of the capacity effects of dispersity routing for fault tolerant real time channels. In proceedings of SIGCOMM Symp. (August 1996) : 194-205.

10. Barath-Kumar, K., and Jaffe, J. Routing to multiple destinations in computer networks. IEEE transactions on communications vol. COM-31, no. 3 (March 1983) : 343-351.
11. Bauer, F., and Varma, A. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. IEEE JSAC (April 1997) : 382-397.
12. Bauer, F., and Varma, A. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. In proceedings of IEEE INFOCOM '96 (1996) : 361-368.
13. Bauer, F., and Varma, A. Degree-constrained multicasting in point-to-point networks. In proceedings of IEEE INFOCOM '95 (1995) : 369-376.
14. Bauer, F., and Varma, A. Distributed algorithms for multicast path setup in data networks. IEEE/ACM transactions on networking vol. 4, no. 2 (April 1996) : 181-191.
15. Bauer, F., and Varma, A. Distributed algorithms for multicast path setup in data networks. Computer Engineering Department, University of California, Santa Cruz (UCSC-CRL-95-10 (August 1995).
16. Bellman, R. Dynamic programming. Princeton University Press (1957).
17. Bertsekas, D. Linear network optimization: algorithms and codes. MIT Press (1991).
18. Bertsekas, D., and Gallager, R. Data networks. (n.p.): 2nd ed., Prentice-Hall (1992).
19. Biersack, E., and Nonnenmacher, J. WAVE: A new multicast routing algorithm for static and dynamic multicast groups. In proceedings of the fifth international workshop on network and operating system support for digital audio and video (1995) : 228-239.
20. Calvert, K., Zegura, E., and Donahoo, M. Core selection methods for multicast routing. In proceedings of the fourth international conference on computer communications and networking (IC³ N '95) (1995) : 638-642.

21. Chen, B., Kamat, S., and Zhao, Wei. Fault-tolerant real-time communication in FDDI-based networks. IEEE Computers (April 1997).
22. Chen, S., and Nahrstedt, K. Distributed QoS routing in high-speed networks based on selective probing. Technical report, CSD, UIUC, 1998.
23. Chow, C.-H., On multicast path finding algorithms. In proceedings of IEEE INFOCOM '91 (1991) : 1274-1283.
24. Cormen, T., Leiserson, C., and Rivest, R. Introduction to algorithms. MIT Press (1997).
25. Dalal, Y., and Metcalfe, R. Reverse path forwarding of broadcast packets. Communications of the ACM vol. 21, no. 12 (December 1978) : 1040-1048.
26. Deering, S. Multicast routing in a datagram internetwork. PhD thesis, Stanford University, December 1991.
27. Deering, S., and Cheriton, D. Multicast routing in datagram internetworks and Extended LANs. ACM Transactions on Computer Systems vol. 8, no. 2 (May 1990) : 85-110.
28. Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C., and Wei, L. Protocol independent multicast (PIM): Protocol specification. Internet Draft RFC (1995).
29. Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C., and Wei, L. The PIM architecture for wide-area multicast routing. IEEE/ACM Trans. Networking Vol. 4, No. 2 (April 1996) : 153-162.
30. Dijkstra, E. A note on two problems in connection with graphs. Numer. Math. Vol. 1 (1959).
31. Dijkstra, E. Two problems in connection with graphs. Numerische Mathematik vol. 1, no. 5 (October 1959) : 269-271.
32. Doar, M. Multicast in the asynchronous transfer mode environment. PhD thesis University of Cambridge, January 1993.

33. Doar, M., and Leslie, I. How bad is naive multicast routing. In proceedings of IEEE INFOCOM '93 (1993) : 82-89.
34. Donahoo, M., and Zegura, E. Core migration for dynamic multicast routing. In proceedings of the fifth international conference on computer communications and networking (IC³ N '96) (1996) : 92-98.
35. Estrin, D., et al. Protocol Independent Multicast (PIM) Sparse Mode/Dense Mode. Available from: <ftp://netweb.usc.edu/pim> Working drafts, 1996.
36. Estrin, D. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing; et al.; Technical Report, Available from: <ftp://catarina.usc.edu/pim>
37. Fenner, W. Internet Group Management Protocol, version 2 (IGMPv2). Available from: ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-igmp-v2-**.txt. Working draft, 1996.
38. Ferrari, D., and Verma, D. A scheme for real-time channel establishment wide-area network. Tech. Rpt.-89-022, International Computer Science Institute, Berkeley, California, May 1989.
39. Fredman, M., and Tarjan, R. Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM Vol. 34, No. 3 (1987) : 596-615.
40. Frieze, A. Disjoint Paths in Expander Graphs via Random Walks: a Short Survey. To appear in Proceedings of Random'98 (1998).
41. Gallager, R., Humblet, P., and Spira, P. A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and Systems (January 1983) : 66-77.
42. Garey, M., and Johnson, D. Computers and intractability: A guide to the Theory of NP-Completeness. New York : W.H. Freeman and Co. (1979).
43. Haberman, B., and Rouskas, G. Cost, delay, and delay variation conscious multicast routing. Technical Report TR-97-03, North Carolina State University, 1997.

44. Hwang, F., and Richards, D. Steiner Tree Problems. Networks vol. 22, no. 1 (January 1992) : 55-89.
45. Imase, M., and Waxman, B. Dynamic Steiner tree problem. SIAM j. Disc. Math. (August 1991) : 369-384.
46. Jia, W. Zhao, W., Xuan, D., and Xu, G. An efficient fault-tolerant multicast routing protocol with core-based tree techniques. IEEE Transactions on Parallel and Distributed Systems Vol.10, No.10 (October 1999).
47. Jia, X. A Distributed Algorithm of Delay-Bounded Multicast Routing for Multimedia Applications in Wide Area Networks. IEEE/ACM Transactions on Networking Vol. 6, No. 6 (December 1998) : 828-837.
48. Jia, X. Optimal multicast routing with quality of service constraints. J. Network Syst. Manament Vol. 4, No. 2 (1996) : 107-131.
49. Jia, X., Cao, J., and Jia, W. Real-time multicast routing with optimal network cost. Proceedings of the IEEE Real-Time Computing Systems and Applications Workshop (1996).
50. Jia, X., Passinou, N., and Makki, K. A real-time multicast routing algorithm for multimedia applications. Computer Commun. J. Vol. 20, No. 12 (November 1997) : 1098-1106.
51. Jia, X., Zhang, Y., Pissinou, N., and Makki, S. An integrated routing and admission control mechanism for real-time multicast connection establishment. International Journal of Communication systems Vol. 14, Issue 3 (April 2001) : 287-303.
52. Jiang, X. Routing Broadband Multicast Streams. Computer Communications vol. 15, no. 1 (January, February 1992) : 45-51.
53. Jiang, X. Routing Broadband Multicast Streams. Computer Communications vol. 16, no. 12 (December 1993) : 767-775.

54. Kadirire, J. Minimising Packet Copies in Multicast Routing by Exploiting Geographic Sperad. Computer Communication Review vol. 24, no. 3 (July 1994) : 47-62.
55. Kadirire, J., and Knight, G. Comparison of Dynamic Multicast Routing Algorithms for Wide-area Packet Switched (Asynchronous Transfer Mode) Networks. In Proceedings of IEEE INFOCOM '95 (1995) : 212-219.
56. Karp, R. Reducibility among Combinatorial Problems. In Complexity of Computer Computations (R.Miller and J. Thatcher, eds.) Plenum Press (1972) : 85-103.
57. Katabi, D. The use of IP-Anycast for building efficient multicast trees. Proceedings of the IEEE Globecom'99 (1999).
58. Katanyutaveetip, D. On Routing Real-time Multicast Traffic. In Proceedings of the IEEE ISCIT 2001 (International Symposium on Communications and Information Technology). (November 2001).
59. Katanyutaveetip, D. Real-time Optimal Multicast Routing. To appear in Computer Communications International Journal, Elsevier Science (2002).
60. Katanyutaveetip, D. Reliable Real-time Multicast Schemes on Internet. Proceedings of the IEEE ISPACS'99 (International Symposium on Intelligent Signal Processing and Communication Systems) (December 1999).
61. Katanyutaveetip, D. Technical Report on Self-healing/Survivable capability in ATM Network. Engineering Journal of Siam University. Vol. 1 (1999) : 97-107
62. Katanyutaveetip, D. Towards reliable real-time Multicast frameworks on the Internet. Engineering Journal of Siam University. Vol. 3 (2000) : 52-60.
63. Kim, S.-B. An Optimal Vp-Based Multicast routing in ATM Networks. In Proceedings of IEEE INFOCOM '96 (1996) : 1302-1309.

64. Koh, S., Shin, M., Yi, J., Hahm, J., and Park, C. Non-core based shared tree architecture for IP multicasting. IEE Electronics Letters Vol. 35, No. 11 (May 1999).
65. Kompella, V., Pasquale, J., and Plozoz, G. Multicasting for Multimedia Applications. In proceedings of IEEE INFOCOM '92 (1992) : 2078-2085.
66. Kompella, V., Pasquale, J., and Plozoz, G. Multicast routing for multimedia communication. IEEE/ACM Transactions on Networking vol. 1, no. 3 (June 1993) : 286-292.
67. Kompella, V., Pasquale, J., and Plozoz, G. Two distributed algorithms for multicasting multimedia information. In proceedings of ICCCN'93 (1993) : 343-349.
68. Kompella, V., Pasquale, J., and Plozoz, G. Two Distributed Algorithms for the Constrained Steiner Tree Problem. In proceedings of the Second International Conference on Computer Communications and Networking (IC³N '93) (1993) : 343-349.
69. Kou, L., Markowsky, G., and Berman, L. A Fast Algorithm for Steiner Trees. Acta Informatica vol. 15, no. 2 (1981) : 141-145.
70. Leung, Y.-W., and Yum, T.-S. Efficient algorithms for multiple destinations routing. In proceedings of IEEE International Conference on Communications (ICC '91) (1991) : 1311-1317.
71. Matta, I., and Guo, L. On routing real-time multicast connections. Proceedings of the IEEE international symposium on computers and communications (1999).
72. Maxemchuk, N. F. Video distribution on multicast networks. IEEE journal on selected areas in communications (April 1997) : 357-372.
73. Menger, K. Zur allgemeinen kurventheorie. Fund. Math. Vol. 10 (1927) : 96-115.

74. Narvaez, P., Siu, K., and Tzeng, H. New dynamic SPT algorithm based on a ball-and-string model. IEEE INFOCOM'99, New York (March 1999).
75. Network Simulator – ns (version 2). [Online]. Available from : <http://www-mash.cs.berkeley.edu/ns/>
76. Ng, J., and Ng, P. Cost-Delay Path Selection Function for Real-Time Multicast Routing. Proceedings of the IEEE International Symposium on Modelling Analysis and Simulation of Computer and Telecommunication Systems (1998).
77. Noronha, C., and Tobagi, F. Optimum Routing of multicast Streams. In proceedings of IEEE INFOCOM '94 (1994) : 865-873.
78. Parris, C. Dynamic Channel Management. PhD thesis, University of California at Berkeley, 1994.
79. Parris, C., and Banerjee, A. An investigation into fault recovery in guaranteed performance service connections. In proceedings of ICC/SUPERCOMM'94, New Orleans, LA (May 1994).
80. Parris, C., and Ferrari, D. The Dynamic Management of Guaranteed Performance Connections in Packet Switched Integrated Service Networks. Technical Report CSD-94-859, University of California, Berkeley, 1994.
81. Parsa, M. Multicast Routing in Computer Networks. PhD thesis, University of California at Santa Cruz, U.S.A., June 1998.
82. Prim, R. Shortest Connection Networks and Some Generalizations. The Bell Systems Technical Journal vol. 36, no. 6 (November 1957) : 1389-1401.
83. Ramanathan, S. An Algorithm for multicast tree generation in networks with asymmetric links. In proceedings of IEEE INFOCOM '96 (1996) : 337-344.
84. Rayward-Smith, V., and Clare, A. On Finding Steiner Vertices. Networks vol. 16 (1986) : 283-294.

85. Rayward-Smith, V. The Computer of Nearly Minimal Steiner Trees in Graphs. International Journal of mathematical Education in Science and Technology vol. 14, no. 1 (January/February 1983) : 15-23.
86. Rouskas, R., and Baldine, I. Multicast routing with end-to-end delay and delay variation constraints. In proceedings of IEEE INFOCOM '96 (1996) : 353-360.
87. Rouskas, R., and Baldine, I. Multicast routing with end-to-end delay and delay variation constraints. IEEE Journal on Selected Areas in Communications (April 1997) : 346-356.
88. Salama, H., Reeves, D., and Viniotis, Y. Evaluation of multicast routing algorithms for Real-time communication on high-speed networks. IEEE J. Select. Areas Commun. Vol. 15 (April 1997) : 332-345.
89. Salama, H., Reeves, D., Viniotis, Y., and Sheu, T.-L. Comparison of Multicast Routing Algorithms for High-Speed Networks. Tech. Pep. TR 29.1930 IBM, September 1994.
90. Shacham, N. Multipoint communication by hierarchically encoded data. In proceedings IEEE INFOCOM'92 (May 1992) : 2107-2114.
91. Shenker, N. Making greed work in networks: A game-theoretic analysis of switch service disciplines. In conference proceedings of ACM Singcomm '94 ACM SIGCOMM. (September 1994).
92. Shields, C., and Garcia-Luna-Aceves, J.J. The ordered core based tree protocol. IEEE INFOCOM (1997).
93. Shim, Y., and Kang, S. An efficient algorithm for multicasting in a large network. Proceedings of the IEEE Asia-Pacific Conference on Computer and Optoelectronics and Communications Conference (1999).
94. Sriram, R., Manimaran, G., and Murthy, C. A rearrangeable algorithm for the construction of delay constrained dynamic multicast trees. IEEE INFOCOM'99, New York (March 1999).

95. Sun, Q., and Langendoerfer, H. Efficient multicast routing for delay-sensitive applications. In proceedings of the Second Workshop on Protocols for Multimedia Systems (PROMS '95) (1995) : 833-838.
96. Takahashi, H., and Matsuyama, A. An Approximate for the Steiner Problem in Graphs. Mathematica Japonica vol. 24, no. 6 (February 1980) : 573-577.
97. Thaler, D., and Ravishankar, C. Distributed center-location algorithms: proposals and comparisons. In proceedings of IEEE INFOCOMM '96 (1996) : 75-84.
98. Thaler, D., and Ravishankar, C. Distributed center-location algorithms. IEEE J-SAC Vol. 15, No. 3 (April 1997).
99. Tode, H., Sakai, Y., Okada, H., and Tezuka, Y. Multicast routing algorithm for nodal load balancing. In proceedings of IEEE INFOCOM '92 (1992) : 2086-2095.
100. Tseng, W., and Kuo, S. Real-time Multicast routing with efficient path selection for multimedia applications. IEICE Transaction on Information and Systems Vol. E84-D, No. 7 (July 2001).
101. Tyan, H., Hou, J., and Wang, B. On providing quality-of-service for core-based multicast routing. Proceedings of the IEEE Distributed Computing Systems Conference (1999).
102. Wall, D. Mechanisms for Broadcast and Selective Broadcast. PhD thesis Stanford University, June 1980.
103. Wall, D. Selective broadcast in packet-switched network. In proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks (February 1982) : 239-258.
104. Wang, B., and Hou, J. Multicast routing and its QoS extension: Problems, algorithms, and protocols. IEEE Network (January 2000) : 22-36.

105. Waters, A. A new heuristic for ATM multicast routing. In proceedings of the Second IFIP Workshop on Performance Modeling and Evaluation of ATM Networks (July 1994) : 8.1-8.9.
106. Waxman, B. Performance evaluation of multipoint routing algorithms. In proceedings of IEEE INFOCOM '93 (1993) : 980-986.
107. Waxman, B. Routing of multipoint connections. IEEE Journal on Selected Areas in Communications vol. 6, no. 9 (December 1988) : 1617-1622.
108. Wei, L., and Estrin, D. Multicast routing in dense and sparse modes: Simulation study of tradeoffs and dynamics. In proceedings of ICCN'95 (September 1995) : 150-157.
109. Wei, L., and Estrin, D. The trade-offs of multicast trees and algorithms. In proceedings of the Third International Conference on Computer Communications and Networking (IC³ N '94) (1994) : 17-24.
110. Wi, S., and Choi, Y. A delay-constrained distributed multicast routing algorithms. In proceedings of the Twelveth International Conference on Computer Communication (ICCC '95) (1995) : 833-838.
111. Widyono, R. The design and evaluation of routing algorithm for real-time channels. Tech. Rep. ICSI TR-94-024 International Computer Science Institute University of California at Berkeley, June 1994.
112. Winter, P. Steiner problem in networks: A survey. Networks vol. 17, no. 2 (1987) : 129-167.
113. Zappala, D. Multicast routing support for real-time applications. PhD thesis, University of Southern California, U.S.A., December 1997.
114. Zhu, Q., Parsa, M., and Garcia-Luna-Aceves, J. A source-based algorithm for delay-constrained minimum-cost multicasting. In proceedings of IEEE INFOCOM '95 (1995) : 377-385.

Appendix



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

APPENDIX A : Journal Acceptance Letter

**computer
communications**

12 Manor Walk, Coventry Road
Market Harborough
Leics LE16 9BP, UK
Tel: 01858 469898
Fax: 01858 431649
Email: comcom@troubador.co.uk
Web: www.elsevier.nl/locate/comcom

23 November 2001

Paper Reference Number: 1979

Dr D Katanyutaveeip
343/11 Jaransanitwong 12
Bangkok 10600
Thailand

Dear Dr Katanyutaveeip

Thank you for sending the revised version of your paper for publication in *Computer Communications* journal:


Real-time optimal multicast routing

Following the revisions that have been made, I am now pleased to be able to accept your paper for publication in the next available issue of the journal.

Your paper will now be included on our web site as having been accepted for publication. The manuscript has already been sent to Elsevier for typesetting, and they will contact you directly if any additional materials are required.

Should you have any queries, please do not hesitate to contact me: (email: comcom@troubador.co.uk). I look forward to hearing from you.

Yours sincerely



Jeremy Thompson
General Editor

INTERNET: www.elsevier.nl/locate/comcom

ELSEVIER SCIENCE

BIOGRAPHY

He received a B.Eng. degree in Computer Engineering from Chulalongkorn University, Bangkok, Thailand, in 1981 and the M.S. degree in Electrical and Computer Engineering from Oregon State University, Corvallis, Oregon, U.S.A. in 1983. He was with Nortel Inc. in Santa Clara, U.S.A. for 8 years, as a hardware engineer, from 1983 to 1991. In 1992, he was employed by Unocal (Thailand) as a Communication Manager, for 4 years. Then, he decided to come back and pursue his Ph.D. study at Chulalongkorn University, majoring in Computer Engineering, in 1999.



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย