On Near Optimal Time and Dynamic Delay and Delay Variation Multicast Algorithms

Meghrig Terzian

A Thesis

In the Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Computer Science) at

Concordia University

Montréal, Québec, Canada

October 2017

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Meghrig Terzian**

Entitled: **On Near Optimal Time and Dynamic Delay and Delay Variation Multicast Algorithms**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. J. Y. Yu

_____ External Examiner
Dr. Muthucumaru Maheswaran

_____ External to Program
Dr. M. Reza Soleymani

_____ Examiner
Dr. Brigitte Jaumard

_____ Examiner
Dr. Dhrubajyoti Goswami

_____ Thesis Supervisor
Dr. Hovhannes A. Harutyunyan

Approved _____
Dr. Volker Haarslev, Graduate Program Director

1/12/2017 _____
Dr. Amir Asif, Dean, Faculty of Engineering and Computer Science

Abstract

On Near Optimal Time and Dynamic Delay and Delay Variation Multicast Algorithms

Meghrig Terzian, Ph.D.

Concordia University, 2018

Multicast is one of the most prevalent communication modes in computer networks. A plethora of systems and applications today rely on multicast communication to disseminate traffic including but not limited to teleconferencing, videoconferencing, stock exchanges, supercomputers, software update distribution, distributed database systems, and gaming.

This dissertation elaborates and addresses key research challenges and problems related to the design and implementation of multicast algorithms. In particular, it investigates the problems of (1) Designing near optimal multicast time algorithms for mesh and torus connected systems and (2) Designing efficient algorithms for Delay and Delay Variation Bounded Multicast (DVBM).

To achieve the first goal, improvements on four tree based multicast algorithms are made: Modified PAIR (MPAIR), Modified DIAG (MDIAG), Modified MIN (MMIN), and Modified DIST (MDIST). The proof that MDIAG generates optimal or optimal plus one multicast time in 2-Dimensional (2D) mesh networks is provided. The hybrid version of MDIAG (HMDIAG) is designed, that gives a 3-additive approximation algorithm on multicast time in 2D torus networks. To make HMDIAG applicable on systems using higher dimensional meshes and tori, it is extended and the proof that it gives a $(2n-1)$-additive approximation algorithm on multicast time in nD torus networks is given.

To address the second goal, Directional Core Selection (DCS) algorithm for core selection and DVBM Tree generation is designed. To further reduce the delay variation of trees generated by DCS, a k-shortest-path based algorithm, Build Lower Variation Tree (BLVT) is designed. To tackle dynamic join/leave requests to the ongoing multicast session, the dynamic version of both algorithms is given that responds to requests by reorganizing the tree and avoiding session disruption. To solve cases where single-core based algorithms fail to construct a DVBM tree, a dynamic three-phase algorithm, Multi-core DVBM Trees (MCDVBMT) is designed, that semi-matches group members to core nodes.

Acknowledgments

I would like to express my most sincere gratitude and acknowledgment to the great people who supported me along the way towards completing my Ph.D. studies. First, I am extremely grateful to my supervisor Prof. Harutyunyan, for his encouragement, guidance, and support. I cannot think of a more inspiring and caring supervisor.

Moreover, I would like to thank my committee members, Prof. Jaumard, Prof. Goswami, and Prof. Soleymani for their time, effort, and willingness to serve on my Ph.D. committee throughout my Ph.D. studies. I would like to also extend my appreciation for the external examiner Pof. Maheswaran for his willingness to read through the thesis and serve on my defense committee.

Finally yet importantly, I would like to thank my family and friends for their unrelenting support. My deepest and endless gratitude goes to my mom for her endless sacrifices. I would also like to thank my sister for always being there for me.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ATabu ..... Adaptive Tabu algorithm

BLVT ..... Build Lower Variation Tree algorithm

BT ..... Broadcast Time

MT ..... Multicast Time

CCNE ..... Candidate Core Nodes Elimination algorithm

CCNG ..... Candidate Core Nodes Generation Satisfying End-to-end Delay algorithm

CF ..... Core Found algorithm

d ..... diagonal node

DCS ..... Directional Core Selection algorithm

DDS ..... Dimensional Distance Sorted algorithm

DDVCA ..... Delay and Delay Variation Constrained Algorithm

DIAG ..... DIAGonal algorithm

DISTance .... DIST algorithm

DP ..... Diagonal Path

DVBM ..... Delay and Delay Variation Bounded Multicast

DVBMT ..... Delay and Delay Variation Bounded Multicast Tree

DVMA ..... Delay Variation Multicast Algorithm

ESC ..... estimation algorithm

FR ..... Forward and Retransmit

HMDIAG .... Hybrid MDIAG algorithm

KMK ..... Kabat et al. Multicast algorithm with K shortest paths algorithm

KMKh ..... Kabat et al. Multicast algorithm with K shortest paths where h= k algorithm

LDP ..... nodes on the Lower side of the Diagonal Path

M ..... multicast group

MCDVBMT . Multi-core DVBMT algorithm

MDIAG ..... Modified DIAG algorithm

MDIST ..... Modified DIST algorithm

M-HCM ..... Multipath Hamiltonian Cycle Model algorithm

MMIN ..... Modified MIN algorithm

MPAIR ..... Modified PAIR algorithm

MST ..... Minimal Steiner Tree

n**D** ..... n-dimensional

OMC ..... Optimal Multicast Cycle

OMP ..... Optimal Multicast Path

OMT ..... Optimal Multicast Tree

PAF ..... Permanent Absorb and Forward

PAFR ..... Permanent Absorb, Forward and Retransmit

PDP ..... Primary Diagonal Path

QoS ..... Quality of Service

RP ..... Rendezvous Point

SC ..... Select Cores algorithm

sd ..... diagonal line from s to d

SDP ..... Secondary Diagonal Path

T ..... multicast Tree

TASNEM .... Tree based Algorithm which Splits torus Networks into two Equally Meshes

UDP ..... nodes on the Upper side of the Diagonal Path

VH ..... Vertical Horizontal algorithm

2D 2-dimensional

Chapter 1

Introduction

1.1 Knowledge Context and Framework of Problem

Data communication in computer networks can be achieved over unicast, broadcast, anycast, and multicast communication. Unicast is communication between a sender and a single receiver node. Broadcast is communication between a sender and all nodes in a network. Anycast is communication between a single sender and the nearest node of a subset of receiver nodes in a network. Multicast is data communication between a sender or a set of senders and a subset of receiver nodes known as multicast group members in a network.

Multicast communication is an important component of the design and implementation of high-speed networks and applications. It is used in many systems and applications including but not limited to teleconferencing, videoconferencing, stock exchanges, distance learning, supercomputers, software update distribution, distributed database systems, and gaming. The performance of these processes and systems depends on the multicast process [41].

Before starting dissemination, multicast routing algorithms construct a multicast tree or path. The performance of the multicast highly depends on the generated tree or path and a tremendous amount of research has been performed to generate trees or paths tailoring specific application requirements. To use network resources more efficiently most approaches for multicast routing disseminate a message along a tree spanning the sender and receiver nodes.

Routing algorithms are classified into four groups, depending on where state information and global topology is available and how routing paths are selected. In source routing, state information and global topology is available at every node and routing paths are selected at the source node. The routing information is added to the message and the global state at every node is updated using a link-state protocol. In distributed routing, state information and global topology is available at every node and routing is achieved hop-by-hop. The destination field of a message carries destination addresses only. Source routing suffers from high message overhead as the addresses of intermediate nodes are added to the message. In distributed routing, the algorithm executed at every node should be simple. To tackle the disadvantages of source and distributed routing, hybrid routing was suggested. In hybrid routing, both the source and intermediate nodes make routing decisions and state information and global topology is available at every node [41]. In hierarchical routing, nodes are divided into groups, which are further divided into higher-level groups. Every node maintains state information. Routing paths including logical nodes representing groups are selected at the source node. At every logical node, source routing is used again to disseminate within a group [12].

A message is divided into packets before transmission, for efficient use of network resources [50]. A packet is the smallest segment of communication that contains routing and sequencing information in its header [50]. Packets are switched using

one of the four switching techniques: circuit switching, store-and-forward, virtual cut-through, and wormhole. In circuit switching, a path between the source and destination nodes is built and reserved. Although path building requires time and reservation of channels throughout message transmission, buffering of data is not required and messages are never blocked [50]. In store-and-forward switching, packets are buffered at every intermediate node before being forwarded to the next node on the path. Channels are only reserved when a packet is being transferred. However, since a packet is buffered at each intermediate node, the time to transmit a packet from the source to the destination is directly proportional to the number of nodes on the path [50]. In virtual cut-through, a packet is only stored at an intermediate node if the next channel required is busy. Consequently, the distance between the source and destination has a slight effect on the transmission latency. However, each node should provide sufficient buffer space for all the messages passing through it. If a packet is blocked at every intermediate node, the virtual cut-through becomes equivalent to store-and-forward switching [50]. In wormhole-routing, a message is divided into flits. A flit is the smallest segment of a message that can be accepted or rejected by a queue or a channel. The header flit holds the routing information and leads the route of flits holding data [46]. In wormhole-routing, the path length does not have a high effect on the transmission latency and small buffers are needed for a channel to hold a flit. However, when the header flit flow is hindered in the network, the flow of trailing flits will not be possible, which can further stop the flow of other messages ending in deadlock [50]. Virtual cut-through switching, buffers blocked messages and removes them from the network, while blocked messages remain in the network in wormhole switching.

The main parameters for evaluating a multicast process are time and traffic, which are negatively correlated [41]. Multicast traffic is the number of links and multicast

time is the number of time units needed to perform the process [41]. The distance between the source and a destination node is the lower bound on the multicast time of every destination node and the number of destinations is the lower bound on the multicast traffic [38].

When the routing method does not allow messages to be replicated at intermediate nodes because it involves high overhead, the multicast communication problem becomes a multicast path problem, where finding an Optimal Multicast Path (OMP) is sought. When replication is not allowed and an acknowledgment should be sent back to the source node once all destination nodes receive the message, the multicast communication problem becomes a multicast cycle problem, where finding an Optimal Multicast Cycle (OMC) is sought. When message replication is allowed and traffic should be minimized, the multicast communication problem becomes a Minimum Steiner Tree (MST) problem. When message replication is allowed and distance sensitive switching is adapted in the network, time will be first minimized then traffic. This makes the multicast communication problem equivalent to an optimal multicast tree (OMT) problem [50], which is proven to be NP-complete in 2D mesh networks [41]. Consequently, many heuristics were proposed to solve the problem that minimizes one parameter first and then tries to reduce the cost of the other.

The torus topology is one of the most prevalent interconnection topologies used in several of the most powerful supercomputers. On the latest list of the TOP500 [60], the fourth fastest supercomputer *Titan*, a Cray XK7 system, uses Gemini interconnect with an underlying 3D torus interconnection [5, 61]. The fifth entry on the list, IBM BlueGene/Q *Sequoia*, uses a 5D torus topology [69].

Path based [17, 64, 65] and tree based [19, 26, 28] algorithms are the two classes of existing multicast algorithms for nD torus networks. Tree based multicast has high

efficiency on time and traffic [38]. The message from the originator is sent to multiple neighbors, which in turn send the message to nodes at a further distance [46]. The parallel way the message is disseminated leads to time efficiency and nodes sharing as much common path as possible leads to traffic efficiency. Path based multicast generates high traffic because a single path is followed to reach all destination nodes and might lead to network congestion. It also generates high latency, because the path lengths are generally long.

In the first part of this dissertation, source based and hybrid multicast tree algorithms suitable for supercomputer system requirements are designed. More specifically, near optimal time multicast algorithms, as time efficiency is crucial for these systems.

Quality of Service (QoS) multicast communication requirements of an application are given as a set of constraints, C. C_i can be a constraint on individual links or the entire communication. A link constraint specifies a QoS requirement on paths. Whereas an entire communication constraint specifies a QoS requirement on the communication tree. These constraints include but are not limited to end-to-end delay, inter-destination delay variation, total bandwidth, buffer utilization, traffic concentration, and minimum residual bandwidth. Some constraints or a combination of constraints, often make the routing problem intractable. A QoS multicast routing algorithm might fail to generate a session satisfying all the QoS constraints of an application because a solution does not exist, or because the search space of the algorithm is not efficient enough. Consequently, the connection request is rejected or the application is negotiated for looser QoS constraints.

Some applications require a multicast communication with minimal use of network resources, represented by an optimization objective. If the optimization function is the sum of the costs of the links of the multicast tree, it is the Steiner tree problem which is

proven to be NP-complete [23]. Other applications require minimizing of an objective function under QoS constraints. Algorithms have been proposed that minimize the inter-destination delay variation under end-to-end delay [8, 13, 30, 32, 35, 53, 54, 56]. Others, minimize the cost of the tree under delay constraints [72]. Another group of algorithms have been proposed that minimize the cost under end-to-end delay and delay variation constraints [7, 32, 36, 37].

Trees constructed by multicast routing protocols fall into two categories: single-source shortest-path and shared core-based. In single-source shortest-path trees, a separate tree is built for each source using least-cost paths between the source node and multicast group members. In shared core-based trees, one tree is built for the entire group and is shared among all senders. A node is chosen as the core or Rendezvous Point (RP), and a shortest-path tree is built rooted at the core node to the multicast group members. The source nodes send a unicast message to the core node, which is responsible for disseminating the message to multicast group members. The position of the core affects the performance of multicast and selecting an optimal core node is an NP-complete problem [62]. Research on core-based routing has focused on core selection [59], multicast tree construction, membership dynamics, and core migration [42]. Several approaches have been suggested for core selection that fall into three categories: random, topology based, and group based including different simple neighborhood search heuristic algorithms [25, 49] or local and global search fitness function minimization algorithms like Tabu, GRASP and VNS [6, 7, 31, 63].

In the second part of the thesis, QoS core-based tree multicast algorithms for high-speed network real-time applications are designed. More specifically, applications that require a message to be sent to group members within a certain bound on delay and delay variation.

1.2 Thesis Contributions

The main contributions of this thesis can be summarized as follows:

- Improvements on four tree based multicast algorithms in 2D mesh and torus networks are made: MDIAG, MPAIR, MMIN, and MDIST. The proof that MDIAG generates optimal or optimal plus one multicast time in 2D mesh networks is provided. Bounds on the time of the modified algorithms in 2D mesh and torus networks is given. Extensive simulations of these algorithms show that they perform better than existing ones, MDIAG is suitable for pro-time applications, and MPAIR is suitable for pro-traffic applications.

- To tackle the disadvantages of centralized routing algorithms, the hybrid version of MDIAG, HMDIAG is designed. HMDIAG performs preprocessing at the source node. At the source node and every intermediate node, another process is performed to retransmit the message to another subset of destination nodes. HMDIAG generates optimal or optimal plus one time in 2D meshes and it is a 3-additive approximation for multicast time in 2D torus networks. To make HMDIAG applicable on systems using higher dimensional tori, it is extended. HMDIAG gives a $(2n-1)$-additive approximation algorithm on multicast time in nD torus networks. Simulation results show that HMDIAG generates less multicast time, latency, and coefficient variation of multicast time than existing algorithms.

- DCS algorithm is designed that uses a novel directional approach to select a core node and generate a Delay and Delay Variation Bounded Multicast Tree (DVBMT).

- To further decrease the inter-destination delay variation of the trees generated

by DCS, Build Lower Variation Tree (BLVT) algorithm based on k-shortest-paths is designed. In DCS and DCSBLVT, the source sends a unicast message to the core node, and the core sends the message to group members using the multicast tree. Thus, the end-to-end delay and inter-destination delay variation values set during the execution of the algorithms reflect the values of the generated tree. DCS and DCSBLVT surpass existing algorithms in efficiency, end-to-end delay, inter destination delay variation, and execution time.

- Moreover, the dynamic version of DCS and DCSBLVT is given, that respond to dynamic join and leave requests to the ongoing multicast session by reorganizing the tree and avoiding session disruption. On average, only 3.4% of the total requests in DCS triggered re-executions and 2.8% in DCSBLVT.

- To address cases where single-core based algorithms fail to generate a tree satisfying delay and delay variation constraints, Multi-Core DVBMT (MCDVBMT) algorithm is designed. When existing single-core based algorithms fail to generate a tree satisfying delay and delay variation constraints, MCDVBMT successfully selects multiple cores and generates trees rooted at the selected cores satisfying both constraints. MCDVBMT generates less inter-destination delay variation and traffic concentration than existing single-core algorithms. In addition, in MCDVBMT only group members receiving the message from the failing core node suffer from recovery delay when a core node fails. However, the end-to-end delay and cost of MCDVBMT is higher than single-core trees.

- The dynamic version of MCDVBMT is designed. On average, only 5.2% of the requests triggered re-executions.

1.3 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 presents a review of current state of knowledge and the motivation for this research; where Section 2.1 is dedicated for multicast in supercomputer systems and Section 2.2 is dedicated for delay and delay variation multicast algorithms. Chapter 3 focuses on near optimal time multicast algorithms in 2D mesh and torus networks; where Section 3.1 defines the system model and specifies the problem, Section 3.2 presents the first contribution, the four modified algorithms MPAIR, MDIAG, MMIN, and MDIST in mesh and torus networks with their time complexities and bounds on time, and Section 3.5 gives the second contribution, HMDIAG, with its time complexity and proof of giving a 3-additive and $2(n-1)$-additive approximation algorithm for multicast time in 2D torus and nD torus networks, respectively. Chapter 4 focuses on DVBM algorithms; where Section 4.1 defines the network model and specifies the problem, Section 4.2 presents the third contribution, DCS algorithm with its time complexity, Section 4.3 presents the fourth contribution BLVT algorithm with its time complexity, Section 4.4 presents the fifth contribution, the dynamic version of DCS and DCSBLVT to tackle dynamic changes to the multicast group, Section 4.5 presents the sixth contribution, MCDVBMT with its time complexity, and Section 4.6 presents the seventh contribution, dynamic MCDVBMT. Chapter 5 is dedicated for numerical results and experiments performed to compare the proposed algorithms with existing ones. Chapter 6 provides a conclusion to this manuscript and future directions for this research. Throughout this dissertation, figures, tables, and algorithms are enumerated relatively to each chapter.

Chapter 2

Literature Review and Motivation

2.1 On Multicast in Mesh and Torus Networks

Direct network topologies, like meshes and tori, are the most commonly used network topologies in supercomputer systems since it is easy to design and cheap to implement multicast on them because of their geometric regularity [50].

A mesh has a geometrically regular structure. Computation of connectivity, distance and routing is very simple. Connectivity and distance are determined by comparing coordinates of the nodes. Routing is choosing a direction at each node, which reduces the complexity of the algorithm significantly by finding shortest paths easily. A mesh network is reliable and fault tolerant, the network would not fail when a node or several links fail. In addition, adding nodes increases the total communication bandwidth, memory bandwidth, and processing capability of the system [66]. However, a communication between the first and last node of a dimension passes through a long path because of the absence of a direct edge between them. If this direct edge is added, the structure becomes symmetric and the distance between nodes reduces roughly by half given that the message can travel from two opposite directions to reach the same destination. Such a structure is known as a torus [50].

The symmetric nature of a torus balances the traffic load. In a mesh, assuming uniform traffic between nodes, links closer to the center will be more loaded than links closer to the borders [66]. However, it is easier to provide deadlock free routing in a mesh than in a torus [50].

Messages divided into packets can be switched using one of the four available switching techniques: circuit switching, store-and-forward, virtual cut-through, and wormhole. The difference between these switching techniques is discussed in the Chapter 1. Deadlock prevention is one of the main issues in wormhole switching and many approaches have been proposed to tackle it [15, 24, 24, 40, 43, 47, 50].

Early multicast communication approaches were based on software, to reduce the delay and overhead they started depending on hardware [38]. All first generation hypercube multiprocessors supported unicast and a few second generation hypercube multiprocessors supported broadcast [38]. Lan et al. presented the key features of a VLSI router to support all three types of communication and proposed an optimal algorithm for generating an OMT when the multicast set is small. They also proposed a heuristic greedy multicast algorithm based on distributed routing of messages in hypercube multiprocessors when the multicast set size is not small. The time complexity of the algorithm is $O(qn + n^2)$, where q is the number of destinations and n is the dimension of the hypercube. The proposed algorithm guarantees shortest paths between the source and each destination. The traffic generated is very close to the optimal solution, and is optimal when the number of destination nodes is less than four [38].

Since first generation supercomputers implemented distance sensitive routing technologies, the network topology widely used was the hypercube because of its dense interconnection that resulted in shorter message paths. When virtual cut-through and wormhole routing were proposed, hypercubes were replaced by low-dimensional

meshes and tori.

Lin et al. proposed two deadlock-free path based multicast algorithms for wormhole routed networks and a routing function for 2D mesh and hypercube topologies. After showing that tree based routing is not suitable for wormhole routing, they proposed a multicast star model. In this model, destination nodes are partitioned into several disjoint subsets and the message is sent along several multicast paths, one for each subset of destination nodes. Subsets are created after labeling nodes based on their order in a Hamiltonian path, such that in the path any two given nodes are connected by a shortest path. The dual-path algorithm creates two subsets, while the multi-path can create up to c paths, where c is the number of outgoing channels of the sender. The path selected by the routing function R for any two random nodes u and v is a shortest path from u to v and is a partial order preserving the label assignment function. The complexity of the message preparation part of the algorithms are $O(q \log q)$ and $\max\{O(q \log q), O(cq)\}$ respectively, where q is the multicast set size. At every node, it takes $O(c)$ time to make the routing decision. Although the multi-path routing requires less number of channels than the dual-path routing, each path must buffer the whole message at the source node and it is not released until the last flit is sent. Moreover, if multiple messages are transmitted, the link between the local processor and router may become a bottleneck [40].

Lin et al. after proving that OMP, OMC, and MST problems are NP-Complete for 2D mesh and n-cube topologies, they proposed a hybrid heuristic algorithm for OMP. In the message preparation part of the algorithm, the source node adds a routing control field including the destination addresses and some routing information. The routing information includes destination nodes sorted according to their position in the Hamiltonian path or cycle. In the second part of the algorithm, message routing, each forward node including the source node selects a next node that is the

closest in the cycle to the next destination. The time complexity of the first part of the algorithm is $O(q \log q)$ where q is the number of destination nodes. The time complexity of the second part of the algorithm is $O(1)$ in a 2D mesh and $O(n)$ in an n-cube [41].

Mckinely et al. (1994) proposed a minimum-time multicast algorithm for nD meshes that uses dimension-ordered and restricted routing of unicast messages. In dimension ordered routing, routing is performed on each dimension monotonically. In restricted routing, channel selection is constrained. The algorithm (U-mesh) is contention free and can deliver a multicast message to $m-1$ destinations in $\lceil \log m \rceil$ steps in one port architectures. After sorting the multicast set into a dimension ordered set (M'), the source node divides M' into two equal sets, lower and upper half. If the source is in the lower half, it sends a copy of the message to the lowest node in the upper half, which in turn is responsible for sending the message to the group members in that half. If the source is in the upper half, it sends a copy of the message to the highest node in the lower half. At each step, the selected node and the nodes assigned to it are removed from M'. The procedure continues until M' has only the source address [46]. In unicast based multicast, multiple copies of the same message is sent leading to high network traffic.

A hardware path based multicast routing algorithm, TPM, for 2D meshes was proposed in [48]. TPM outperforms U-mesh. It divides the mesh into up two four submeshes and sends at most four messages. TPM uses two startup times.

Applying the U-mesh algorithm in a torus does not result in a contention free multicast. Consequently, Robinson et al. extended the idea of the U-mesh algorithm and proposed the U-torus algorithm that is a minimum-time multicast algorithm for nD torus networks that uses dimension-ordered and restricted routing of unicast messages. The algorithm is contention free and can deliver a multicast message to

m - 1 destinations in $\lceil \log m \rceil$ steps under UTR or BTR. UTR and BTR are the two routing algorithms they propose for Unidirectional and Bidirectional torus networks, respectively adapting the virtual channels approach. The algorithm first sorts the multicast set into a dimension ordered set (M'), then rotates the order such that source node is at the head of the chain called R-chain. After, the algorithm performs the same steps as the U-Mesh algorithm. The complexity of the algorithm is $O(q \log q)$ where q is the size of the multicast set [10]. Multiple copies of the same message is sent in U-mesh algorithm, leading to high network traffic.

The critical issues in virtual channel implementation are multiplexing and scheduling. Sharing of bandwidth results in an increase in network latency. Moreover, an increase in the number of virtual channel, makes scheduling more complicated [50]. Virtual channels are implemented with a flow control protocol, that determines how resources like buffers and channel bandwidth are allocated and how message collisions are resolved. A message collision occurs when a packet cannot proceed because the buffer it needs is occupied by another message [47]. A comprehensive survey of algorithms using virtual channels is provided by Mohapatra (1998) [47].

The dual-path and multi-path algorithms emphasize on reducing the number of startup latencies, and do not perform well in the presence of high traffic loads, since they generate long paths. To overcome this limitation, Boppana et al. proposed the column-path path based multicast routing algorithm which is based on dimension ordered routing and focuses on using shorter paths rather than less number of paths [10]. The algorithm divides the destination nodes into at most $2k$ subsets, where k is the number of columns in the mesh, such that at most two messages are directed to each column. If a column has one or more destination nodes in rows above the source node row, one copy of the message is sent to serve all those destinations. Similarly, if a column has one or more destinations in the rows below the source node row, one copy

of the message is sent to serve all those destinations [3]. The column-path multicast algorithm uses short paths and performs better in the presence of high traffic loads. The drawback in this approach is the high number of startups, one for each path [44].

In the time optimal, centralized, and tree based Vertical Horizontal (VH) algorithm for mesh and torus networks, the message is routed to each destination node in dimensional order. The algorithm delivers the message to each destination along a shortest path, achieves optimal multicast time, and has $O(qD)$ time complexity where q is the number of destinations and D is the diagonal of the mesh or torus. However, it generates high traffic because of the followed dimensional order. If VH routes in horizontal dimension first, the top row becomes a major path and if most of destinations are located at the bottom rows of the mesh, the message will be routed from the top row to the bottom only to one or two destinations [4].

The non-mimimal, centralized, and tree based DISTance (DIST) algorithm, sorts destination nodes in ascending order of distances from the source node. The multicast tree is constructed by adding the sorted nodes to the multicast tree through a shortest path to a node on the hitherto created tree (Algorithm 2.1) [44]. XY routing is a deadlock free, dimension ordered, and minimal routing algorithm in 2D meshes. Packets are sent first in the X dimension and then in the Y dimension. In nD meshes, routing is completed in one dimension before proceeding to the next dimension. The time complexity of the algorithm is $O(qDN)$ where q is the number of destination nodes, D is the diagonal of the mesh or torus, and N is the total number of nodes in the network.

The centralized, minimal, and tree based DIAGonal (DIAG) algorithm was proposed to reduce the traffic generated by the VH algorithm. The main concept in DIAG is setting the major path, the diagonal of the multicast zone. The algorithm first finds the multicast zone and the diagonal node d. Computes the diagonal line

Algorithm 2.1 DIST

```
1: function DIST(M, s)
2:     Sort destination nodes in increasing order of distances from s
3:     T ← T ∪ s
4:     repeat
5:         u ← first node in M
6:         Find a closest node v in T to u
7:         Add node u to T through a shortest path from v by XY routing
8:         M ← M − {u}
9:     until M = ∅
10:    return T
11: end function
```

(sd) between s and d, and adds it to T. Next, destination nodes are sorted in increasing order of distance from the source node and added to T though a closest node v in T within the zone $\{s \Leftrightarrow u\}$ (Algorithm 2.2). The algorithm takes $O(qN)$ time in a 2D mesh and torus, where q is the total number of destinations and N is the total number of nodes in the network. DIAG does not perform well if destinations are located at a far distance from the diagonal of the mesh [4].

DDS (Dimensional Distance Sorted) is a centralized, minimal, and tree based multicast routing algorithm for mesh and torus networks. The algorithm sorts the destinations in dimensional distance, which is the distance along one dimension between them. For example, for any two nodes $x(x_1, \cdots, x_i, \cdots, x_n)$ and $y(y_1, ..., y_i, ..., y_n)$, the ith dimensional distance $d_i = |x_i - y_i|$. The minimum dimensional distance between x and y denoted as $d_{min} = \min\{d_1, \cdots, d_i, \cdots, d_n\}$. Destination nodes (d_i) are added to T through the closest node v in the T within the zone $\{s \Leftrightarrow d_i\}$. The time complexity of DDS in a 2D mesh or torus is $O(qN)$, where q is the total number of destinations and N is the total number of nodes in the network [66].

The XY centralized and path based multicast algorithm was proposed by Wang et al. (2005) which increases the parallelism of message passing by increasing the number of paths through which messages can be passed concurrently to two. It reduces the

Algorithm 2.2 DIAG

```
1: function CreateDiagonalPath(M, s)
2:     DP ← s
3:     u ← s
4:     x_d ← Max {x_{d_i}} ∀i | 1 ≤ i ≤ q
5:     y_d ← Max {y_{d_i}} ∀i | 1 ≤ i ≤ q
6:     repeat
7:         x_{u'} ← x_u + 1, y_{u'} ← y_u, x_{u''} ← x_u, y_{u''} ← y_u + 1
8:         if D(u', sd) ≤ D(u'', sd) then
9:             u ← u'
10:        else
11:            u ← u''
12:    until u = d
13:    return DP
14: end function
15: function CreateT(M, s, DP)
16:    Sort destination nodes in increasing order of distances from s
17:    T ← DP
18:    repeat
19:        u ← first node in M
20:        Find a node v in T in the zone {s ⇔ u} that is the closest to u
21:        Add node u to T through a shortest path to v by XY routing
22:        M ← M − {u}
23:    until M = ∅
24:    Cut the tail part of the DP that does not have destination or replicate nodes
25:    return T
26: end function
```

total traffic by reducing the back and forth traveling distance of the base path and by passing a message through a shortest path. In a 2D mesh two message-passing paths are possible, one travels along the X dimension and the other along the Y dimension and together they cover all the destination nodes in the mesh. After splitting nodes into two subsets joined at the source node, and forming a sub-Hamiltonian path for each subset, the message in X path and Y path are routed concurrently from one destination to another in the order as they occur in X path or Y path. The complexity is $O(N)$ where N is the number of nodes in the mesh [66].

Al-Dubai et al. proposed a path based multicast algorithm, Qualified Groups

(QG). They take into consideration the multicast latency at network and node levels. The QG algorithm has four phases. In phase one, the multicast area (GMA) is defined which is the area in the mesh that includes the multicast set. In phase two, the mesh is divided into submeshes. In phase three, the submeshes obtained from the previous phase are tested and further divided from 2 to 2^n subgroups. In phase four, for each qualified group the node closest to the source node is selected as a representative node to receive the multicast message from the source node. Each representative node acts as a source node in the group and sends the message to the destination nodes in the group. The QG algorithm uses the deadlock free e-cube routing algorithm [3].

PAIR is another centralized, minimal, and tree based algorithm designed to reduce the traffic of the tree based DIAG and VH algorithms. Given the q destination nodes, destination nodes are arranged as $\{(x_1, y_1), (x_2, y_2), \cdots, (x_q, y_q)\}$, where $x_1 \leq x_3 \leq x_5 \leq \cdots \leq x_k$ and $y_2 \leq y_4 \leq y_6 \leq \cdots \leq y_q$. The algorithm first selects a pair of nodes u and v from the destination set with minimum x and y values denoted as x_{min} and y_{min} and then constructs a multicast tree from the source to the intermediate node (x_{min}, y_{min}). Next, it connects nodes u and v to (x_{min}, y_{min}) through a shortest path. This process is repeated until every destination node is added to the multicast tree (Algorithm 2.3) [45]. The complexity of the algorithm in a 2D mesh and torus is $O(qD)$ where q is the number of destination nodes and D is the diagonal of the mesh.

MIN was designed to further reduce the multicast traffic of PAIR and obtain a near optimal time. MIN is a centralized, non-minimal, and tree based algorithm that focuses on reaching destination nodes through a shortest path from the existing multicast tree. In MIN, instead of generating many intermediate nodes, only one intermediate node is generated, and the rest of the destinations are connected to the existing multicast tree in the same order as PAIR without an intermediate node (Algorithm 2.4) [45]. The time complexity of MIN in a 2D mesh or torus is $O(qDN)$,

Algorithm 2.3 PAIR

```
1: function PAIR(M, s)
2:     T ← T ∪ s
3:     Arrange destination nodes in M
4:     repeat
5:         Select destination nodes u and v, where u has the minimum x value x_min
           in M and v has the minimum y value y_min in M
6:         Find a node w in T in the zone {s ⇔ (x_min, y_min)} that is the closest to
           (x_min, y_min)
7:         Add node (x_min, y_min) to T through a shortest path to w by XY routing
8:         Add u to T through a shortest path to (x_min, y_min) by XY routing
9:         Add v to T through a shortest path to (x_min, y_min) by XY routing
10:        M ← M − {u, v}
11:    until M = ∅
12:    Cut the end parts of T not having d_i or replicate node
13:    return T
14: end function
```

where q is the number of destination nodes, D is the diameter of the mesh, and N is the number of nodes in the network.

VH, DIAG, DDS, PAIR and MIN in torus networks are applied using the generic algorithm presented in Section 3.2.5.

TDP, a deadlock free Hamiltonian path based multicast algorithm for 2D torus networks, divides the torus into two equal meshes according to the location of the source node. One mesh contains the nodes with y-coordinate value less than that of the source node or greater than that of the source node minus $\frac{m}{2}$. The second mesh contains the remaining nodes. Destination nodes in each mesh receive the message in a Hamiltonian path based order. Consequently, in TDP the path length becomes a dominant factor, leading to high latency [1].

Two Hamiltonian path based multicast routing algorithms for 2D torus networks, uniform and fixed routing, were proposed. The algorithms differ in their message preparation part. In uniform routing, destination nodes are divided into two groups having almost equal number of destination nodes. In fixed routing, the routing paths

Algorithm 2.4 MIN

```
1: procedure MIN(M, s)
2:     T ← T ∪ s
3:     Arrange destination nodes in M
4:     Select destination nodes u and v, where u has the minimum x value x_min
       in M and v has the minimum y value y_min in M
5:     Add node (x_min, y_min) to T through a shortest path to S by XY routing
6:     Add u to T through a shortest path to (x_min, y_min) by XY routing
7:     Add v to T through a shortest path to (x_min, y_min) by XY routing
8:     M ← M − {u, v}
9:     repeat
10:        Select a node f that has the minimum x value x_min in M
11:        Find a node c_1 in T that is the closest to f
12:        Add node f to T through a shortest path to c_1 by XY routing
13:        Select a node g that has the minimum y value y_min in M
14:        Find a node c_2 in T that is the closest to g
15:        Add node g to T through a shortest path to c_2 by XY routing
16:        M ← M − {f, g}
17:    until M = ∅
18: end procedure
```

have a maximum length restriction. The proposed algorithms have similar behavior in 2D torus networks and use two startup times [65].

GTTPM and TTPM are two path based multicast routing algorithms for 2D torus networks that divide the 2D torus into two meshes. TTPM uses the vertical wraparound links while GTTPM uses the horizontal ones. GTTPM resolves the issue of TTPM not being able to include all destination nodes when the x-dimension is larger than the y-dimension of the torus network. Both algorithms use two startup times. At startup time one, the message is sent to a set of nodes through a Main Path (MP) in a way that all destination nodes can be reached in the second phase of communication. At startup time two, nodes on the MP send the message to the remaining destination nodes [16, 17].

Multipath Hamiltonian Cycle Model (M-HCM) is a path based algorithm based on the Hamiltonian cycle model dividing the network into two subnetworks, after

labeling nodes according to their position on a Hamiltonian cycle starting from the source. The high-channel network contains directional common links with nodes labeled from low to high and directional wraparound links with nodes labeled from high to low. The low-channel network contains directional common links with nodes labeled from high to low and directional wraparound links with nodes labeled from low to high. Destination nodes are partitioned into 2^n subsets and two messages are sent to each subnetwork. The time complexity of the message preparation part of M-HCM is $O(q \log q)$ and at every node involved in the multicast process constant time is spent. M-HCM utilizes one startup time [64].

Another Hamiltonian path based a multicast routing algorithm, RG, for 2D torus networks was proposed. RG divides the 2D torus into disjoint subnetworks and the destination nodes into several groups. It uses two startup times. At startup time one, for each group the nearest destination node to the source node is selected to become the leader of that group. Next, the message is sent from the source node to the leaders. At startup time two, the leaders retransmit the message to all remaining destination nodes in their own groups [18].

Tree based Algorithm which Splits torus Networks into two Equally Meshes (TASNEM), is a tree based algorithm for 2D torus networks dividing the torus into two equal meshes. One mesh has the nodes with y-coordinate value between that of the source node and source node $\pm \frac{m}{2}$. The other mesh has the remaining nodes. In each mesh, the message is sent along a main path and many branching horizontal paths. The time complexity of both the message preparation part and time spent at every node involved in the multicast process in TASNEM is $O(q)$. TASNEM utilizes two startup times [19].

Tree based multicast has high efficiency on both time and traffic [38]. The message from the source node is sent to multiple neighbors that in turn send the message

to other nodes located at a further distance from the source node [46]. The time efficiency is due to the high degree of parallelism in the message distribution and the traffic efficiency is due to destination nodes sharing as much common path as possible. Path based multicast algorithms generate high traffic, since they follow a certain order along a single Hamiltonian path and as paths get longer multicast time becomes higher [19]. Most of the above-mentioned algorithms are path based or rely on unicast, except TASNEM, DIAG, PAIR, DIST, MIN, DDS, and VH.

In this dissertation, improvements on four tree based multicast algorithms in 2D mesh and torus networks are made: MDIAG, MPAIR, MMIN, and MDIST. The proof that MDIAG generates optimal or optimal plus one multicast time in 2D mesh networks is provided. Bounds on the time of the modified algorithms in 2D mesh and torus networks is given. Extensive simulations of these algorithms show that they perform better than existing ones, MDIAG is suitable for pro-time applications, and MPAIR is suitable for pro-traffic applications. To tackle the disadvantages of centralized routing algorithms, the hybrid version of MDIAG, HMDIAG is designed. HMDIAG performs preprocessing at the source node. At the source node and every intermediate node, another process is performed to retransmit the message to another subset of destination nodes. HMDIAG generates optimal or optimal plus one time in 2D meshes and it is a 3-additive approximation for multicast time in 2D torus networks. To make HMDIAG applicable on systems using higher dimensional mesh and tori, it is extended. HMDIAG gives a $(2n-1)$-additive approximation algorithm on multicast time in nD torus networks. Simulation results show that HMDIAG generates less multicast time, latency, and coefficient variation of multicast time than existing algorithms.

2.2 On Delay and Delay Variation Multicast

Multicast communication is deployed in many applications requiring efficient and fair delivery of messages to recipients. If the inter-destination delay variation between replicated data in a distributed database system is high during an update, unfairness, inconsistencies, and incorrect computations occur. Updates should also be propagated within an upper bound on delay, for instant effect. Delay and Delay Variation Bounded Multicast (DVBM) imposes certain Quality of Service (QoS) conditions on message dissemination, including low end-to-end delay, inter-destination delay variation, and error probability.

Rouskas and Baldine defined the DVBM Tree problem (DVBMT) and proved it to be NP-complete [53]. DVBMT has been studied in various networks [2, 8] and many methods have been suggested to tackle it [8, 13, 30, 32, 35, 53, 54, 56].

Buffering at the source node, intermediate nodes, or destination nodes may be used to achieve minimum delay variation. However, each approach introduces an additional load on the network. When the source node buffers messages, it maintains additional information about all destinations and sends multiple copies of the message at different times. When selected intermediate nodes buffer messages, multiple copies of the same message will be sent at different times. When destination nodes buffer multicast messages before passing them to the user, end-users may use the information in the messages to compete against each other. Furthermore, the amount of buffering needed is proportional to the maximum variation of end-to-end delays. If the variation is smaller buffering would be used more efficiently. Thus, buffering at the receivers may be used along with DVBM algorithms.

Rouskas and Baldine after defining the DVBMT problem they proposed the Delay Variation Multicast Algorithm (DVMA) [53]. DVMA generates a shortest path tree,

T, rooted at s to all nodes in M. If T violates the maximum end-to-end delay value (Δ), a valid tree does not exist. If T satisfies both constraints, it is a valid tree for the multicast session. If T satisfies only the Δ constraint, the longest path group member d_i in T is selected and k-shortest-paths satisfying the Δ constraint from s to d_i are generated. After sorting the paths in increasing order of delay, a tree is gradually constructed by starting with the shortest path of the k paths. The rest of the group members are appended gradually, through paths satisfying both constraints to a node on the hitherto created tree. This is achieved by generating l-shortest-paths within the Δ constraint from every node on the hitherto created tree to the group member to be added, and then selecting the lowest inter-destination delay variation generating path of the l paths. If all group members are added to the tree and the maximum inter-destination delay value (σ) is satisfied, a solution is found. However, if all k-shortest-paths are used and a tree satisfying the σ constraint is not generated, the tree with the lowest inter-destination delay variation is returned. The time complexity of the algorithm is $O(klmn^4)$ [53].

The location of a core node, v_c, affects the end-to-end delay and inter-destination delay variation of the trees generated for core-based DVBMT in three cases. Let v_i be a multicast group member. A Dotted link between two nodes indicates the presence of relay nodes.

Case (1): v_i is on the minimum delay path from s to v_c (Figure 2.1(a)).

Case (2): The minimum delay path from v_i to v_c and s to v_c share common links (Figure 2.1(b)).

Case (3): s is on the minimum delay path from v_c to v_i (Figure 2.1(c)).

A discussion on existing algorithms considering or failing to consider these cases is presented below.

A core based tree method to tackle the DVBMT problem was first suggested by

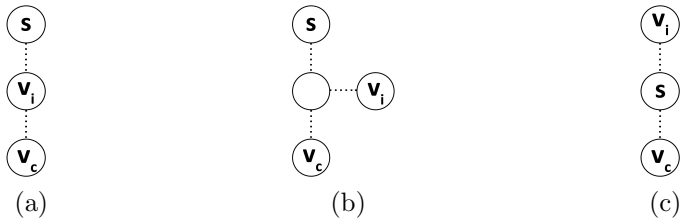


Figure 2.1: Cases

Sheu and Chen. The proposed Delay and Delay Variation Constrained Algorithm (DDVCA), finds shortest paths from the source to all nodes in the network and from every node in M to all nodes. After setting the inter-destination delay variation of every node satisfying the Δ constraint to reach all members, the node with the minimum inter-destination delay variation value is selected as the core node. A group member in a tree generated by DVBMT receives the message from the source, if its location conforms with case (1) (Figure 2.1(a)). The time complexity of the algorithm is $O(mn^2)$ [56].

The algorithm Kim et al. propose, estimation (ESC), finds shortest paths from the source to all nodes in the network and from every node in M to all nodes in the network. After setting the inter-destination delay variation of every node satisfying the Δ constraint to reach all members, the minimum inter-destination delay variation value (dv_{min}) is selected. If the dv_{min} value holding node is not unique, the potential variation of these nodes is calculated. The potential variation calculation function considers if the location of the nodes conforms with cases (1) or (3). The node with the minimum potential variation value is selected as the core node. A group member in a tree generated by ESC receives the message from the source, if its location conforms with case (1). The time complexity of the algorithm is $O(mn^2)$ [35].

Kabat et al. propose another core-based algorithm for DVBMT, Kabat et al. Multicast algorithm with K shortest paths (KMK) [30]. The algorithm also takes into consideration if the location of the nodes conforms with cases (1) or (3). The

algorithm finds shortest paths from the source to all nodes in the network and from every node in M to all nodes in the network. Next, the inter-destination delay variation of every node is set and the minimum inter-destination delay variation value (dv_{min}) is selected. To prevent the occurrence of case (3) in the generated trees, all candidate core nodes that have s on their shortest paths to a group member are marked as non-candidate nodes. A value $p = dv_{min} + stddev(dv)/2$, where $stddev$ is the standard deviation of the delay variations of candidate core nodes, is introduced. Nodes that do not satisfy the Δ constraint and have higher inter-destination delay variation than p are marked as non-candidate nodes. Next, pass and compare values are set for candidate core nodes reflecting the maximum distance of all group members from the candidate core node if case (1) occurs. The node with the minimum compare value is selected as the core node, v_c. If the lowest compare value node is not unique, the node with the lowest inter-destination delay variation is selected as v_c. In the second part of the algorithm, a k-shortest-path based approach is proposed to further decrease the inter-destination delay variation of the generated tree. k-shortest-paths constrained by $\Delta - \max_{v_i \in M} D(v_c, v_i)$ delay value are selected from s to v_c. Group members are connected to v_c through shortest paths. s is connected to v_c through the best of the k paths and cycles are removed. The algorithm including both parts is KMKh where h = k. The time complexity of KMK and KMKh are $O(mn^2)$ and $O(n^3)$, respectively [30].

Sahoo et al. propose an Adaptive Tabu (ATabu) search based algorithm for core selection. The algorithm finds shortest paths from the source to all nodes in the network and from every node in M to all nodes in the network. After setting the inter-destination delay variation of every node, the node (*current*) with the minimum inter-destination delay variation is selected as the starting point of the search. Neighbors within the search radius of *current* are evaluated and the node (*best*) with the lowest

fitness value is selected. If *best* has lower fitness than *current*, the next search starts from node *best*. If *best* does not have lower fitness than *current*, local minima is reached. The Tabu list is updated to include the high quality nodes. Entrapment in local optima is avoided by reselecting one of the previous high quality solutions from the Tabu list as the starting point of the next search, when the maximum allowed number for solution cycling is reached. The search process of adaptive Tabu is faster than the standard Tabu search as the search radius is gradually decreased with fitness function decrease. The time complexity of the algorithm is $O(n^3)$ [54].

In the next couple of paragraphs, the weaknesses of existing core-based DVBMT algorithms are listed and the motivation for proposing DCS and DCSBLVT is stated.

When the location of a group member in trees constructed by DDVCA conforms with case (1), it receives the message from the source node [56]. This leads to a different end-to-end delay and inter-destination delay variation values than the ones set during the execution of the algorithm. Moreover, DDVCA models the problem with an undirected graph that does not reflect realistic networks.

In ESC, the estimation value is set for a small subset of candidate core nodes, the ones having the lowest inter-destination delay variation value. When the location of all these considered candidate nodes in trees constructed by ESC conforms with cases (1) or (3), the end-to-end delay and inter-destination delay variation values of the tree are different than the ones set during the execution of the algorithm, as the nodes receive the message directly from the source [35]. Thus, it is possible that a higher quality core node existed but was not checked for candidacy.

In trees generated by KMKh, group members receive the message from the core node [30]. The k-shortest-path approach they propose, only improves the inter-destination delay variation of the tree when a group member is located on the extended path. The tolerance value (p) the algorithm sets to mimic the inter-destination delay

variation of the tree and the discarding of nodes conforming with case (1), result in failure to generate solutions in 9% of the cases according to the performed simulations in Chapter 5.

In ATabu, the fitness function does not reflect the inter-destination delay variation of the generated tree. When cases (1) or (3) occur in the generated tree with some group members, the delay s to v_c is added to inter-destination delay variation of the remaining v_c to group member delays. ATabu on average checks the candidacy of 30% of the nodes guided by the fitness function that penalizes cases (1) and (3). Thus, it is possible that a higher quality core node existed but was not checked.

KMK and KMKh set their own tolerance value (p) to mimic the maximum accepted inter-destination delay variation of a DVBM tree. DDVCA, ESC, and ATabu do not consider a tolerance value on the inter-destination delay variation of a DVBM tree, they return the lowest inter-destination delay variation tree the algorithm finds. DDVCA, ESC, KMK, and ATabu cannot generate a lower inter-destination delay variation tree than the shortest path tree rooted at the selected core node. The inter-destination delay variation of a tree can be lowered by replacing shortest paths from the source to some group members with longer paths.

The core node in Directional Core Selection (DCS) algorithm designed, is selected by searching from group members by gradual radius expansion. A core node is found when the searches intersect at a node complying with both constraints. To further decrease the inter-destination delay variation of the tree, another algorithm Build Lower Variation Tree (BLVT) is designed. In BLVT shortest paths to some group members are replaced by longer paths. Longer paths are selected by generating k-shortest-paths bound by the maximum delay from the source to a group member ($maxD$) of the tree, and taking the longest path of the k paths. The source sends a unicast message to the core node, and the core sends the message to group members

using the multicast tree. In DCS and DCSBLVT the end-to-end delay and inter-destination delay variation values set during the execution of the algorithms reflect the values of the generated tree.

In dynamic DVBMT, multicast group members can leave the multicast session and new nodes can join the multicast session after issuing leave and join requests, respectively. The multicast tree should be updated in response to changes in multicast group membership, without violating the end-to-end delay and inter-destination delay variation constraints. Dynamic DVBMT can be tackled by re-executing the algorithm after every dynamic request to generate a new tree and use it for routing subsequent packets. However, this method is very costly as the algorithm is re-executed, old paths are removed, and new paths are formed.

Dynamic DVBMT is not handled by DDVCA, ESC, KMK, KMKh, and ATabu. Thus, the algorithms are re-executed after every request to obtain a new tree for the modified multicast group. The approach given in this dissertation for dynamic DVBMT, tries to avoid reconstruction of the multicast tree by adding a new path without effecting any of the paths from the source to nodes in M. However, if adding a new path does not satisfy the end-to-end delay and inter-destination delay variation constraints, reconstruction cannot be avoided.

In the next couple of paragraphs, existing multi-core based algorithms for multicast communication are given and the motivation for proposing MCDVBMT is stated.

Distributed Core Multicast (DCM) is a core-based routing protocol that utilizes multiple cores, but does not take into consideration any QoS constraints. QoS Core Selection Algorithm (QCSA) is a distributed core selection algorithm taking into consideration core-to-end delay, delay-jitters, and bandwidth QoS constraints [14]. Putthividya and Tavanapong propose three clustering algorithms to choose a minimal set of core nodes satisfying end-to-end delay [52]. They also propose a distributed

algorithm with backup cores [51]. Delay constrained multiple core selection algorithms are suggested by [33,55]. Delay constrained and cost minimizing multi-core algorithm is proposed in [34]. Ordered Core-Based Tree (OCBT) constructs a unique tree governed by multiple cores to span the entire receiver group [57]. Using multiple cores reduces the delay between source nodes and group members, avoids the core node being a single point of failure, and reduces traffic concentration [9,71]. However, using multiple cores increases the cost of the session, since every sender router sends one unicast stream to each core [22, 51, 52].

When single-core based multicast algorithms fail to generate a tree satisfying delay and delay variation constraints such that all paths to multicast group members pass through a single core node, the multi-core might as it is less constrained.

In the multi-core multicast approach, each core is the root of a separate multicast tree, and there is no coordination between cores. In single-core multicast trees, a core node is a single point of failure. When the core fails, all group members suffer from recovery delay, that includes the cost of re-running the algorithm, tearing down old paths, and establishing new ones [20]. On contrary, when one of the core nodes fail in a multi-core multicast, only group members receiving the message from the failing core node suffer from recovery delay.

Single core-based trees may cause traffic concentration, where some links in the network are much more heavily utilized than others [11, 68]. On contrary, the multi-core approach avoids the problem of traffic concentration as different cores are used to disseminate the message to multicast group members.

Motivated from the above-mentioned advantages of multi-core trees, a multi-core multicast approach to solve the DVBM problem is designed. The proposed three-phase algorithm, Multi-Core DVBM Trees (MCDVBMT), constructs multiple multicast trees rooted at every core node satisfying end-to-end delay and inter-destination

delay variation constraints. Simulation results show that when existing single-core based algorithms fail to construct a tree satisfying both QoS constraints, MCDVBMT generates a result using multiple rooted trees at selected core nodes. MCDVBMT generates less inter-destination delay variation and traffic concentration than existing single-core algorithms. In addition, group members suffer from lower recovery delay when a core node fails. However, they have higher end-to-end delay and cost than single-core trees [27]. The dynamic version of MCDVBMT is also given that reorganizes the multicast trees in response to changes to the multicast group members.

Chapter 3

On Near Optimal Time Multicast Algorithms in Mesh and Torus Networks

3.1 System Model and Problem Specification

An interconnection network consists of nodes. Every node has its own router, processor, local memory, and communication links. The router is responsible for the entering, leaving, and passing of messages through the node and is connected to its processor by pairs of internal channels. One internal channel is for incoming traffic and the other, for outgoing. The number of internal channels indicate the number of messages that a processor can send concurrently. Pairs of external channels connect nodes to each other.

Multicast communication is modeled as a graph theoretical problem where $G(V, E)$ is a graph with a set of V nodes and E edges representing links between nodes. The source node, s and $d_1, d_2, d_3, \ldots, d_q$ denoting q destination nodes form the multicast

set, M.

An Optimal Multicast Tree (OMT), $T(V, E)$, for a multicast set M in G is a subtree of G, such that $q \subseteq V(T)$, $D_T(s, u_i) = D_G(s, u_i)$ for $1 \leq i \leq q$, and $|E(T)|$ is as small as possible.

An nD mesh has $k_0 \times k_1 \times \cdots k_{n-1}$ nodes, where k_i is the number of nodes in the ith dimension. Every node in an nD mesh has an n-coordinate vector $(x_0, x_1, \cdots, x_{n-1})$, where $0 \leq x_i \leq k_i - 1$ for all i, $0 \leq i \leq n-1$. Two nodes $x(x_0, x_1, \cdots, x_{n-1})$ and $y(y_0, y_1, \cdots, y_{n-1})$ are connected if $x_i = y_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $x_j = (y_j \pm 1)$ [46]. The distance between nodes x and y is $\sum_{i=0}^{n-1} |y_i - x_i|$. The degree of a node in a nD mesh can be from n to $2n$. The diameter of a 2D mesh is $\sum_{i=0}^{n-1} k_i - n$.

In a mesh, a communication between the first and the last node of a dimension passes through a long path because of the absence of a direct edge between them. If an edge is added, the structure becomes symmetric and the distance between these nodes reduces roughly by half given that the message can travel from two opposite directions to reach the same destination. The resulting structure is a torus.

An nD torus is an nD mesh with wraparound links. Two nodes $x(x_0, x_1, \cdots, x_{n-1})$ and $y(y_0, y_1, \cdots, y_{n-1})$ are connected if $x_i = y_i$ for all i, $0 \leq i \leq n-1$, except one, j, where $x_j = (y_j \pm 1) \bmod k_j$. The distance between nodes $x(x_0, x_1, \cdots, x_{n-1})$ and $y(y_0, y_1, \cdots, y_{n-1})$ is $\sum_{i=0}^{n-1} \min(|y_i - x_i|, k_i - |y_i - x_i|)$ [46]. Every node in a nD torus has degree $2n$. The diameter of an nD torus is $\sum_{i=0}^{n} \lfloor \frac{k_i}{2} \rfloor$. Many properties result from the symmetric nature of an nD torus including low contention latency, high channel bandwidth, and balanced use of transmission channels.

A zone Z of an nD mesh or torus is a submesh represented by its two diagonal nodes. A node is in a zone, if its coordinate values are within the coordinate values of the two diagonal nodes of that zone [28]. Zoning is used to restrict the boundary

of the network.

A d_i is a destination node where $1 \leq i \leq q$. $D(u, v)$ is the distance between nodes u and v. d is the diagonal node $d = (X_{max}, Y_{max})$ where $X_{max} = \max\{x_1, \ldots, x_k\}$, $Y_{max} = \max\{y_1, \ldots, y_k\}$, x_k is the x coordinate of node d_k and y_k is the y coordinate of node d_k. sd is the diagonal line from s to d. DP is the diagonal path from s to d, approximating the sd. $D_{max} = \max\{D(s, d_1), \ldots, D(s, d_k)\}$. UDP is the set containing destination nodes in the upper part of the DP and LDP is the set containing destination nodes in the lower part of the DP. Intermediate nodes are nodes between s and leaf nodes.

The multicast communication services used by HMDIAG are the following:

- *Permanent Absorb, Forward and Retransmit (PAFR)*: The message is absorbed while being forwarded to the node towards the next destination node. The node might also retransmit the message.
- *Forward and Retransmit (FR)*: The message is forwarded towards the next destination node. The node might also retransmit the message.
- *Permanent Absorb and Forward (PAF)*: The message is absorbed. The node might also forward the message.

The problem of multicast in an nD torus is transformed to multicast in 2^n equal nD meshes (Algorithm 3.1). The algorithm, given a vector of dimension details, $dims$, of size n, $depth = 0$, and $curr =$ "" recursively generates the *sources* and *endpoints* of the meshes. Algorithm 3.2, given the source node and the number of dimensions, generates the endpoint of the zone.

For example, given $dims = ((0, m-1), (0, n-1))$, Algorithm 3.1 creates four meshes $\{(0,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1)\}$, $\{(m-1, 0) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1)\}$, $\{(0, n-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil)\}$, $\{(m-1, n-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil)\}$ (Figure 3.1). $sources = ((0,0), (m-1, 0), (0, n-1), (m-1, n-1))$ and $endpoints = ((\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil -$

Algorithm 3.1 Generate Subproblems

```
1: function GenerateSubproblems(dims,depth,curr)
2:     if depth = dims.size() then
3:         sources.push(curr)
4:         endpoints.push(GenerateEndpoints(curr, dims))
5:         curr = ""
6:         return
7:     for i = 0 : dims[depth].size() do
8:         GenerateSubproblems(dims, depth + 1, curr + "" + dims[depth][i])
       return sources, endpoints
9: end function
```

$1), (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil))$. Similary, given $dims = ((0, m-1), (0, n-1), (0, p))$, Algorithm 3.1 creates eight meshes $\{(0,0,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil - 1)\}$, $\{(m-1,0,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil - 1)\}$, $\{(m-1,0,p-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil)\}$, $\{(0,0,p-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil)\}$, $\{(0,n-1,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil - 1)\}$, $\{(m-1,n-1,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil - 1)\}$, $\{(m-1,n-1,p-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil)\}$, $\{(0,n-1,p-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil)\}$. $sources = ((0,0,0), (m-1,0,0), (m-1,0,p-1), (0,0,p-1), (0,n-1,0), (m-1,n-1,0), (m-1,n-1,p-1), (0,n-1,p-1))$ and $endpoints = ((\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil - 1), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil - 1), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil), (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{p}{2} \rceil), (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil - 1), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil - 1), (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil), (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{p}{2} \rceil))$.

M is also partitioned into 2^n subsets, containing the destination nodes belonging to the 2^n meshes. Dividing an nD torus into 2^n nD meshes forces every destination node to receive the message from the closest source node. This reduces the upper

Algorithm 3.2 Generate Endpoints

```
1: function GenerateEndpoints(source,dims)
2:     sourceCoordinates ← tokenize(source)
3:     for i = 0 : sourceCoordinates.size() do
4:         if sourceCoordinates[i] = "0" then
5:             endpoint+ =" ceil("+ dims[i][1] + "/2) − 1"
6:         else
7:             endpoint+ =" ceil("+ dims[i][1] + "/2)"
       return endpoint
8: end function
```

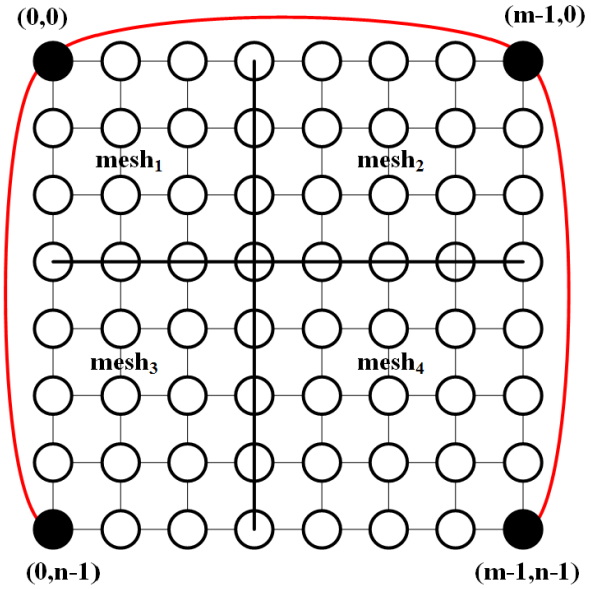


Figure 3.1: 2D torus

bound on multicast latency by almost half, enables destination nodes to receive the message in comparable time, and avoids traffic congestion by distributing the traffic load.

The multicast scheme the algorithms follow to send the message to the 2^n source nodes of the 2^n submeshes, is generated by Algorithm 3.3. Once the scheme from s to all source nodes of the meshes is generated, the source nodes of the meshes start local multicast. Prioritizing source nodes receiving the message earlier to further send the message to remaining uninformed source nodes, makes meshes start local multicast in parallel. In the meshes, the message is first transmitted along the Primary Diagonal Paths (PDP-s) and then intermediate nodes on the PDP-s retransmit the message along Secondary Diagonal Paths (SDP-s) and/or paths branching from the PDP. In an nD torus, at most $n-1$ DP-s are created. One PDP and $n-2$ SDP-s.

The Routing function $R(u, v) = w$ utilized for sending a message from a node

$u(x_0, x_1, \cdots, x_{n-1})$ to a node $v(y_0, y_1, \cdots, y_{n-1})$ is :

$$R(u,v) = \begin{cases} v & \text{if } \exists \text{ a pair, p in scheme } | \\ & p[0] = u \text{ and } p[1] = v \\ (z_i, \cdots, z_{n-1}) \mid z_i = x_i \text{ and } z_j = x_j + 1 & \text{if } x_i = y_i \, \forall i \mid 0 \leq i \leq n-1 \\ & \text{except } j \text{ where } x_j < y_j \\ (z_i, \cdots, z_{n-1}) \mid z_i = x_i \text{ and } z_j = x_j - 1 & \text{if } x_i = y_i \, \forall i \mid 0 \leq i \leq n-1 \\ & \text{except j where } x_j > y_j \end{cases} \tag{1}$$

In the presented theoretical results, the underlying architecture is assumed to be one-port. At each time unit a node can send a message to one of its neighbors or receive a message from one of its neighbors. A message can be transmitted over different links simultaneously.

When calculating multicast latency, the underlying architecture is assumed to be all-port, where a node can transmit multiple messages at a time because of the existence of several pair of internal channels. Nodes also support intermediate reception (IR). IR allows a router to send an incoming message to the local processor and forward it to another router at the same time [21].

Any node in a mesh or torus can be s, for simplicity it is fixed at (0,0) [66].

Algorithm 3.3 Generate Multicast Scheme

```
1: function GenerateMulticastScheme(s, sources)
2:     Q.push(s), visited[s] =true
3:     while all sources are not visited do
4:         sender = Q.pop()
5:         for every unvisited source, s_i, sender is connected to by a wraparound link
6:         scheme.push(sender, s_i)
7:         visited[s_i] =true
       return scheme
8: end function
```

It is also assumed that messages are switched using one of the distance insensitive switching techniques.

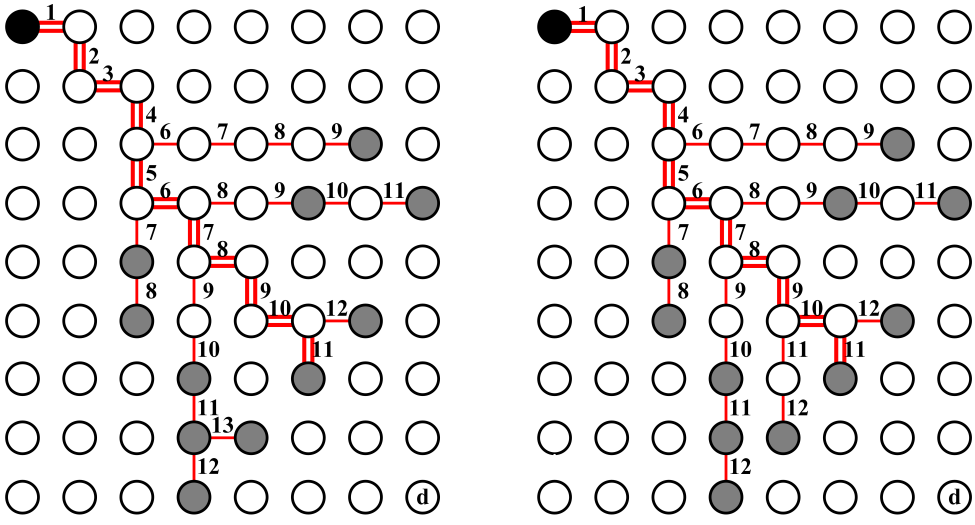
3.2 Modified Algorithms in 2D Mesh and Torus

3.2.1 Modified DIAG Algorithm in 2D Mesh

Modified DIAG (MDIAG) is divided into two parts. Part one (CreateDiagonalPath) of the algorithm creates the Diagonal Path (DP). Part two (CreateT), creates the multicast tree (T).

When creating T, MDIAG instead of selecting a closest node to u within the zone $\{s \Leftrightarrow u\}$, selects node v on the DP with the same x or y value as u, depending on the relative position of u to DP.

The main advantage of MDIAG is that it generates optimal time in most cases. When it does not generate optimal time, it generates optimal plus one time (proved in Proposition 3.2.5). However, MDIAG might generate more traffic.



(a) DIAG T (b) MDIAG T

Figure 3.2: DIAG and MDIAG created T-s

Figure 3.2(a) and Figure 3.2(b) show the T-s created for the same set of destination nodes by DIAG and MDIAG. DIAG creates a T with multicast time 13 and traffic 27. MDIAG creates a T with multicast time 12 and traffic 28. Double lines in the figures denote the DP, labels on links specify the time unit at which the message passes through that link, and shaded nodes represent the destination nodes.

Algorithm 3.4 Modified DIAG (MDIAG)

```
1: function CreateDiagonalPath(M, s)
2:     DP ← s, u ← s
3:     x_d ← max{x_{d_i}} ∀i | 1 ≤ i ≤ q
4:     y_d ← max{y_{d_i}} ∀i | 1 ≤ i ≤ q
5:     repeat
6:         x_{u'} ← x_u + 1, y_{u'} ← y_u, x_{u''} ← x_u, y_{u''} ← y_u + 1
7:         if D(u', sd) ≤ D(u'', sd) then
8:             u ← u'
9:         else
10:            u ← u''
11:        u_previous ← u
12:        DP ← DP ∪ u
13:        if direction changed from x to y then
14:            maxX ← u_previous
15:        else
16:            maxY ← u_previous
17:    until u = d
18:    return DP, maxX, maxY
19: end function
20: function CreateT(M, s, DP, maxX, maxY)
21:    T ← DP
22:    repeat
23:        u ← first node in M
24:        if u ∈ UDP then
25:            Select node v from maxX or DP where y_v = y_u
26:        else
27:            Select v from maxY or DP where x_v = x_u
28:        Connect u to v
29:        M ← M − {u}
30:    until M = ∅
31:    Cut the tail part of the DP that does not have a d_i or replicate nodes
32:    return T
33: end function
```

Algorithm 3.5 Modified PAIR (MPAIR)

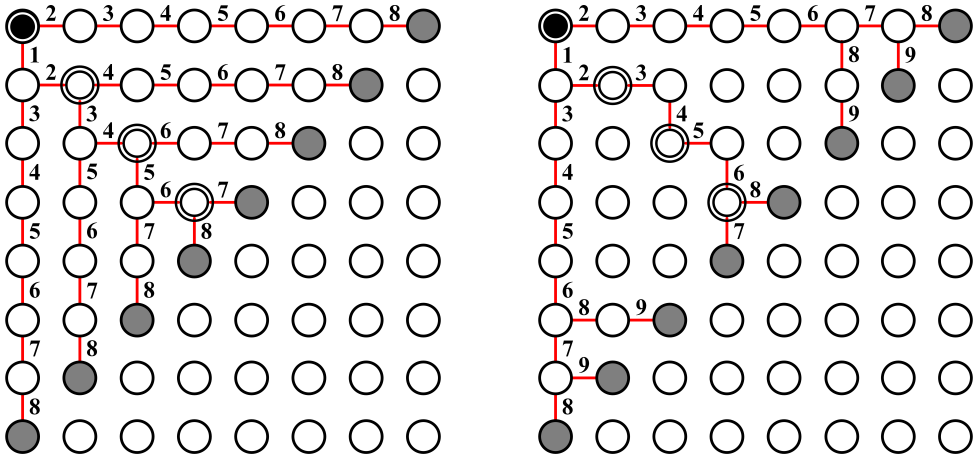
1: **function** MPAIR(M, s)
2: Arrange destination nodes in M
3: $T \leftarrow T \cup s$
4: **repeat**
5: Select destination nodes u and v, where u has the minimum x value x_{min} in M and v has the minimum y value y_{min} in M
6: Find a node w in T in zone $\{s \Leftrightarrow (x_{min}, y_{min})\}$ that is the closest to (x_{min}, y_{min})
7: Add node (x_{min}, y_{min}) to T through a shortest path to w by XY routing
8: Find a node c_1 in T and in zone $\{s \Leftrightarrow u\}$, that is the closest to u
9: Add u to T through a shortest path to c_1 by XY routing
10: Find a node c_2 in T and in zone $\{s \Leftrightarrow v\}$, that is the closest to v
11: Add v to T through a shortest path to c_2 by XY routing
12: $M \leftarrow M - \{u, v\}$
13: **until** $M = \emptyset$
14: Cut end parts of T not having d_i or replicate node
15: **return** T
16: **end function**

3.2.2 Modified PAIR Algorithm in 2D Mesh

Modified PAIR (MPAIR) instead of connecting nodes u and v to their corresponding intermediate node, connects them to a closest node c_1 to u and c_2 to v within the zone $\{s \Leftrightarrow u\}$ and $\{s \Leftrightarrow v\}$, respectively. This improves traffic but might increase time.

The multicast scheme MPAIR and MPAIR follow, first sends the message to the furthest intermediate node creating an abstract main path. Then nodes branching from that path receive the message.

Figure 3.3(a) and Figure 3.3(b) show the T-s created for the same set of destination nodes by PAIR and MPAIR, respectively. PAIR creates a T with time 8 and traffic 35. MPAIR creates a T with time 9 and traffic 27. Double lined nodes in the figures denote intermediate nodes.



(a) PAIR T (b) MPAIR T

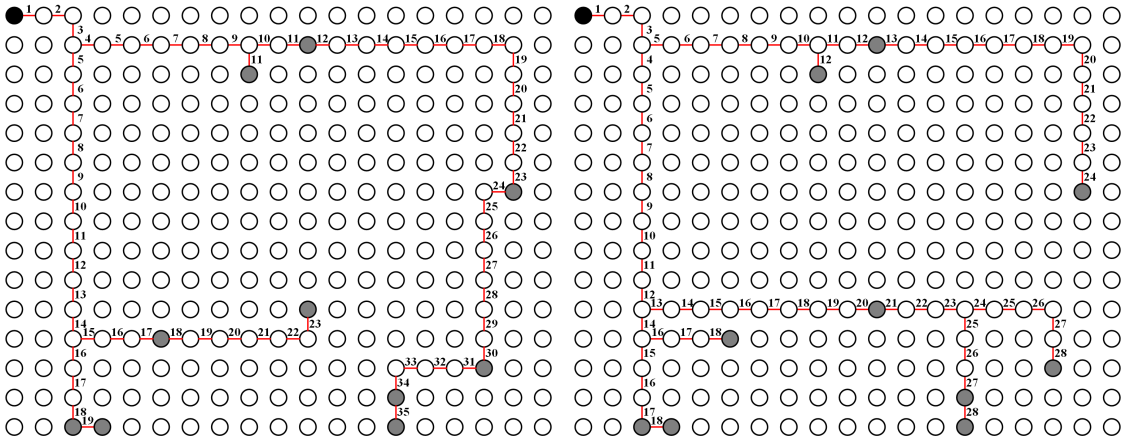
Figure 3.3: PAIR and MPAIR created T-s

3.2.3 Modified MIN Algorithm in 2D Mesh

Modified MIN (MMIN) instead of selecting the closest node c_1 and c_2 to every f and g, it selects the closest c_1 and c_2 to every f and g within the zone $\{s \Leftrightarrow f\}$ and $\{s \Leftrightarrow g\}$, respectively (Algorithm 3.6). This improves the time, but might increase the traffic.

The multicast scheme MIN and MMIN follow, first sends the message to the furthest destination node creating an abstract main path. Then nodes branching from that path receive the message.

Figure 3.4(a) and Figure 3.4(b) show the T-s created for the same set of destination nodes by MIN and MMIN algorithms. The T created by MIN has multicast time 35 and traffic 59, whereas the T created by MMIN has time 28 and traffic 61.



(a) MIN T (b) MMIN T

Figure 3.4: MIN and MMIN created T-s

Algorithm 3.6 Modified MIN (MMIN)

procedure MMIN(M, s)
  Arrange destination nodes in M
  $T \leftarrow T \cup s$
  Select a pair of destination nodes u and v, where node u has the minimum x value x_{min} in M and node v has the minimum y value y_{min} in M
  Find a node w in T in zone $\{s \Leftrightarrow (x_{min}, y_{min})\}$ that is the closest to (x_{min}, y_{min})
  Add node (x_{min}, y_{min}) to T through a shortest path to w by XY routing
  Find a node c in T and in the zone $\{s \Leftrightarrow u\}$, that is the closest to u
  Add node u to T through a shortest path to c by XY routing
  Find a node d in T and in the zone $\{s \Leftrightarrow v\}$, that is the closest to v
  Add node v to T through a shortest path to d by XY routing
  $M \leftarrow M - \{u, v\}$
  repeat
    Select a node f that has the minimum x value x_{min} in M
    Find a node c_1 in T and in the zone $\{s \Leftrightarrow f\}$, that is the closest to f
    Add node f to T through a shortest path to c_1 by XY routing
    Select a node g that has the minimum y value y_{min} in M
    Find a node c_2 in T and in the zone $\{s \Leftrightarrow g\}$, that is the closest to g
    Add node g to T through a shortest path to c_2 by XY routing
    $M \leftarrow M - \{f, g\}$
  until $M = \emptyset$
end procedure

3.2.4 Modified DIST Algorithm in 2D Mesh

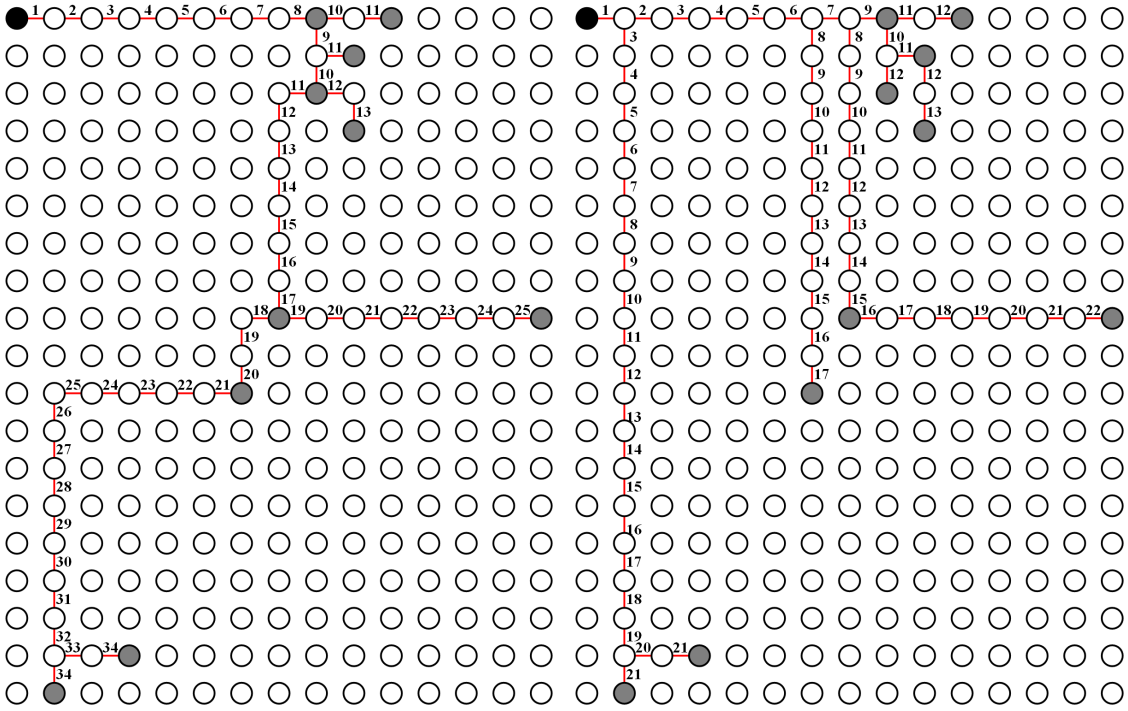
Modified DIST (MDIST) instead of selecting a closest node v to u, it selects a closest node v to u within the zone $\{s \Leftrightarrow u\}$ (Algorithm 3.7). This improves the time, but might increase traffic.

The multicast scheme DIST and MDIST follow, first sends the message to the furthest destination node creating an abstract main path. Then nodes branching from that path receive the message.

Figure 3.5(a) and Figure 3.5(b) show the T-s created for the same set of destination nodes by DIST and MDIST algorithms. The T created by DIST has multicast time 34 and traffic 47, whereas the T created by MDIST has time 22 and traffic 60.

Algorithm 3.7 Modified DIST (MDIST)

```
1: procedure MDIST(M, s)
2:     Sort destination nodes in increasing order of distances from s
3:     T ← T ∪ s
4:     repeat
5:         u ← first node in D
6:         Find a node v in T and in zone {s⇔u} that is the closest to u
7:         Add node u to T through a shortest path from v by XY routing
8:         D ← D − {u}
9:     until D = ∅
10: end procedure
```



(a) DIST T (b) MDIST T

Figure 3.5: DIST and MDIST created T-s

3.2.5 Modified Algorithms in 2D Torus

Algorithm 3.8 transfers the problem of multicast in a 2D torus into multicast in four 2D meshes. Using Algorithm 3.1, an $m \times n$ torus is divided into four meshes and *subproblems* are generated. The generated *subproblems* are $\{(0,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil - 1)\}$, $\{(m-1,0) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil - 1)\}$, $\{(0,n-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil - 1, \lceil \frac{n}{2} \rceil)\}$, $\{(m-1,n-1) \Leftrightarrow (\lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil)\}$. The source nodes of the four meshes are $(0,0)$, $(n-1,0)$, $(0,m-1)$, and $(m-1,n-1)$, respectively (Figure 3.1). On every subproblem algorithm A is applied and a tree is generated. A can be MPAIR, MDIAG, MMIN, or MDIST. The trees generated by the subproblems are connected to each other according to the scheme generated by Algorithm 3.3 (Algorithm 3.8).

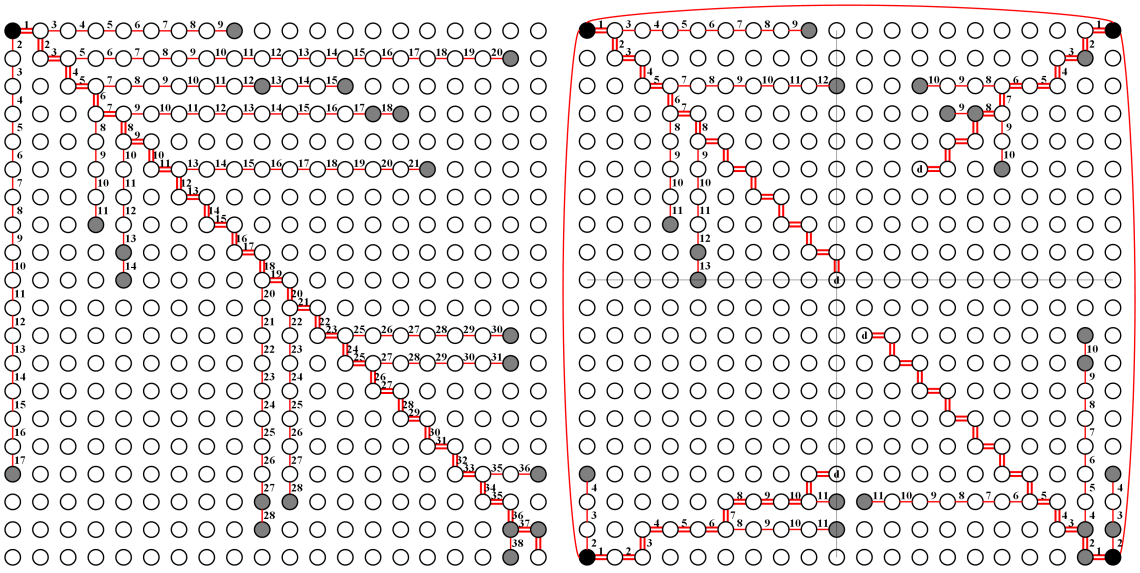
Algorithm 3.8 Multicast in 2D Torus

```
1: function GenerateMT(T, s, A)
2:     partition the nD torus into 2^n meshes using Algorithm 3.1 and get sources
       and endpoints
3:     partition M into 2^n subsets, M_i, containing destination nodes belonging to
       mesh_i for all i, 1 ≤ i ≤ 2^n
4:     for i = 0 : 2^n do
5:         if M_i ≠ ∅ then
6:             Apply A on mesh_i and M_i and get T_i
7:             T ← T ∪ T_i
8:     scheme ← GenerateMulitcastScheme((0, 0), sources)
9:     for every sender, receiver pair in scheme do
10:        T ← T∪ add a link from sender to receiver
11:    return T
12: end function
```

Figure 3.6(a) and Figure 3.6(b) show the T-s generated for the same set of destination nodes by MDIAG in a 2D mesh and torus, respectively. MDIAG in a 2D mesh generates a T with time 38 and traffic 143. In a 2D torus, it generates a T with time 15 and traffic 86.

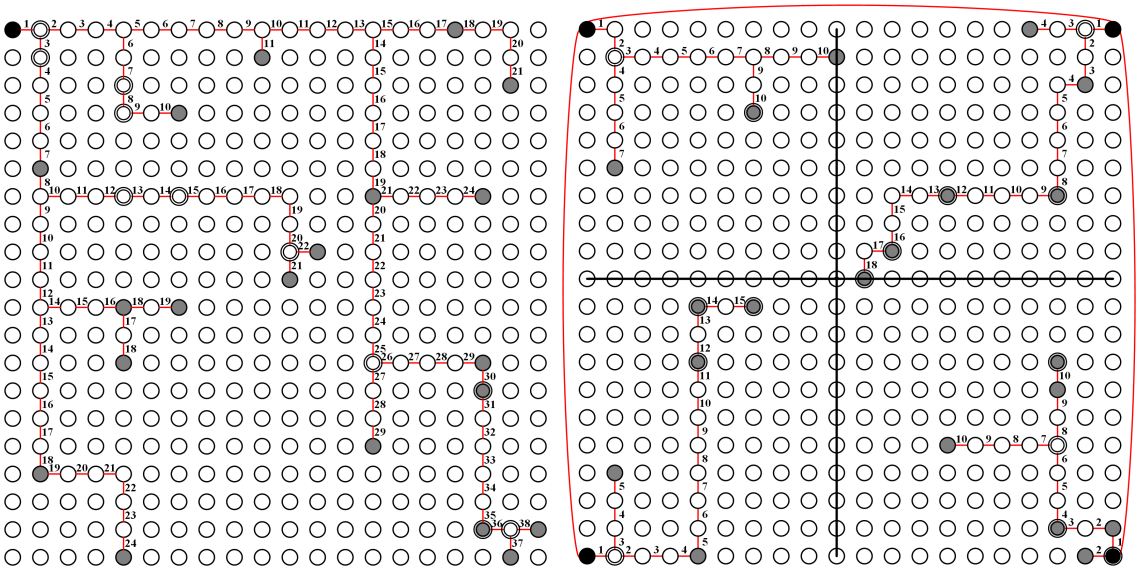
Figure 3.7(a) and Figure 3.7(b) show the T-s generated for the same set of destination nodes by MPAIR in a 2D mesh and torus, respectively. MPAIR in a 2D mesh generates a T with time 38 and traffic 100. In a 2D torus, it generates a T with time 20 and traffic 71.

Figure 3.8(a) and Figure 3.8(b) show the T-s generated for the same set of destination nodes by MMIN in a 2D mesh and torus, respectively. MMIN in a 2D mesh generates a T with time 37 and traffic 105. In a 2D torus, it generates a T with time 20 and traffic 75.



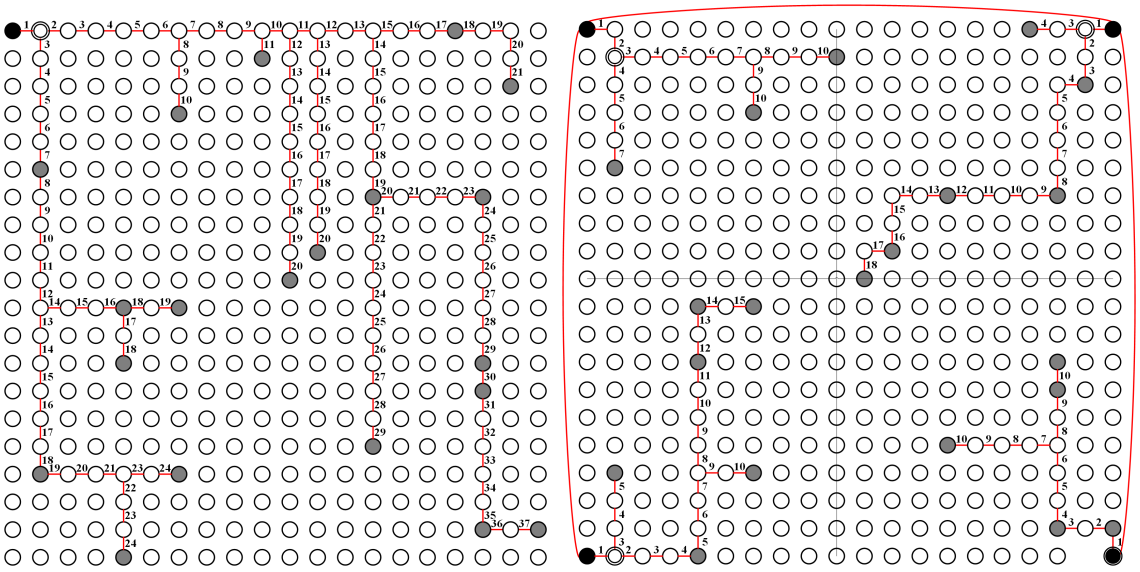
(a) MDIAG T in 2D mesh (b) MDIAG T in 2D torus

Figure 3.6: MDIAG T-s in 2D mesh and torus



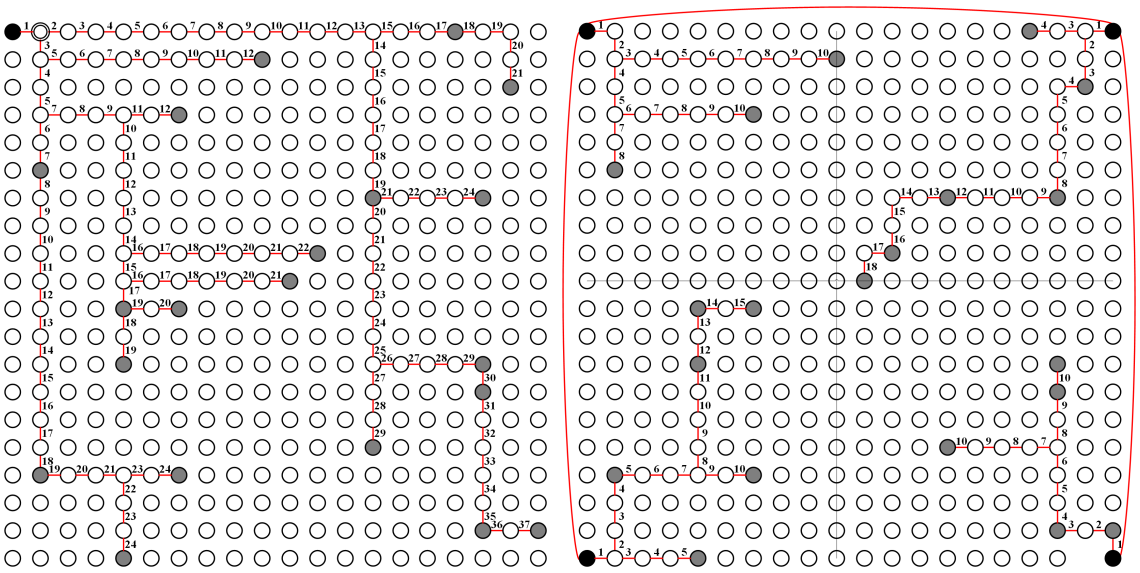
(a) MPAIR T in 2D mesh (b) MPAIR T in 2D torus

Figure 3.7: MPAIR T-s in 2D mesh and torus



(a) MMIN T in 2D mesh (b) MMIN T in 2D torus

Figure 3.8: MMIN T-s in 2D mesh and torus



(a) MDIST T in 2D mesh (b) MDIST T in 2D torus

Figure 3.9: MDIST T-s in 2D mesh and torus

Figure 3.9(a) and Figure 3.9(b) show the T-s generated for the same set of destination nodes by MDIST in a 2D mesh and torus, respectively. MDIST in a 2D mesh generates a T with time 37 and traffic 112. In a 2D torus, it generates a T with time

20 and traffic 75.

3.2.6 Time Complexity of the Modified Algorithms

3.2.6.1 Time Complexity of MDIAG

The time complexity of DIAG is $O(qn^2)$ in an $n \times n$ mesh and torus [28].

Proposition 3.2.1 *The time complexity of MDIAG is $O(qn)$ in an $n \times n$ mesh and torus.*

Proof The multicast preparation algorithm of MDIAG takes $O(n)$ time. In the T creation part, for every destination node d_i, finding the closest node on the DP takes $O(\log n)$ time since binary search is used on the $minX$ or $minY$. If binary search on $minX$ or $minY$ vectors does not return a node, another binary search is performed on the DP. Connecting a d_i to a closest node takes $O(n)$ time. Therefore, the complexity of part two of MDIAG is $O(qn)$. Thus, the time complexity of MDIAG is $O(qn)$ in an $n \times n$ mesh.

Dividing the torus into four meshes and partitioning the destination nodes into four subsets takes $O(q)$ time. Applying MDIAG in the four meshes of size $\frac{n}{2} \times \frac{n}{2}$ takes $\sum_{i=1}^{4} O(q_i \frac{n}{2}) = O(q\frac{n}{2})$. Combining the four T-s into a single one takes $O(1)$ time. Thus, the time complexity of MDIAG is $O(q\frac{n}{2} + q + 1) = O(qn)$ in an $n \times n$ torus. This improvement in time complexity is due to a change in the implementation of part two of the algorithm. In DIAG it takes $O(qn^2)$ time, whereas in MDIAG it takes $O(qn)$ time. ∎

3.2.6.2 Time Complexity of MPAIR

The time complexity of PAIR is $O(qn)$ in an $n \times n$ mesh and torus [45].

Proposition 3.2.2 *The time complexity of MPAIR is $O(qn)$ in an $n \times n$ mesh and torus.*

Proof Arranging destination nodes takes $O(q \log q)$ time. To find a closest node in T to the d_i destination nodes, only leaf nodes of the T within the zone $\{(0,0) \Leftrightarrow (d_i)\}$ are checked. Leaf nodes are the closest nodes to a d_i. There are at most $n + n - 1$ leaf nodes within the zone $\{(0,0) \Leftrightarrow (n,n)\}$, so it takes $O(q(n + n - 1)) = O(qn)$ time. Connecting a d_i to a closest node takes $O(n)$ time. Thus, the time complexity of MPAIR is $O(q \log q + qn)) = O(qn)$ in an $n \times n$ mesh.

Dividing the torus into four meshes and partitioning the destination nodes into 4 subsets takes $O(q)$ time. Applying MPAIR in the four meshes of size $\frac{n}{2} \times \frac{n}{2}$ takes $\sum_{i=1}^{4} O(q_i \frac{n}{2}) = O(q\frac{n}{2})$. Combining the four T-s into a single one takes $O(1)$ time. Thus, the time complexity of MPAIR is $O(q\frac{n}{2} + q + 1) = O(qn)$ in an $n \times n$ torus. ∎

3.2.6.3 Time Complexity of MMIN

The time complexity of MIN is $O(qn^3)$ in an $n \times n$ mesh.

Proposition 3.2.3 *The time complexity of MMIN is $O(qn)$ in an $n \times n$ mesh and torus.*

Proof Arranging destination nodes takes $O(q \log q)$ time. Selecting a closest node in T to a node to be added to the tree takes $O(n)$ time, since only the leaf nodes of the T are checked. Finding a path from the existing multicast tree to the node to be added takes $O(n)$ time. Thus, the time complexity of MMIN is $O(qn)$ in an $n \times n$ mesh.

Dividing the torus into four meshes and partitioning the destination nodes into 4 subsets takes $O(q)$ time. Applying MMIN in the four meshes of size $\frac{n}{2} \times \frac{n}{2}$ takes $\sum_{i=1}^{4} O(q_i \frac{n}{2}) = O(q\frac{n}{2})$. Combining the four T-s into a single one takes $O(1)$ time. Thus, the time complexity of MMIN is $O(q\frac{n}{2} + q + 1) = O(qn)$ in an $n \times n$ torus. ∎

3.2.6.4 Time Complexity of MDIST

The time complexity of DIST is $O(qn^3)$ in an $n \times n$ mesh.

Proposition 3.2.4 *The time complexity of MDIST is $O(qn)$ in an $n \times n$ mesh and torus.*

Proof Sorting destination nodes takes $O(q \log q)$ time. Finding a path from the existing multicast tree to the node to be added takes $O(n)$ time. Connecting nodes takes $O(n)$ time. Thus, the time complexity of MDIST is $O(qn)$ in an $n \times n$ mesh. Dividing the torus into four meshes and partitioning the destination nodes into 4 subsets takes $O(q)$ time. Applying MDIST in the four meshes of size $\frac{n}{2} \times \frac{n}{2}$ takes $\sum_{i=1}^{4} O(q_i \frac{n}{2}) = O(q\frac{n}{2})$. Combining the four T-s into a single one takes $O(1)$ time. Thus, the time complexity of MDIST is $O(q\frac{n}{2} + q + 1) = O(qn)$ in an $n \times n$ torus.

3.2.7 Bounds on Time of the Modified Algorithms

In [44], it is proven that the minimum possible multicast time in a 2D mesh:

$$T_{optimal} = \begin{cases} D_{max} & \exists \text{ a unique node at distance } D_{max} \\ D_{max} + 1 & \text{otherwise} \quad . \end{cases} \tag{2}$$

Proposition 3.2.5 *In a 2D mesh MDIAG always generates optimal or optimal plus one time.*

Proof In the multicast scheme followed by MDIAG, the message is first sent to the last node on the DP. Next, remaining uninformed destination nodes branching from the DP receive the message.

Case 1: MDIAG generates optimal time.

Case 1a: There are more than one nodes at distance D_{max}.

If there is no destination node on the DP with distance D_{max}, the message is sent to the furthest node v on the DP. If there is a destination node u with D_{max} distance on the DP, it will receive the message at time D_{max}. If a D_{max} distance node branches from v, it will receive the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + 1$ for all i where $1 \leq i \leq q$. All these nodes branch from the path leading to v or u or are on the DP. A branching leads to a delay of one time unit. Since $D(s, d_i) \leq D_{max}$, $D(s, d_i) + 1 \leq D_{max} + 1$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most $D_{max} + 1$, which is optimal.

Case 1b: There is a unique node u at distance D_{max} on the DP.

If there is a unique node u at distance D_{max} on the DP, it receives the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + 1$ for all i where $1 \leq i \leq q$. The time unit of delay is the result of branching from the DP. Since $D(s, d_i) \leq D_{max} - 1$, $D(s, d_i) + 1 \leq D_{max}$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most D_{max}, which is optimal.

Case 1c: There is a unique node u at distance D_{max} branching from the last DP node.

The message is sent to the furthest node v on the DP. The D_{max} distance node branches from v, thus it receives the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + 1$ for all i where $1 \leq i \leq q$. All these nodes branch from the path leading to v. A branching leads to a delay of one time unit. Since $D(s, d_i) \leq D_{max} - 1$, $D(s, d_i) + 1 \leq D_{max}$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most D_{max}, which is optimal.

Case 2: MDIAG does not generate optimal time when there is a unique node u at distance D_{max} not on the DP or branching from the last DP node.

The message is sent to the furthest node v on the DP. All remaining destination

nodes receive the message by time $D(s, d_i) + 1$ for all i where $1 \leq i \leq q$. All these nodes branch from the path leading to v. A branching leads to a delay of one time unit. Since $D(s, d_i) \leq D_{max}$, $D(s, d_i) + 1 \leq D_{max} + 1$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most $D_{max} + 1$ which is optimal plus one.

Thus, MDIAG generates $T_{optimal}$ or $T_{optimal} + 1$. ∎

Proposition 3.2.6 *In a 2D mesh* $Time(DIAG) - 1 \leq Time(MDIAG) \leq Time(DIAG)$

Proof In DIAG, a d_i can be connected to a node on the T generating an extra time unit. ∎

Proposition 3.2.7 *In a 2D mesh* $Time(MPAIR) - 1 \leq Time(PAIR) \leq Time(MPAIR)$

Proof In MPAIR, a d_i can be connected to a node on the T generating an extra time unit. ∎

Proposition 3.2.8 *In a 2D mesh, Time(MDIST) ≤ Time(DIST)*

Proof Since MDIST is a shortest path algorithm, Time(MDIST) ≤ Time(DIST). ∎

Proposition 3.2.9 *In a 2D mesh, Time(MMIN) ≤ Time(MIN)*

Proof Since MMIN is a shortest path algorithm, Time(MMIN) ≤ Time(MIN). ∎

Proposition 3.2.10 *In a 2D torus* $Time(torus) \leq \max\{Time(mesh\ 1), Time(mesh\ 2), Time(mesh\ 3), Time(mesh\ 4)\} + 2$

Proof To provide an upper bound on the multicast time of any algorithm in a 2D torus, the delays caused by the wraparound links are added. This delay depends on the distribution of the destination nodes.

To multicast in a 2D torus, node $(0,0)$ informs node $(m-1,0)$ at time 1. At time 2, nodes $(0,0)$ and $(m-1,0)$ inform $(0,n-1)$ and $(m-1,n-1)$ respectively (Figure 3.1). At time 3, all four meshes start multicast within their respective meshes. If no nodes lie in mesh 3, mesh 1 starts local multicast at time 2. Also, if no nodes lie in mesh 4, mesh 2 starts local multicast at time 2. Thus, the maximum possible number of delays wraparound links may cause is two. ∎

3.3 Hybrid MDIAG (HMDIAG) Algorithm

In centralized routing, routing paths are selected at the source node and the routing information is added to the message. Consequently, centralized routing suffers from high message overhead as the addresses of all intermediate nodes are added to the message. In centralized routing, the time spent at the source node to select routing paths and start message sending should also be low. To tackle the disadvantages of centralized routing, the hybrid version of MDIAG, HMDAIG is designed. In hybrid routing, both the source and intermediate nodes make routing decisions and some intermediate nodes are added to the message.

Algorithm 3.9 Hybrid MDIAG (HMDIAG)

1: **procedure** HMDIAG($M, message$, n)
2: partition the nD torus into 2^n meshes using Algorithm 3.1 and get *sources* and *endpoints*
3: partition M into 2^n subsets, M_i, containing destination nodes belonging to $mesh_i$ for all i, $1 \leq i \leq 2^n$
4: apply CreateMessageHeader on $(M_i, sources[i], endpoints[i])$ for all i, $1 \leq i \leq 2^n$, to create $header_i$
5: encapsulate $header_i$ to $message$ to create $message_i$ for all i, $1 \leq i \leq 2^n$
6: send $message_i$ for all i, $1 \leq i \leq 2^n$
7: nodes in header of $message_i$ for all i, $1 \leq i \leq 2^n$, perform Communication-ServiceOperations
8: **end procedure**

Algorithm 3.10 Create Message Header

```
1: function CreateMessageHeader(M′, s, e)
2:     Sort M′ in increasing order of distances from s, DP ← s, v ← s
3:     for i = 0 : q′ do dOnDP[i] ← 0
4:     for j = 0 : n − 1 do
5:         if s_j < e_j then d_j ← Max{d_{q_j}} ∀ d_q ∈ M′
6:         else d_j ← Min{d_{q_j}} ∀ d_q ∈ M′
7:     create u_{n−1} nodes
8:     while v ≠ d do
9:         for j = 0 : n − 1 do
10:            u_j ← v
11:            if s_j < e_j then u_{j_j} ← v_j + 1
12:            else u_{i_j} ← v_j − 1
13:        v ← Min{M(u_i, sd)} ∀ i | 0 ≤ i ≤ n − 1
14:        index ← binarySearch(v, M′)
15:        if index ≠ −1 then v.flag ← "PAFR", dOnDP[i] ← 1
16:        else v.flag ← "FR"
17:        DP ← DP ∪ v
18:    header′ ← DP
19:    for i = 0 : q′ do
20:        if dOnDP[i] = 0 then
21:            v ← dOnDP[i], v.flag ← "PAF", header′ ← header′ ∪ v
22:    return header′
23: end function
```

HMDIAG divides the nD torus into 2^n meshes and M into 2^n sets. Creates at most 2^n headers by Algorithm 3.9, if there is a destination node in every mesh. The headers include nodes on the Primary Diagonal Path (PDP) and destination nodes that receive the message from paths branching from the PDP. Next, it encapsulates each header to a copy of the *message*, and transmits them (Algorithm 3.9).

In Algorithm 3.10, given a set of destination nodes $M' \in \{M_1, \cdots, M_{2^n}\}$, a source node $s \in \{s_1, \cdots, s_{2^n}\}$, and an endpoint $e \in \{e_1, \cdots, s_{2^n}\}$ *header′* is created. *header′* includes nodes on a DP and destination nodes not on the DP with their flags.

When an intermediate node receives the header flit, if it is not the first node in the header, forwards it towards the first node in the header using R.

Algorithm 3.11 Communication Service Operations

```
1: procedure CommunicationServiceOperations(flit)
2:     if P and A flags are set then Absorb flit
3:     Remove current from flit.H
4:     if R flag is set then
5:         previous ← node from where flit received
6:         next ← first node in flit.H
7:         DPDirDim ← the dimension the DP is taking based on current and next
           nodes
8:         for all nodes d_i in flit.H do
9:             if d_i.flag = "PAF" then
10:                if d_i and current have same value for the DPDirDim but different
                       dimension and D(s_i, d_i) = d(s_i, current) + d(current, d_i) then
11:                    retransHeader ← retransHeader ∪ d_i
12:                if d_i and current have same value for the DPDirDim and
                       D(s_i, d_i) = d(s_i, current) + d(current, d_i) then
13:                    newheader ← newheader ∪ d_i
14:        if newheader ≠ ∅ then
15:            newheader ←CreateMessageHeader(newheader, current, e_i)
16:            newflit ← flit, newflit.H ← newheader
17:            flit.H ← flit.H - newheader
18:        if retransHeader ≠ ∅ then
19:            retransflit ← flit, retransflit.H ← retransHeader
20:            flit.H ← flit.H - retransflit
21:    for all nodes d_i in flit.H do
22:        if d_i.flag = "PAFR" then
23:            destinationNodesLeft ← destinationNodesLeft + 1
24:    if F flag is set and destinationNodesLeft ≠ 0 then
25:        forward flit to the node returned by R(current, next)
26:    if newheader ≠ ∅ then
27:        retransmit newflit to node R(current, first node in newheader)
28:    if retransHeader ≠ 0 then
29:        retransmit retransflit to node R(current, first node in retransflit.H)
30: end procedure
```

When an intermediate node is the first node in the header, it performs CommunicationServiceOperations (Algorithm 3.11). If flags P and A are set, the flit is absorbed. The current node is removed from the header. If flag R is set, the header is checked for nodes with flag "PAF" that represent destination nodes not on a DP. The goal is to check if there are destination nodes that can be reached through a path branching from the current node towards any dimension except the dimension the PDP takes. If there is a difference in only one dimension between the *current* node and PAF nodes in the *header*, these nodes are nodes that receive the message along one direct dimension and are added to $retransHeader$. If there is a difference in more than one dimension, these nodes represent nodes that receive the message through more than one dimension and are added to *newheader*. Consequently, another DP, SDP, is created to send the message to the latter nodes. The flit is copied, the header of the $retransflit$ and the $newflit$ are set, and sent. The remaining nodes with flag "FR" were added by CreateMessageHeader to make branchings possible. Data flits follow the header flit. The order of first forwarding then retransmitting is crucial to achieve optimal or optimal plus $n-1$ time in a respective mesh (Proposition 3.3.2).

HMDIAG uses at most $n-1$ startup times. At startup one, message transmission is performed along the PDP. Retransmissions from the PDP and any SDP takes another startup time. HMDIAG is deadlock free since the source node is fixed, meshes do not share links, and the routing function R creates a channel dependency graph with no cycles.

Given $M = \{(0,0),(1,0),(1,1),(2,1),(2,2),(1,3)\}$ and message msg, Figure 3.10 shows how HMDIAG performs multicast in a 2D torus. Double lines represent the PDP.

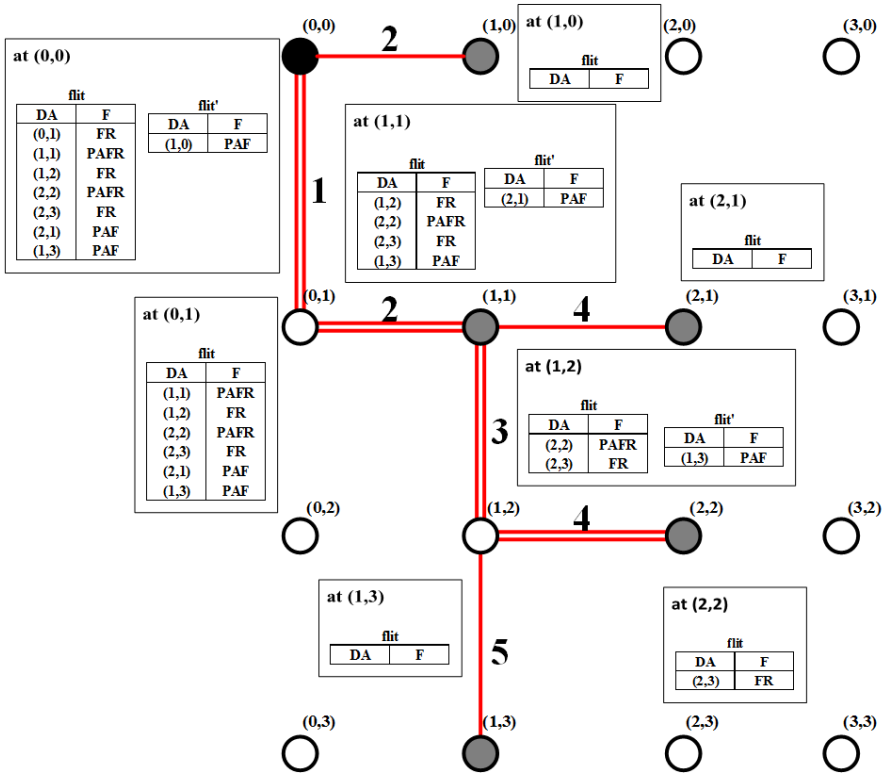


Figure 3.10: HMDIAG multicast in 2D torus

3.3.1 Time Complexity of HMDIAG

Partitioning an nD torus and M takes $O(q)$ time. Sorting q destination nodes takes $O(q \log q)$ time. Assume, all n dimensions have m nodes. Creating a DP takes $O(m \log q + q \log q)$ time, since at most $nm - n + q$ nodes can be on the DP-s created for all 2^n meshes and for each node a binary search takes $O(\log q)$ time. This process is repeated at most $n - 1$ times one for the PDP and $n - 2$ times for the SDP-s. Adding destination nodes not on the DP to the header, takes $O(q)$ time. Thus, the time complexity of CreateMessageHeader is $O(m \log q + q \log q)$. The time complexity of CommunicationServiceOperations is $O(q + m)$, since at most $nm - n + q$ nodes can be in the headers of all 2^n meshes and n is a constant.

Thus, the time complexity of the message preparation part of HMDIAG in nD torus is $O(m \log q + q \log q)$ and at every node in the message header $O(q+m)$ or $O(m \log q +$

$q \log q$) time is spent.

3.3.2 3-additive Approximation for Multicast Time in 2D Torus Networks Proof

Dividing a 2D torus into four equal meshes forces every destination node to receive the message from the closest source node s, s_2, s_3, or s_4. The optimal multicast time in a 2D or mesh or torus satisfies Equation 2: Consequently, given a multicast set M and the number of destination nodes at D_{max} distance, the optimal time in a 2D torus is the same as the optimal time in the 2D mesh including the D_{max} distance node or nodes.

HMDIAG in 2D torus follows the following multicast scheme: at time 1, s sends the message to s_2. At time 2, s sends the message to s_3 and s_2 sends the message to s. At time 3, all source nodes start local multicast.

Proposition 3.3.1 *The multicast time generated by HMDIAG in a 2D torus is always* $T_{optimal} + 3$ *in the worst case.*

Proof According to proposition 3.2.5, in the four meshes the time generated from their respective source nodes is $T_{optimal}$ or $T_{optimal} + 1$.

The four source nodes start local multicast at time 3. The time generated by HMDIAG depends on in which mesh the D_{max} distance node or nodes lie.
If the D_{max} distance node or nodes are in $mesh_4$, for any d_i in $mesh_4$, $D(s, d_i) = D(s_4, d_i) + 2$. Thus, the two time units of delay correspond to the two wraparound links connecting s_4 to s. Thus, the time is $T_{optimal}$ or $T_{optimal} + 1$.
If the D_{max} distance node or nodes are in $mesh_2$ or $mesh_3$, for any d_i in $mesh_2$, $D(s, d_i) = D(s_2, d_i)+1$. Similarly, for any d_i in $mesh_3$, $D(s, d_i) = D(s_3, d_i)+1$. Thus, there is one time unit of delay that makes the generated time equal to $T_{optimal} + 1$ or

$T_{optimal} + 2$.

If the D_{max} distance node or nodes are in $mesh_1$, the two extra times are pure delays. Thus, the time is $T_{optimal} + 2$ or $T_{optimal} + 3$.

Consequently, HMDIAG is a 3-additive approximation algorithm for multicast time. ∎

3.3.3 $(2n-1)$-additive Approximation for Multicast Time in nD torus networks Proof

Dividing an nD torus into 2^n equal meshes forces every destination node to receive the message from the closest source node s_i.

The optimal multicast time in an nD mesh or an nD torus satisfies:

$$T_{optimal} = \begin{cases} D_{max} & \exists \text{ a unique node at distance } D_{max} \text{ and } D_{max}-1 \\ D_{max}+x & \exists \text{ a unique node at distance } D_{max} \text{ and } x+1 \text{ nodes at} \\ & \text{distance } D_{max}-1 \text{ not on the path leading to the } D_{max} \\ & \text{distance node and branching from the same node} \\ & \text{where } 1 \le x \le n-1 \\ D_{max}+x & \exists\ x+1 \text{ nodes at distance } D_{max} \text{ branching from the same} \\ & \text{node where } 1 \le x \le n-1\ . \end{cases} \tag{3}$$

Consequently, given a multicast set M and the number of destination nodes at D_{max} distance, the optimal time in an nD torus is the same as the optimal time in the nD mesh including the D_{max} distance node or nodes.

Algorithm 3.3 prioritizes source nodes receiving the message earlier to send the

message to remaining uninformed source nodes of the $2^n - 1$ meshes. Consequently, meshes start local multicast at the earliest possible time unit. At least n time units are needed to propagate the message from s to s_{2^n}. At every time unit, the message propagates in one dimension. By the time the furthest source node, s_{2^n}, receives the message, all remaining source nodes are informed, as they are $n-1$ dimensions away from s. Thus, following this scheme, in an nD torus meshes start local multicast at time $n+1$.

Proposition 3.3.2 *The multicast time generated by HMDIAG in an nD torus is at most* $T_{optimal} + 2n - 1$.

Proof First, the proof that in an nD mesh HMDIAG always generates optimal or optimal plus $n-1$ time is given. In the multicast scheme followed by HMDIAG, the message is first sent to the last node on the PDP. Next, the message is sent to the last node on the SDP-s. Finally, remaining uninformed destination nodes branching from the PDP or SDP-s receive the message.

In an nD mesh, every node n_i receives the message by time $D(s, n_i)+NumberOfDP$ where $NumberOfDP$ is the number of diagonal paths. $NumberOfDP$ can at most be $n-1$, one PDP and $n-2$ SDP-s. A branching from a DP results in a time unit of delay. Consequently, the number of delays a node n_i suffers from depends on its location. If n_i is on the PDP, it receives the message at time $D(s, n_i)$. If n_i branches from PDP, it receives the message at time $D(s, n_i) + 1$. If n_i is on the first SDP, it receives the message at time $D(s, n_i) + 1$. If n_i branches from the first SDP, it receives the message at time $D(s, n_i) + 2$. If n_i is on the ith SDP, it receives the message at time $D(s, n_i) + i$. If n_i branches from the ith SDP, it receives the message at time $D(s, n_i) + i + 1$. There exists at most $n-2$ SDP-s. Thus, a node n_i in an nD array receives the message by time $D(s, n_i) + n - 1$.

Case 1: There is a unique node u at distance D_{max} on the PDP.

If there is a unique node u at distance D_{max} on the PDP, it receives the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + n - 1$ for all i where $1 \leq i \leq q$. Since $D(s, d_i) \leq D_{max} - 1$, $D(s, d_i) + n - 1 \leq D_{max} + n - 2$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most $D_{max} + n - 2$, which is optimal plus $n - 2$.

Case 2: There is a unique node u at distance D_{max} branching from the last PDP node.

The message is sent to the furthest node v on the PDP. The D_{max} distance node branches from v, thus it receives the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + n - 1$ for all i where $1 \leq i \leq q$. Since $D(s, d_i) \leq D_{max} - 1$, $D(s, d_i) + n - 1 \leq D_{max} + n - 2$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most $D_{max} + n - 2$, which is optimal plus $n - 2$.

Case 3: There is a unique node u at distance D_{max} not on the DP or branching from the last DP node.

The message is sent to the furthest node v on the DP. All remaining destination nodes receive the message by time $D(s, d_i) + n$ for all i where $1 \leq i \leq q$. Since $D(s, d_i) \leq D_{max} - 1$, $D(s, d_i) + n - 1 \leq D_{max} + n - 1$ for all i where $1 \leq i \leq q$. Thus, the multicast time is at most $D_{max} + n - 1$, which is optimal plus $n - 1$.

Case 4: There are more than one nodes at distance D_{max}.

If there is no destination node on the PDP with distance D_{max}, the message is sent to the furthest node v on the PDP. If there is a destination node u with D_{max} distance on the PDP, it will receive the message at time D_{max}. If a D_{max} distance node branches from v, it will receive the message at time D_{max}. All remaining destination nodes receive the message by time $D(s, d_i) + n - 1$ for all i where $1 \leq i \leq q$. Since $D(s, d_i) \leq D_{max}$, $D(s, d_i) + n \leq D_{max} + n - 1$ for all i where $1 \leq i \leq q$. Thus, the

multicast time is at most $D_{max} + n - 1$, which is optimal.

Thus, in the 2^n meshes the time generated from their respective source nodes at most $T_{optimal} + n - 1$.

In an nD torus at most n delays are added to a node, since it takes at most n time units to propagate the message from s to s_{2^n}. Thus, any node n_i in an nD torus receives the message by time $D(s, n_i) + 2n - 1$.

Consequently, HMDIAG is an $(2n-1)$-additive approximation algorithm for multicast time in nD torus. ∎

Chapter 4

On Delay and Delay Variation Multicast Routing

4.1 Network Model and Problem Specification

The multicast communication in a network is modeled as a graph $G(V, E, s, M)$ with a set of V nodes, E edges denoting links between nodes, s is the source node, $d_1, d_2, d_3, \ldots, d_q$ are the q members of the multicast group, M and $n = |V|$. An edge $e = (v_i, v_j)$ from node $v_i \in V$ to node $v_j \in V$ indicates a directed communication link from v_i to v_j. Every edge has a delay $D(e) : E \rightarrow \mathbb{R}^+$, that reflects the queuing, transmission, and propagation delays of the link. The total delay of a path from s to u, $P_T(s, u)$, is the sum of the delays of all the links on the path (4).

$$D(P_T(s, u)) = \sum e \in P_T(s, u) D(e) \tag{4}$$

The maximum difference between path delays from s to any two nodes in M is the inter-destination delay variation, dv (5).

$$dv = \max\{|\sum_{e \in P_T(s, v_i)} D(e) - \sum_{e \in P_T(s, v_j)} D(e)| \, \forall v_i, v_j \in M\} \tag{5}$$

The maximum path delay from s to a node in M is the maximum end-to-end delay, $maxD$ (6).

$$maxD = \max_{v_i \in M} \sum_{e \in P_T(s,v_i)} D(e) \tag{6}$$

Given a directed weighted graph $G = (V, E)$, $s \in V$, $M \subseteq V - s$, a link-delay function $D(e) : E \to \mathbb{R}^+$ where $e \in E$, maximum end-to-end delay Δ, and maximum inter-destination delay variation σ, the objective of DVBMT is to generate a tree, $T(s, M)$, which is a subgraph of G spanning s and the q group members with $maxD \leq \Delta$ and $dv \leq \sigma$. In addition to s and the q group members, T may have relay nodes. Relay nodes are intermediate nodes on the paths from s to group members.

The objective of DCS is to generate a tree T rooted at a selected core node satisfying $maxD \leq \Delta$ and $dv \leq \sigma$.

The objective of MCDVBMT is to generate trees rooted at multiple core nodes satisfying $maxD \leq \Delta$ and $dv \leq \sigma$, where the cardinality of core nodes $|C|$ is $1 < |C| < q$. $|C|$ should not be very close to one to avoid the drawbacks of single core multicast trees. $|C|$ should also not be very close to $|M|$, as it increases the cost.

This objective is solved in three phases: Phase one, generates for every multicast group member candidate core nodes satisfying the Δ constraint. Phase two, by eliminating candidate core nodes, reduces the maximum variation among candidate core nodes of all multicast group members. It returns a reduced candidate core node set such that selecting any combination of candidate core nodes for all multicast group members satisfies the σ constraint. Phase three, selects core nodes from the candidate core sets such that $|C|$ is optimized.

Phase one and two operate on vector sets of candidate core nodes. In phase three, the vector sets of candidate core nodes is modeled as a bipartite graph.

Let $G = (U \cup W, E)$ be a simple bipartite graph with U the set of left-hand vertices, W the set of right-hand vertices, edge set $E \subseteq U \times W$, $q = |U|$, and $p = |W|$. A

vertex $u \in U$ is a multicast group member and a vertex $w \in W$ is a candidate core node. If candidate core node u belongs to the candidate core node set of multicast group member w, an edge (u, w) exists between them.

A matching in G is a set of edges, $MG \subseteq E$, such that each vertex in $U \cup W$ is an endpoint of at most one edge in MG. A maximum matching is a matching of maximum size. A relaxation of the problem is semi-matching, where each vertex in U is an endpoint of exactly one edge in MG. For a semi-matching to exist, the degree of every U vertex must be at least 1. Finding semi-matchings of maximum weight was studied in [39] and matching U vertices with W vertices as fairly as possible by minimizing the variance of the matching edges was studied in [29].

Assignment of cores to group members is equivalent to finding a semi-matching. Thus, the objective of phase three of MCDVBMT is to find a semi-matching in G where the number of nodes in W participating in the matching is optimized. Given that phase two generates candidate core nodes within the Δ and σ constraints, any matching in the bipartite graph satisfies both constraints.

4.2 Directional Core Selection (DCS) Algorithm

Directional Core Selection (DCS) selects a core node by starting a search from every group member. Q is a vector of priority queues of size q. $Q[i]$ holds visited vertices of search $i \mid 0 \leq i \leq q-1$. *cost* and *predecessor* are vector of vectors of size $q+1$. $cost[0]$ and $predecessor[0]$ hold delay and predecessor values obtained from Dijkstra's algorithm from s to all nodes in G. $cost[i+1]$ and $predecessor[i+1]$ hold the costs and predecessors of visited vertices of search $i \mid 0 \leq i \leq q-1$. *explored* is an integer vector of size n. $explored[i]$ represents the number of directional searches that have visited vertex i. *candidate* is a Boolean vector of size n. $candidate[i]$ indicates if node i is a candidate. dv is a vector of size n. $dv[i]$ holds the inter-destination delay

variation if node i is the core. v_c is the index of the selected core (Algorithm 4.1).

The first step of DCS algorithm is finding shortest paths from the source to all nodes in the network. The group member from where every search starts is pushed into queue, Q. The transpose of the directed graph $G(V, E)$, graph $G'(V, E')$ such that $\forall e \in E \mid e = (u, v)$, $e' = (v, u) \in E'$, is computed. The search radius is gradually expanded until the searches intersect at a node complying with both constraints or when all nodes are explored from all directions.

Algorithm 4.1 Directional Core Selection (DCS)

```
1: function DCS(M, s, G, σ, Δ)
2:     Create Q, cost, predecessor, explored, candidate, dv, v_c ← −1
3:     cost[0], predecessor[0] ← Dijkstra from s to all nodes in G
4:     for i = 0 : q − 1 do
5:         Q[i].push(makePair(M[i], 0)   ▷ start point of every search is member M[i]
6:         cost[i + 1][M[i]] ← 0         ▷ cost to M[i] from M[i] is zero
7:     G' ← transpose G
8:     while v_c = −1 do
9:         r ← index of the search with the lowest cost or expansion radius in Q
10:        if r = −1 then break          ▷ all directional searches explored all nodes
11:        u ← Q[r].pop()
12:        explored[u] + +, visited[r][u] ← true
13:        if candidate[u] and explored[u] = q then
14:            v_c ← CF(cost, candidate, u, σ, Δ, dv)
15:        if v_c! = −1 then break       ▷ v_c found
16:        for every non-visited neighbor v of u in G' do
17:            tempCost ← cost[r + 1][u]+ Delay(v, u)
18:            if cost[r + 1][v] > tempCost then
19:                cost[r + 1][v] ← tempCost, predecessor[r + 1][v] ← u
20:                Q[r].push(makePair(v, tempCost))
21:    if v_c = −1 then                  ▷ All searches explored all nodes, no solution
22:        v_c ← lowest dv value node
23:        T ← BLVT(v_c, s, Δ, M, predecessor)
24:    else                              ▷ solution found, T construction
25:        T.push(makePair(getPath(s, v_c, predecessor[0]), cost[0][v_c]))
26:        for i = 0 : q do
27:            T.push(makePair(getPath(v_c, M[i], predecessor[i + 1]), cost[i + 1][v_c]))
28:    return T
29: end function
```

Algorithm 4.2 Core Found (CF)

```
1: function CF(cost, candidate, v_c, σ, Δ, dv)
2:     for i = 1 : q + 1 do
3:         delays[i − 1] ← cost[i][v_c] + cost[0][v_c]   ▷ delays is a vector of size q. delays[i] represents the delay to group member i
4:         if delays[i − 1] > Δ then
5:             candidate[v_c] ← false                    ▷ Δ not satisfied
6:             return −1
7:     minD ← Min {delays[j]}∀j, 0 ≤ j ≤ q − 1
8:     maxD ← Max {delays[j]}∀j, 0 ≤ j ≤ q − 1
9:     if σ = 0 or maxD − minD ≤ σ then
10:        return v_c
11:        dv[v_c] ← maxD − minD                         ▷ record dv[v_c], to be used if no solution found
12:    else
13:        candidate[v_c] ← false                        ▷ σ not satisfied
14:    return −1
15: end function
```

The lowest expansion radius search round, r, is selected. The minimum cost node, u, of $Q[r]$ reflecting the lowest expansion radius node is selected and removed from $Q[r]$. $visited[r][u]$ is set to true and $explored[u]$ is incremented by one. If $explored[u] = q$ and $candidate[u] =$ true, Algorithm 4.2 is called. The algorithm examines if u is a valid core node. If u complies with both constraints or if the minimum inter-destination delay variation is set to zero ($\sigma = 0$), u is selected as the core node. $\sigma = 0$ denotes finding the lowest possible inter-destination delay variation tree. If u does not comply with both constraints, $candidate[u]$ is set to false. Next, all non-visited neighbors, v, of u are selected, pushed into $Q[r]$ with their expansion radius or cost, and their predecessor is set. If a shorter path to v is found, its cost and predecessor values are modified.

When all nodes are explored by all searches and a node complying with both constraints is not found, the minimum inter-destination delay variation node from dv is selected as the core node and Algorithm 4.3 is called. Algorithm 4.3 replaces

shortest paths with longer paths from v_c to some group members, to lower the inter-destination delay variation of the tree. If a solution is found, Tree T is constructed by connecting s to v_c and all nodes in M to v_c.

4.2.1 Time Complexity of DCS

Getting G' takes $O(E)$ time. Dijkstra from the source to all nodes in the network takes $O(E \log n)$ time. Selecting the next expansion radius round, r, takes $O(q)$ time. Selecting all expansion radius rounds takes $O(q^2 n)$ time, since it is performed at most qn times. Removing all nodes from Q takes $O(qE \log n)$ time, since removing a node from Q takes $O(\log n)$ time and at most nE nodes are removed. Inserting all nodes into the Q takes $O(qE \log n)$ time, since inserting a node into Q takes $O(\log n)$ and at most qE edges are relaxed. CF takes $O(q)$ time and is called at most n times (Algorithm 4.2). The tree generation takes $O(qn)$ time. Thus, the complexity of DCS is $O(E + E \log n + q^2 n + qE \log n + qE \log n + qn) = O(n^3 \log n)$ in the worst case.

4.3 Build Lower Variation Tree (BLVT) Algorithm

Build Lower Variation Tree (BLVT) algorithm replaces shortest paths from v_c to some group members with longer paths, to decrease the inter-destination delay variation of the generated trees. The lowest cost path member in T is selected and its shortest path is replaced with a longer path within the Δ and $maxCost$ values. This process is repeated until the lowest cost path cannot be replaced or all paths except the maximum delay path in T are replaced. *paths* and *costs* represent the paths and the costs of the paths in T, respectively (Algorithm 4.3).

Algorithm 4.3 Build Lower Variation Tree (BLVT)

```
1: function BLVT(v_c, s, Δ, M, pred)
2:     paths.push(getPath(s, v_c, pred[0]))
3:     delays.push(cost[0][v_c])
4:     for i = 0 : q do
5:         delays.push(cost[i][v_c])
6:         paths.push(getPath(v_c, M[i], pred[i + 1]))
7:     i ← index of node with maximum delay value in delays
8:     maxD ← delays[i]
9:     replaced[i] ← true                          ▷ not to increase the maxD
10:    maxCost ← maxD − cost[0][v_c]               ▷ the maximum allowed cost
11:    while currentReplaced do
12:        j ← index of a non-replaced path member with minimum value in delays
13:        if j = −1 then break                    ▷ all paths replaced
14:        else
15:            kpaths ← k-shortest-paths from j to v_c with cost ≤ maxCost
16:            if kpaths = ∅ then
17:                currentReplaced ← false         ▷ the minimum cost path cannot be re-
                                                     placed
18:            else
19:                maxP ← maximum cost path of kpaths
20:                paths[j] ← maxP
21:                delays[j] ← maxP.cost
22:                replaced[j] ← true
23:        for i = 0 : paths do
24:            T.push(makePair(paths[i], delays[i]))
25:    return T
26: end function
```

The algorithm selects the index i of the member with maximum path length and sets $replaced[i]$ to true, to disable increasing the $maxD$ of T. The $maxCost$ value is set, which is the bound on path length. Path replacement is achieved by generating k-shortest-paths from v_c to the lowest cost path member without passing through any of the nodes already in T. k is the number of edges in the shortest path to be extended [70]. If paths are found, the highest cost path of the k paths is selected and $paths[j]$, $delays[j]$, and $replaced[j]$ are updated. If a path is not found, the dv cannot be improved since a selected path cannot be replaced.

4.3.1 Time Complexity of BLVT

Generating *paths* and *costs* takes $O(qn)$ time. Selecting indices i and j takes $O(q)$ time. Selecting k-shortest-paths takes $O(knE \log n)$ time, since it makes kl calls to Dijkstra to generate the spur paths where l is the length of spur paths and can at most be n. The algorithm tries to replace at most $q-1$ paths. Consequently, DCSBLVT has $O(qknE \log n)$ time complexity.

4.4 Dynamic DCS and DCSBLVT

In the dynamic reorganization approach designed, reconstruction of the multicast tree is avoided by adding a new path, without effecting any of the paths from s to nodes in M. However, if adding a new path does not satisfy both constraints, reconstruction cannot be avoided. Algorithm 4.4 takes a request r, a node u, and an algorithm A and tries to restructure the multicast tree before reconstructing. A is DCS or DCSBLVT and k is 1 when A is DCS. Assume T is the multicast tree and M is the multicast group of the current multicast session and because of a dynamic request, the new multicast group is M' and tree is T'. When a node $u \in M$ issues a leave request, $M' = M$, $T' = T$, and the index i of u in M' is selected. Nodes in M' are shifted to remove u.

- If u is not v_c, the unique path u takes in T' is removed. In Algorithm 4.4, the paths taken in T' are shifted to remove the path taken by u and the size of T' is updated. If u is the minimum or maximum path length node in T', the dv and $maxD$ of T' changes.

- If u is v_c, u stops forwarding packets to its local processor. The dv of T' changes.

Algorithm 4.4 Dynamic DCS and DCSBLVT

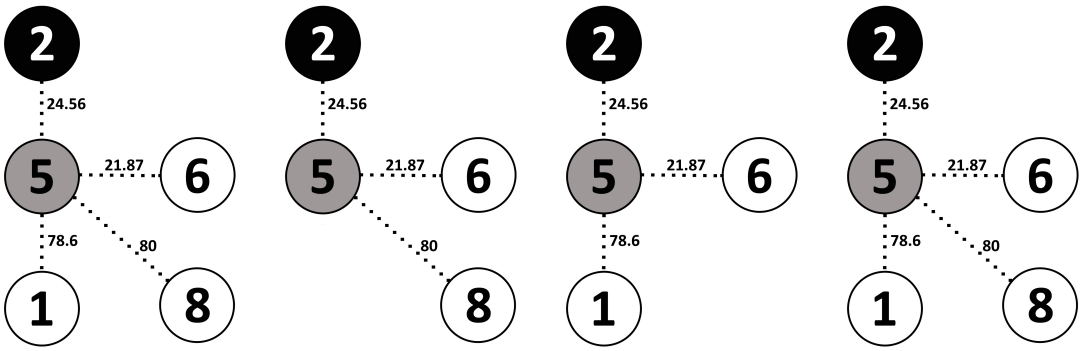
```
1: function DynamicReorganization(G ,v_c, s, Δ, M, T, r, u, A, k)
2:     T' = T
3:     M' = M
4:     if r.equals("leave") then
5:         i ← index of u in M'
6:         for j = i : q do                    ▷ T' has q + 1 paths. Removing u.
7:             M'[j] = M'[j + 1]
8:         resize M'
9:         if u! = v_c then
10:            for j = i + 1 : q do
11:                T'[j] = T'[j + 1]
12:            resize T'
13:        else
14:            u stops forwarding packets to its local processor
15:    else
16:        M' ← M' ∪ u
17:        costToCore ← cost from s to v_c
18:        maxCost = Δ − costToCore
19:        if u is not on a path from v_c to a group member then
20:            kpaths ← k-shortest-paths from v_c to u with cost ≤ maxCost
21:            nP ←minimum variation path of kpaths not violating σ
22:            if nP = ∅ then
23:                T' ←A(M', s, G, σ, Δ)              ▷ re-execution
24:            else
25:                T'.push(makePair(nP, nP.cost))     ▷ new path added
26:        else
27:            nP ← getPath(s, v_c) + getPath(v_c, u)
28:            if u receiving from nP does not violate σ then
29:                T'.push(makePair(nP, nP.cost))
30:            else
31:                T' ← A(M', s, G, σ, Δ)             ▷ re-execution
32:    return T', M'
33: end function
```

Given $M = \{6, 1, 8, 5\}, s = 2, \sigma = 81$, and $\Delta = 200$, Figure 4.1 (a) shows T with $v_c = 5$, $dv = 80$, and $maxD = 104.56$. A dotted link between two nodes is not a direct link, indicating the presence of relay nodes. If node 1 issues a leave request, the path taken by 1 is removed from T', the dv and $maxD$ of T' stay

unaltered (Figure 4.1 (b)). If node 8 issues a leave request, the path taken by 8 is removed from T'. Since 8 was the node with maximum path length in T, T' has $dv = 78.6$ and $maxD = 103.16$ (Figure 4.1 (c)). If node 5, which is the core node, issues a leave request, it stops forwarding packets to its local processor. T' has $dv = 58.13$ and $maxD = 104.56$ (Figure 4.1 (d)).

When a node u issues a join request, u is added to M', the delay from s to v_c ($costToCore$) is selected. $maxCost$ is the maximum allowed path length for u to receive the message through.

- If u is not on a path from v_c to a group member in T' and $A = DCSBLVT$, the algorithm finds k-shortest-paths from v_c to u without passing through any of the nodes already in T'. Path lengths are bounded by $maxCost$. The minimum inter-destination delay variation path, nP, not violating the σ constraint is selected. If $A = DCS$, the algorithm finds the shortest path sP from v_c to u. If sP does not violate both σ and Δ constraints, it is saved in nP.
  - If $nP = \emptyset$, A is re-executed to select a new core and construct a new multicast tree.



(a) T (b) T' after 1 issues a leave request (c) T' after 8 issues a leave request (d) T' after 5 issues a leave request

Figure 4.1: Dynamic leave request examples

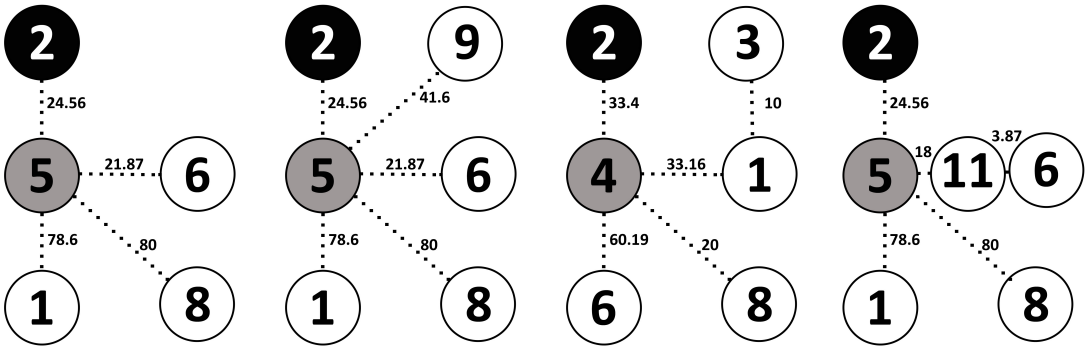
- Otherwise, nP is added to T'.

- If u is on a path from v_c to a group member in T', nP, the path from v_c to u added to the path from s to v_c, is selected.
  - If σ constraint is satisfied, u starts to forward multicast packets to its processor.
  - Otherwise, A is re-executed to select a new core and construct a new multicast tree.

  Given $M = \{6, 1, 8\}, s = 2, \sigma = 63$, and $\Delta = 140$, Figure 4.2 (a) shows T with $v_c = 5$, $dv = 58.13$, and $maxD = 104.56$. If 9 issues a join request, the shortest path nP from $v_c = 5$ to 9 is found. Since nP does not violate Δ and σ constraints, it is added to T'. The dv and $maxD$ of T' stay unaltered (Figure 4.2 (b)). If 3 issues a join request, the shortest path from v_c to 3 violates the Δ constraint. A is re-executed and T' is generated with $v_c = 4$, $dv = 40.19$, and $maxD = 93.59$ (Figure 4.2 (c)). If 11 issues a join request, since it is already on the path sending the message from v_c to group member 6 and this path delay satisfies the Δ and σ constraints, node 6 starts receiving the message through the path already existing in T'. The $dv = 62$ and $maxD = 104.56$ (Figure 4.2 (d)).

4.4.1 Time Complexity of Dynamic DCS and DCSBLVT

When a node u issues a leave request, getting the index of u in M' takes $O(q)$ time, shifting and resizing also take $O(q)$ time each. Thus, the time complexity of a leave request is $O(q)$ (Algorithm 4.4). When a node u issues a join request, checking if a node is on a path from v_c to a group member takes $O(qn)$ time. If u is not on a path from v_c to a group member and $A = DCS$, finding the shortest path, nP, from v_c to



(a) T (b) T' after 9 issues a join request (c) T' after 3 issues a join request (d) T' after 11 issues a join request

Figure 4.2: Dynamic join request examples

u takes $O(E \log n)$ time. If adding nP does not violate the σ and Δ constraints, the time complexity of the join operation is $O(qn + E \log n) = O(E \log n)$. Otherwise, if adding nP violates the σ constraint, DCS is re-executed which takes $O(n^3 \log n)$ time. If u is not in T and $A = DCSBLVT$, finding k-shortest-paths takes $O(knE \log n)$. Selecting nP takes $O(k)$ time. If adding nP does not violate the σ constraint, the time complexity of the join operation is $O(qn + k + knE \log n) = O(knE \log n + qn)$. Otherwise, if adding nP violates the σ constraint, DCSBLVT is re-executed which takes $O(qknE \log n)$ time. If u is on a path from v_c to a group member and $A = DCS$, retrieving nP takes $O(n)$ time. If nP does not violate the σ constraint, the time complexity of the join operation is $O(qn + n) = O(qn)$. Otherwise, if nP violates the σ constraint, DCS is re-executed which takes $O(n^3 \log n)$ time. If u is on a path from v_c to a group member and $A = DCSBLVT$, retrieving path nP takes $O(n)$ time. If nP does not violate the σ constraint, the time complexity of the join operation is $O(qn + n) = O(qn)$. Otherwise, if nP violates the σ constraint, DCSBLVT is re-executed which takes $O(qknE \log n)$ time.

4.5 Multi-Core DVBMT (MCDVBMT) Algorithm

Phase one of Multi-Core Delay Variation Bound Multicast Trees (MCDVBMT) (Algorithm 4.5) calls Candidate Core Nodes Generation Satisfying End-to-end Delay (CCNG), to get candidate nodes and costs satisfying the Δ constraint. The algorithm also returns predecessors of nodes on shortest paths found (Algorithm 4.6). Phase two calls Candidate Core Nodes Elimination (CCNE), to get revised candidate nodes and costs satisfying the σ constraint (Algorithm 4.7). Phase three calls Select Cores (SC), to select core nodes C (Algorithm 4.8). d is the maximum allowed covering degree of candidate nodes.

Algorithm 4.5 Multi-Core DVBMT (MCDVBMT)

1: **function** MCDVBMT(G, s, M, Δ, σ, d)
2: $cN, cC, pred \leftarrow$ CCNG(G, s, M, Δ)
3: $cN, cC \leftarrow$ CCNE(cN, cC, σ)
4: $C, core \leftarrow$ SC(cN, cC, M, d)
5: **for** $i = 0 : q - 1$ **do**
6: T.push(getPath($s, core[i], pred[0]$) $\cup$ getPath($core[i], M[i], pred[i+1]$))
7: **return** C, T
8: **end function**

4.5.1 Candidate Core Nodes Generation Satisfying End-to-end Delay (CCNG)

CCNG creates cN, cC, and $pred$. $cN[i]$ holds the candidate core nodes satisfying the Δ constraint for group member i, and $cC[i]$ their corresponding costs. $pred[0]$ holds predecessors of nodes on shortest paths from s to all nodes and $pred[i+1]$ from group member $i \mid 0 \leq i \leq q-1$ to all nodes (Algorithm 4.6).

The algorithm starts by finding the shortest path from s to every node in G. Next, the transpose of directed graph $G(V, E)$, graph $G'(V, E')$ such that $\forall e \in E \mid e = (u, v)$,

$e' = (v, u) \in E'$, is generated. The shortest path from every i group member to all nodes is found and cN and cC are generated. For every group member, every node in G except the source node are checked if it can serve as a candidate core node. A node j in G is a candidate core node to a group member i, if $cost(s, j) + cost(j, i) \leq \Delta$.

4.5.2 Candidate Core Nodes Elimination (CCNE)

CCNE algorithm reduces the cardinality of candidate core nodes of group members, to lower the maximum variation among all candidate core node costs. The algorithm first sorts every multicast group member candidate core nodes, in increasing order of costs. If one of the multicast group members does not have a candidate core node, the algorithm terminates, no solution can be found. The following steps are repeated until the variation among the costs in cC is less than σ or the size of the q candidate core node vectors is 1. This reflects the semi-matching of every q multicast group member to a core node (Algorithm 4.7).

Algorithm 4.6 Candidate Core Nodes Generation Satisfying End-to-end Delay (CCNG)

```
1: function CCNG(G, s, M, Δ)
2:     cost[0], pred[0] ← Dijkstra(s, G)
3:     G' ← transpose G
4:     for i = 1 : q do
5:         cost[i], pred[i] ← Dijkstra(M[i − 1], G')
6:     for i = 0 : q − 1 do
7:         for j = 0 : n − 1 do
8:             if j! = s then
9:                 delay ← cost[0][j] + cost[i + 1][j]
10:                if delay <= Δ then
11:                    cN[i].push(j)
12:                    cC[i].push(cost[0][j] + cost[i + 1][j])
13:    return cN, cC, pred
14: end function
```

The lowest cost candidate core node of every group member is taken and pushed into a vector, $costsMin$. Similarly, the highest cost candidate core node of every group member is taken and pushed into a vector, $costsMax$. If the variation of costs in $costsMin$ is greater than that of $costsMax$, the minimum cost node of all candidate nodes is removed from cN and cC. If the variation of costs in $costsMax$ is greater than that of $costsMin$, the maximum cost node of all candidate nodes is removed from cN and cC. With every elimination of a candidate core node, the overall variation between all core nodes decreases.

Algorithm 4.7 Candidate Core Nodes Elimination (CCNE)

```
1: function CCNE(cN, cC,σ)
2:     for i = 0 : q − 1 do
3:         if |cN[i]| = 0 then return
4:         sort(cN[i], cC[i])
5:     while true do
6:         matched ← 0
7:         for i = 0 : q − 1 do
8:             if |cN[i]| = 1 then matched + +
9:         var ← delay variation of costs in cC
10:        if var ≤ σ or matched = q then break
11:        for i = 0 : q − 1 do
12:            costsMin.push(cC[i][0])
13:            costsMax.push(cC[i][|cC[i]| − 1])
14:        maxMin ← max(costsMin)
15:        minMin ← min(costsMin)
16:        maxMax ← max(costsMax)
17:        minMax ← min(costsMax)
18:        minSideDv ← maxMin − minMin
19:        maxSideDv ← maxMax − minMax
20:        if minSideDv > maxSideDv then
21:            remove minMin node from cN and its cost from cC
22:        else
23:            remove maxMax node from cN and its cost from cC
24:    return cN, cC
25: end function
```

4.5.3 Select Cores (SC)

This phase tries to find a semi-matching in G. If only a single node w can cover a group member i, w is added to C, i is marked as matched, the cost and cover node of i is updated, the number of group members matched m is incremented, and $cN[i]$ and $cC[i]$ are cleared. Next, if w is a group member, it is marked as matched, m is incremented, and its cost and cover node are set. The generated $adjN$ and $adjC$, represent for every candidate core node the list of group members it can cover and their costs, respectively. The size of $adjN$ and $adjC$ is at most n and every candidate node can cover at most q members. Next, every non-matched member a node in C can cover is marked as matched, m is incremented, costs and cover nodes are set, and is removed from $adjN$ and $adjC$ (Algorithm 4.8).

Algorithm 4.8 Select Cores (SC)

```
1: function SC(cN, cC, M, d)
2:     U ← M, W ← V − {s}
3:     for i = 0 : q − 1 do
4:         if |cN[i]| = 1 then
5:             w ← cN[i][0], matched[i] ← true, m++
6:             set cost and core of M[i]
7:             if w ∉ C then C.push(w)
8:             if w ∈ M and !matched[w] and w ∈ cN[w] then
9:                 matched[w] ← true, m++
10:                set cost and core of w
11:            clear cN[i], cC[i]
12:    generate adjN and adjC from cN and cC
13:    for every node c_i in C do
14:        for every node u_i, c_i is linked to do
15:            if !matched[u_i] then
16:                matched[u_i] ← true, m++
17:                set cost and core of u_i
18:        clear W[c_i], adjN[c_i], adjC[c_i]
19:    remove matched members from adjN and adjC
20:    sort W, adjN, adjC
21:    adjNT ← adjN, adjCT ← adjC, WT ← W
```

```
22:     for i = 0 : |W| − 1 do N ← cN[i]
23:         if |N| >= d or |N| = 0 then remove W[i]
24:     while true do
25:         if m = q then break
26:         if |WT|! = 0 then
27:             while m! = q and |WT|! = 0 do
28:                 sort WT, adjNT, adjCT
29:                 w ← WT[0]
30:                 for every node u_i, w is linked to do
31:                     if !matched[u_i] then
32:                         matched[u_i] ← true, m + +
33:                         set cost and core of u_i
34:                 if w ∈ M and !matched[w] and w ∈ cN[w] then
35:                     matched[w] ← true, m + +
36:                     set cost and core of w
37:                 C.push(w)
38:                 clear WT[0], adjNT[0], adjCT[0]
39:                 remove matched members from adjNT and adjCT
40:             if m = q then break
41:         adjNT ← adjN, adjCT ← adjC, WT ← W
42:         while m! = q do
43:             execute steps 29 − 41
44:     return C, core
45: end function
```

W, $adjN$, and $adjC$ are sorted in decreasing order of covering degree. To select an acceptable number of candidate core nodes, candidate nodes covering zero or more than d members are removed. Copies of W, $adjN$, and $adjC$ are taken in WT, $adjNT$, and $adjCT$ to be used if removal of nodes with degree greater than d leads to failure to find a semi-matching. The node w with the largest degree is removed from W and added to C. The cost and cover node of all nodes w can cover is updated. Nodes covered by w are removed from $adjN$ and $adjC$. This ends the first iteration of the algorithm, and is repeated until all q group members are matched or W is empty.

If W is empty and not all nodes are matched, removal of candidate nodes of cover size greater than d, has led to failure to find a semi-matching. Thus, a candidate core

node of cover size greater than d is added to C until all nodes are matched.

4.5.4 Time Complexity of MCDVBMT

Dijkstra from s to all nodes in G takes $O(E \log n)$ time. Getting G' takes $O(E)$ time. Dijkstra from all q group members to all nodes in G takes $O(qE \log n)$ time. Generating cN and CC takes $O(qn)$ time. Thus, the time complexity of CCNG is $O(qE \log n)$.

Sorting cN and cC takes $O(qn \log n)$ time. Checking if a group member has no candidate core nodes takes $O(q)$ time. Checking the number of candidate core nodes of every group member takes $O(q)$ time. Getting the variation takes $O(q)$ time. Pushing nodes to $costsMin$ and $costsMax$ also takes $O(q)$ time. Finding min and max values for both candidate sets takes $O(q)$ time and is repeated at most $|q(n-1)|$ times. Removing a node from the end of cN and cC takes $O(1)$ time. Removing a node from the front of cN and cC takes $O(n)$ time. At most $|q(n-1)|$ nodes are removed. Removal of nodes takes at most $O(qn^2)$ time, if all nodes are removed from the end. Thus, the time complexity of CCNE is $O(qn^2)$.

Generating U takes $O(qn)$ time. Generating W takes $O(n^2)$ time. Adding multicast group member core nodes with single candidate node to C and checking if the added core is a multicast group member takes $O(q^2)$ time. Generating $adjN$ and $adjC$ takes $O(qn)$ time. Sorting W, $adjN$, and $adjC$ takes $O(n \log n)$ time. Checking if nodes in C cover other multicast group members and removing matched members takes $O(q^2n + qn)$ time. Checking and removing candidate nodes with degree zero or greater than d from W takes $O(n^2)$ time. The next steps are repeated until all multicast group members are matched. If every iteration covers only one group member, the steps are repeated at most q times. Sorting W, $adjN$, and $adjC$ takes $O(qn \log n)$ time. Removing a node from W together with its $adjN$ and $adjC$ takes

$O(qn)$ time. Removing matched multicast group members takes $O(q^2n)$ time, since at most qn nodes are removed and removing a node takes $O(q)$ time. Checking if the just added core node is a multicast group member and updating it takes $O(q^2)$ time. When all points are not matched and size of W is zero, the same steps are repeated on $WT, adjNT$, and $adjCT$. Thus, the time complexity of SC is $O(q^2n)$.

Thus, the time complexity of MCDVBMT is $O(qE \log n + q^2n + q^2n) = O(n^3 \log n)$.

4.6 Dynamic MCDVBMT

In dynamic DVBMT nodes join and leave the multicast group during the life of the multicast session. The multicast tree should be updated in response to changes in multicast group members, without violating both end-to-end delay and inter-destination delay variation constraints. Nodes in M may leave the group after issuing a leave request, while nodes that want to join an ongoing multicast session issue a join request.

Dynamic DVBMT can be tackled by re-executing the algorithm after every dynamic request to generate a new tree and use it for routing subsequent packets. However, this method is very costly as the algorithm is re-executed, old paths are removed, and new paths are formed.

Let T be the set of multi-core rooted trees generated by MCDVBMT of the ongoing multicast session with multicast set M and suppose that because of a join or leave request, the new multicast set is M', the set of multi-core rooted trees is T', and set of core nodes is C'.

The proposed dynamic organization algorithm takes the source node s, the trees T, the group members M, the cores C, the core node of every multicast group member *core*, Δ, σ of the ongoing multicast session, a request r, and a node u generating the request and first sets $M' = M$, $T' = T$, and $C' = C$ (Algorithm 4.9). Reconstruction is avoided by augmenting new paths or using already existing paths without affecting

any of the paths from s to nodes in M not affected by the request. However, if reorganization is not possible, the algorithm is re-executed to generate new cores and trees.

Algorithm 4.9 Dynamic MCDVBMT

```
1: function DO(G, s, M, Δ, σ, C, core, T, r, u)
2:     T' = T, M' = M, C' = C
3:     if r.equals("leave") then
4:         remove u from M', c_i = core[u]
5:         remove core of u from core
6:         if u ∉ C' then
7:             remove the path u takes from T'
8:             if c_i covers only u then
9:                 remove c_i from C'
10:                remove path s to c_i from T'
11:        else
12:            u stops forwarding messages to its local processor
13:    else
14:        M' ← M' ∪ u
15:        if u ∉ T' then
16:            pStoC, cStoC ← paths and costs from s to every node in C'
17:            G' ← transpose(G)
18:            cost[1], pred[1] ← Dijkstra(u, G')
19:            for i = 0 : |C'| − 1 do
20:                paths.push(pStoC[i]∪ path(u, C'[i])
21:                costs.push(cStoC[i] + cost[1][C'[i]])
22:            select paths, costs satisfying σ and Δ
23:            if paths = ∅ then re-execution
24:            else
25:                select minimum variation path p ∈ paths add to T'
26:        else
27:            paths, costs ← paths and costs existing in T' from nodes in C' to u
28:            record nodes in C' from where u is not reachable in C''
29:            select paths, costs satisfying σ and Δ
30:            if paths = ∅ then
31:                execute steps 16 − 25 on C''
32:            else
33:                select minimum variation path p ∈ paths add to T'
34:    return T', M'
35: end function
```

When a node $u \in M$ issues a leave request, u is removed from M'. Next, the core node, c_i, covering u is selected and the core node of u is removed from *core*.

- If $u \notin C'$, the path u takes is removed from T'. If c_i does not cover other members, it is removed from C' and the path s to c_i is also removed from T'. If u was the minimum or maximum path length node in T', the dv and $maxD$ of T' changes.

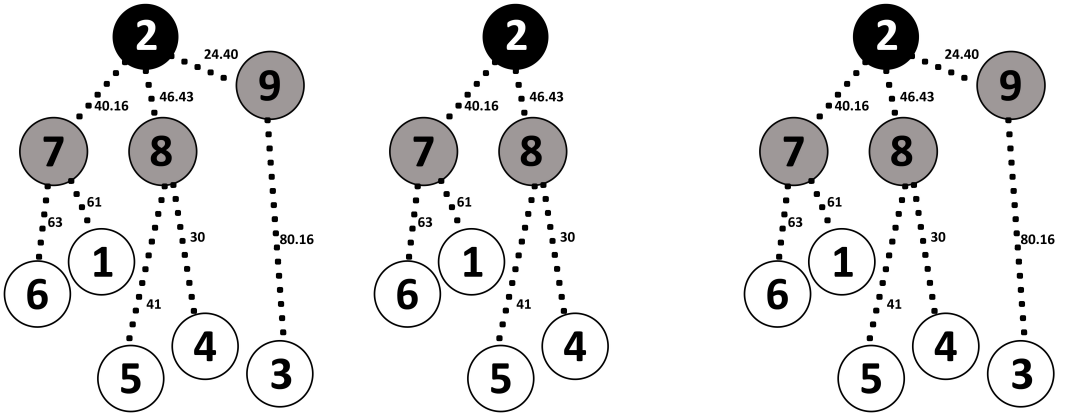
- If $u \in C'$, u stops forwarding messages to its local processor. The dv of T' changes.

Given $M = \{6, 1, 8, 5, 4, 3\}, s = 2, \sigma = 62$, and $\Delta = 140$, Figure 4.3 (a) shows T with $C = \{7, 8, 9\}$, $core = \{7, 7, 8, 8, 8, 9\}$, $dv = 58.13$, and $maxD = 104.56$. A Dotted link between two nodes is not a direct link, indicating the presence of relay nodes. If node 3 issues a leave request, it is removed from M' and its core from *core*. The path it takes is also removed from T'. The cover node, 9, covering 3 is selected. 9 does not cover any other nodes, so it is removed from C' together with the path it takes from T'. Since 3 was the node with maximum path length in T', the updated T' has $dv = 56.73$ and $maxD = 103.16$ (Figure 4.3 (b)). If node 8 issues a leave request, it is removed from M' and *core*. Since 8 is a core node, it stops forwarding packets to its local processor. The delay variation of T' changes to 28.13 (Figure 4.3 (c)).

When a node u issues a join request, it is added to M'.

- If $u \notin T'$, the algorithm selects the paths and costs from s to all core nodes in $pStoC$ and $cStoC$, respectively. The transpose of G, G', is generated and shortest path from u to all core nodes in G' is found. Next, *paths* and *costs* from s to u by passing through every node in C' and not violating the Δ and σ constraints are generated.

  - If $paths = \emptyset$, MCDVBMT is re-executed.



(a) T (b) T' after 3 issues a leave request (c) T' after 8 issues a leave request

Figure 4.3: Dynamic MCDVBMT leave request examples

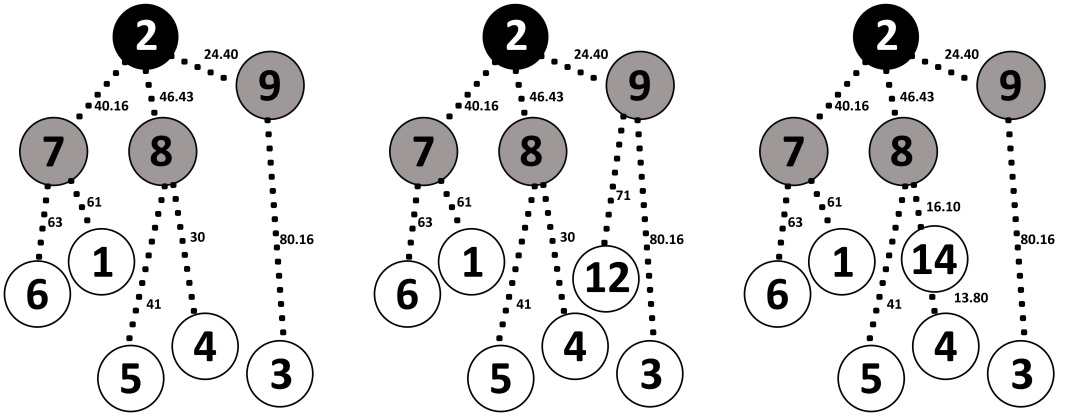
  - Otherwise, the lowest variation path p from $paths$ is added to T'.

- If $u \in T$, the algorithm selects the $paths$ and $costs$ from s to u already in T' by passing through every node in C' and not violating both constraints. It also selects C'' core nodes from where u cannot be reached.

  - If $paths = \emptyset$, the same steps when $u \notin T'$ are performed but on C'' instead of C'

  - Otherwise, the lowest variation path p from $paths$ is added to T'.

Given $M = \{6, 1, 8, 5, 4, 3\}, s = 2, \sigma = 62$, and $\Delta = 140$, Figure 4.4 (a) shows T with $C = \{7, 8, 9\}$, $core = \{7, 7, 8, 8, 8, 9\}$, $dv = 58.13$, and $maxD = 104.56$. If node $12 \notin T$ issues a join request, 9 is chosen as its core node. The dv and $maxD$ of T' are unaltered (Figure 4.4 (b)). If node $14 \in T$ issues a join request, 14 is already on the path from core node 8 to group member 4. Since the path satisfies both constraints, 8 is set as the core node of 14. The dv and $maxD$ of T' are unaltered (Figure 4.4 (c)).



(a) T (b) T' after 12 issues a join request (c) T' after 14 issues a join request

Figure 4.4: Dynamic MCDVBMT join request examples

4.6.1 Time Complexity of Dynamic MCDVBMT

When node u issues a leave request, removing u from M', selecting and removing core of u from *core* takes $O(q)$ time. Checking if u is a core node takes $O(q)$ time. Removing $T'[u]$ takes $O(q)$ time. Checking if the core node of u covers other multicast group members and removing them takes $O(q)$ time. Thus, a leave operation takes $O(q)$ time.

When a node u issues a join request, checking if a node is in T', takes $O(qn)$ time.

If $u \notin T'$, selecting paths from s to all core nodes in T' takes $O(q(n + |E|))$ time. Getting the transpose of G takes $O(|E|)$ time. Finding shortest paths from u to core nodes takes $O(E \log q)$ time. Generating *paths* and *costs* takes $O(qn)$ time. Selecting paths not violating Δ and σ constraints takes $O(q)$ time. If none of the paths satisfy both constraints, MCDVBMT is executed which takes $O(n^3 \log n)$ time, otherwise the lowest path selection takes $O(q)$ time. Thus, when $u \notin T'$, a leave request takes $O(n^3 \log n)$ or $O(n^3)$.

If $u \in T'$, for every core node selecting and generating paths takes $O(n^3)$. Selecting paths not violating Δ and σ constraints takes $O(q)$ time. If none of the paths satisfy

both constraints, the steps executed next as stated above, takes $O(n^3)$ or $O(n^3 \log n)$ time. Otherwise the lowest path selection takes $O(q)$ time. Thus, when $u \in T'$, a leave request takes $O(n^3 \log n)$ or $O(n^3)$.

Chapter 5

Numerical Results

The performance of the MPAIR, MDIAG, MDIST, and MMIN is evaluated through simulations in a 20 $\times$ 20 mesh and torus for $20 \leq q \leq 360$. Multicast sets are randomly generated. 100 runs are performed for each algorithm for the same set of destination nodes. After generating a T, the problem of multicast in the nD mesh or torus is transformed into broadcast in the T. Thus, the broadcast time of the T is calculated using the algorithm proposed by Slater et al. that reflects the time in hops needed to perform the broadcast communication [58].

To compare the time generated by the four algorithms, the number of times an algorithm generates an extra time unit, reflecting a loss is counted. This loss in time is only in one time unit. To compare traffic, the average traffic generated is considered.

The time and traffic of PAIR with MPAIR, DIAG with MDIAG, DIST and MDIST, and MIN and MMIN is compared. Since DIAG and MPAIR are pro-traffic, the traffic they generate is compared. The average time and traffic generated by the modified algorithms in 2D mesh and torus networks is compared.

Next, the performance of HMDIAG is compared with TASNEM [19] and M-HCM [64] in a 2D torus with $k_0 = k_1 = 40$. The routing performance under various multicast set sizes is explored starting from 40 destination nodes to 1560. Multicast

sets are randomly generated and 100 runs are performed for the same cardinality of destination node set for each algorithm.

The communication latency consists of three parts: startup, network and blocking latency. Startup latency is the time required to start a message, including operating system overheads. Network latency is the latency caused by propagation and router delays, blocking latency is the latency caused by delays due to contention in the network. Network latency depends on the traffic generated by the routing algorithm. The startup latency is in microseconds (ms) and network latency is in nanoseconds (ns) [48]. Consequently, researchers have focused on minimizing the startup latency since it dominates the communication latency. The network latency has a slight effect on the communication latency, except when the maximum path length to reach all destination nodes is very long. In addition, when the network traffic increases, the blocking time which is a function of path length may become considerable.

The startup latency is set to 0.5 μs. 0.3 μs correspond to message sending latency and 0.2 μs correspond to message receiving latency. The network latency is set to 25 ns. 5 ns correspond to link dissemination time and 20 ns correspond to router delays. These values reflect prevalent systems utilized values [19,65]. Blocking latency is ignored. Thus, multicast latency is $(500 \times NStartup) + (25 \times LPath)$, where $NStartup$ is the number of startups used by an algorithm and $LPath$ is the maximum path length to reach a destination node.

To reflect the parallelism achieved by the three algorithms, the coefficient variation of multicast time they generate is compared. It is calculated by $\frac{\sigma_{multicastTime}}{M_{nl}}$, where $\sigma_{multicastTime}$ is the standard deviation of the multicast time and M_{nl} is the mean multicast time [3]. A low variation indicates that destination nodes receive the multicast message in comparable arrival time. The traffic generated by the algorithms, which is the number of links in the T is marked.

The time, traffic, latency and variation generated by HMDIAG in 2D torus where $k_0 = k_1 = 40$ and 3D torus where $k_0 = k_1 = k_2 = 12$ for $40 \leq q \leq 1560$ is compared. The multicast and broadcast time of the generated multicast trees in 2D and 3D torus networks is also compared.

In the simulations for DVBM, graphs are randomly generated using Waxman model with $\alpha = 0.8$, $\beta = 0.7$, and average node degree 4 [67]. The position of nodes is fixed randomly in a grid of size 4000 km $\times$ 2400 km. The delay $D(u, v)$ between two nodes u and v is set to $l(u, v) \times L \times 20\ ms$, where $l(u, v)$ is the Euclidean distance between the nodes and L is the maximum possible distance between two nodes.

The performance of DCS and DCSBLVT is compared to KMK, KMKh, DDVCA, ESC, and ATabu. The values recorded are the average over 100 graphs for every n and q, where $20 \leq n \leq 120$ and $q = \{20\%$ of n, 50% of n, 80% of $n\}$.

The failure percentage, inter-destination delay variation, and maximum end-to-end delay of the seven algorithms is compared. The execution time and number of nodes explored by KMK and DCS is compared. $\Delta = 1.5 \times maxD$ of the shortest path tree rooted at s and $\sigma = p$ from KMK. To compare KMKh to DCSBLVT, results are recorded with $\sigma = 0$. When $\sigma = 0$, the algorithms return the tree with the smallest inter-destination delay variation they can find.

The approach proposed for dynamic DVBMT tree reorganization is studied. The initial tree is constructed using DCS or DCSBLVT. q join or leave requests are performed for each group. Requests are chosen as either a join or a leave request randomly. The percentage of algorithm re-execution compared to the total number of requests is given. Examples are also given on the change in inter-destination delay variation and end-to-end delay of the tree as nodes join or leave the group. One set of experiments are performed on DCS another on DCSBLVT.

Next, the performance of MCDVBMT is evaluated by comparing it to single-core

based algorithms including DDVCA, ESC, KMK, and DCS. Each point plotted or value in a table represents the average over 200 graphs for the stated n and q. q is set to 20%, 40%, 60%, and 80% of n where $20 \leq n \leq 100$. Δ is set to $1.5 \times maxD$ of the shortest path tree rooted at s to all group members. σ is set to p from KMK. The performance of MCDVBMT is also evaluated with three maximum allowed covering degree values for candidate core nodes. The average case behavior of the five algorithms in terms of failure percentage, inter-destination delay variation, maximum end-to-end delay, cost, and traffic concentration is studied.

The behavior of MCDVBMT to dynamic requests with d set to $0.75 \times q$, $0.5 \times q$, and $0.25 \times q$ is compared. Each value represents the average over 200 graphs. q join or leave requests are performed for each generated session. Requests are chosen as join or leave randomly. The percentage of trees that were re-executed before receiving all the q requests is presented. The percentage of re-executions compared to the total number of requests is also presented.

5.1 Results on MDIAG, MPAIR, MMIN, and MDIST

Table 5.1 shows that MDIAG generates one time unit less time than the time of DIAG in 25% of the cases when $q = 20$ and in 17% of the cases when $q = 40$. This value of gain in time decreases reaching zero when $q = 140$. However, although there is gain in time, MDIAG generates more traffic and the percent value increases with q until reaching a maximum value after which it starts to decrease. The maximum loss in traffic is 51% when $q = 80$.

Table 5.1: MDIAG and MPAIR in 2D mesh network

q	**DIAG to MDIAG gain in time unit**	**DIAG to MDIAG traffic loss**	**PAIR to MPAIR loss in time unit**	**PAIR to MPAIR traffic gain**
20	25%	24%	31%	22%
40	17%	42%	24%	35%
60	7%	48%	14%	37%
80	6%	51%	11%	38%
100	1%	50%	5%	37%
120	1%	48%	5%	36%
140	0%	46%	2%	35%
160	0%	40%	1%	32%
180	0%	39%	1%	31%
200	0%	34%	0%	28%
240	0%	27%	0%	24%
280	0%	20%	0%	19%
320	0%	14%	0%	13%
360	0%	7%	0%	7%

In summary, MDIAG runs in $O(qn)$ time compared to DIAG with run time $O(qn^2)$, making MDIAG a good candidate algorithm for pro-time applications since the time is optimal or optimal plus one and the average loss in traffic is approximately 35%.

MPAIR generates one extra time unit more than the time of PAIR in 31% of the cases when $q = 20$ and in 24% of the cases when $q = 40$. This value decreases reaching zero when $q = 200$. However, although there is loss in time when q is small, there is gain approximately 34% in traffic. The gain in traffic increases with q and reaches its maximum value, 38%, when $q = 80$.

In summary, MPAIR and PAIR have the same time complexity $O(qn)$. Therefore, MPAIR is a viable candidate algorithm for pro-traffic applications since on average there is approximately 28% gain in traffic and loss in a time unit in 7% of the cases. Moreover, when q is greater than or equal to 200 there is no loss in time (Table 5.1).

MDIST generates less time than DIST in 72% of the cases when $q = 20$. This percent gain in time decreases as q increases. The maximum gain in time is 14 units at $q = 20$. However, this gain occurs in expense of traffic. For example, at $q = 20$ there is an average increase of 16 units of traffic in 96% of the cases. The maximum loss in traffic units is 16 when $q \leq 80$. When $q > 80$, the loss in traffic starts to decrease. This is mainly because as q increases the traffic would generally increase, and most of the links in the mesh would be in the T.

In summary, MDIST has lower complexity $O(qn)$ than the complexity of DIST which is $O(qn^3)$. Therefore, MDIST is a suitable candidate algorithm for pro-time applications since on average there is approximately 6.5 time units gain and on average 15 traffic units loss (Table 5.2).

MMIN generates less time than MIN in 70% of the cases when $q = 20$. This percent gain in time decreases as q increases. The maximum gain in time is 10 units when $q \leq 140$, after which it starts to decrease. However, this gain occurs in expense

of traffic. For example, at $q = 20$ there is an average increase of 6 units of traffic in 82% of the cases. The maximum loss in traffic is 6 when $q \leq 80$. When $q > 80$, the loss in traffic starts to decrease.

In summary, MMIN has lower complexity $O(qn)$ than the complexity of MIN which is $O(qn^3)$. Therefore, MMIN is a viable candidate algorithm for pro-time applications since on average there is approximately 5 time units gain and on average 4 traffic units loss (Table 5.3).

Table 5.2: MDIST in 2D mesh

q	**DIST to MDIST time gain %**	**DIST to MDIST max time gain**	**DIST to MDIST traffic loss %**	**DIST to MDIST average traffic loss**
20	72%	14	96%	16
40	72%	13	98%	16
60	71%	11	98%	16
80	71%	9	98%	16
100	64%	9	99%	12
120	62%	9	100%	12
140	60%	8	98%	9
160	57%	8	97%	8
180	47%	6	96%	7
200	47%	6	89%	5
240	31%	4	75%	3
280	28%	4	54%	2
320	16%	2	20%	2
360	0%	0	4%	2

Table 5.3: MMIN in 2D mesh

q	MIN to MMIN time gain %	MIN to MMIN max time gain	MIN to MMIN traffic loss %	MIN to MMIN average traffic loss
20	70%	10	82%	6
40	69%	10	83%	6
60	67%	8	94%	6
80	55%	8	98%	6
100	55%	6	98%	5
120	55%	6	98%	5
140	48%	6	98%	4
160	30%	5	89%	4
180	29%	5	85%	3
200	22%	5	85%	3
240	4%	2	42%	2
280	2%	2	20%	1
320	0%	0	2%	1
360	0%	0	2%	1

Figure 5.1 shows the average traffic of the four algorithms in a 2D mesh. MPAIR and MMIN generate the least traffic.

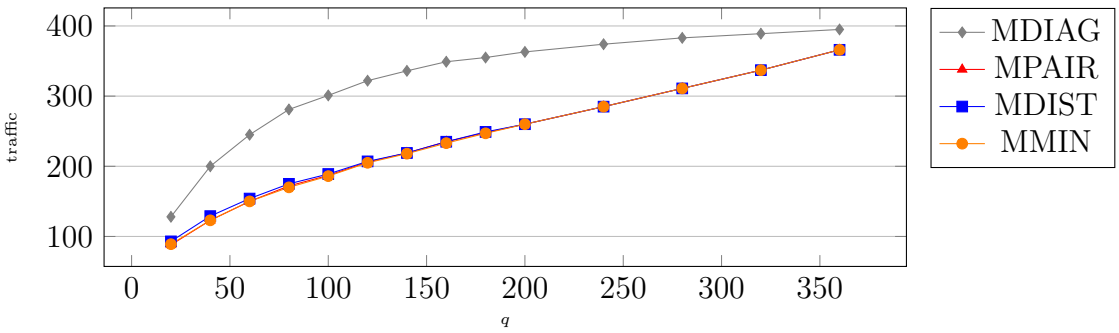


Figure 5.1: Average traffic of the modified algorithms in 2D mesh

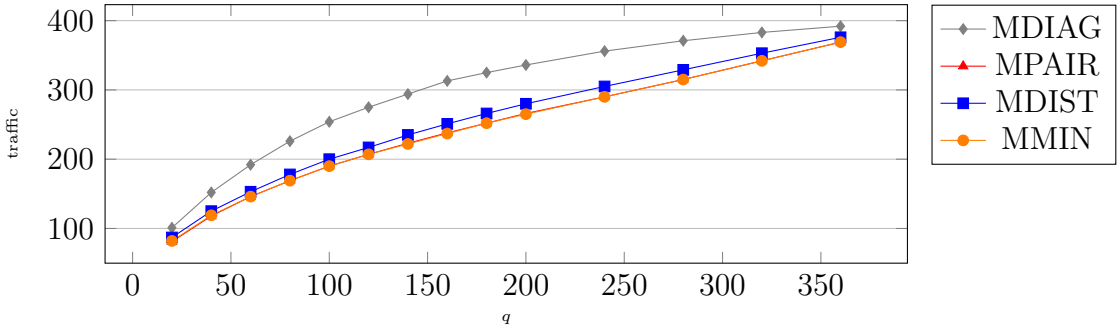


Figure 5.2: Average traffic of the modified algorithms in 2D torus

Figure 5.2 shows the average traffic of the four algorithms in a 2D torus. MPAIR and MMIN generate the least traffic.

Figure 5.3 shows the time increase of MPAIR, MDIST and MMIN from DIAG. Although MMIN generates the least traffic, it increases the time the most.

To further compare the performance of MPAIR and MMIN, the average variance of multicast latency they generate is compared. Figure 5.4 and Figure 5.5 show the average variance of multicast latency of MPAIR and MMIN in 2D mesh and torus networks. MPAIR in 2D mesh and torus networks has less average variance of multicast latency than MMIN, showing that MPAIR is more parallel than MMIN because of the intermediate nodes. In addition, the average variance of multicast latency in a 2D mesh for both algorithms is less than half of its average value in a 2D torus.

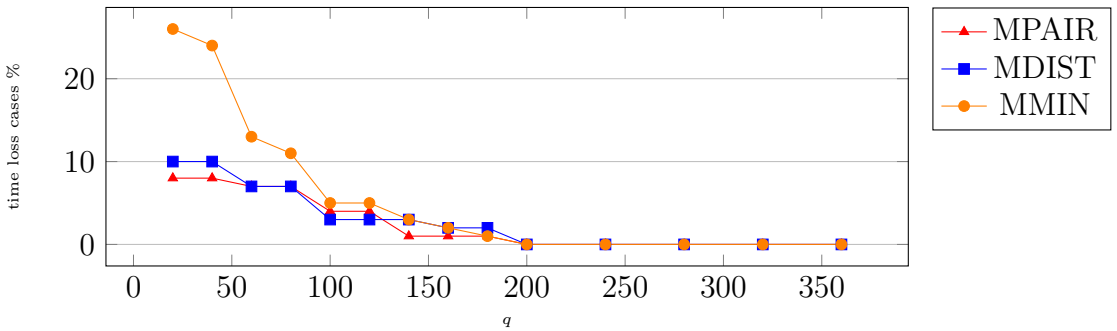


Figure 5.3: Loss in time from DIAG to MPAIR, MDIST, and MMIN

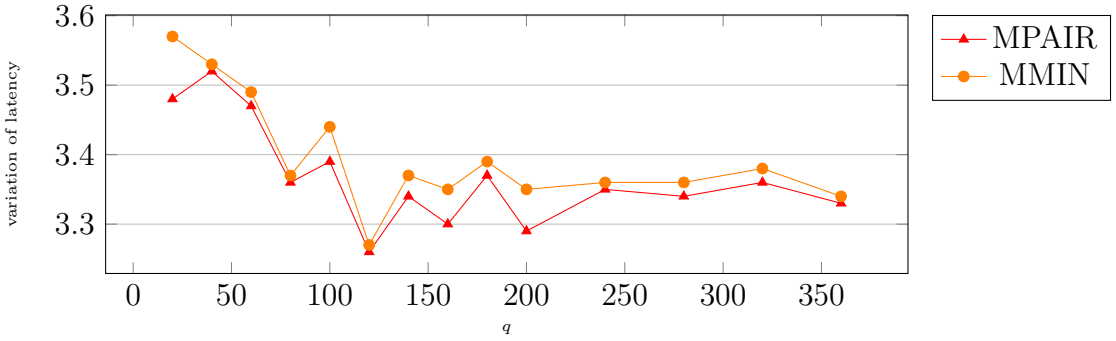


Figure 5.4: Variation of latency of MPAIR and MMIN in 2D mesh

Thus, MPAIR is the best candidate algorithm for pro-traffic applications and MDIAG as the best candidate algorithm for pro-time applications as it generates optimal or optimal plus one time.

Table 5.4 shows that in MDIAG and MPAIR algorithms the average time generated in a 2D torus is almost half of the average time generated in a 2D mesh for all values of q. In MDIAG the average traffic generated in 2D torus is less than the average traffic generated in 2D mesh for all values of q. In MPAIR the average traffic generated in 2D torus is lower than the average value generated in 2D mesh when $q \leq 80$, after which it is slightly more. M in this table represents a mesh and T, a torus.

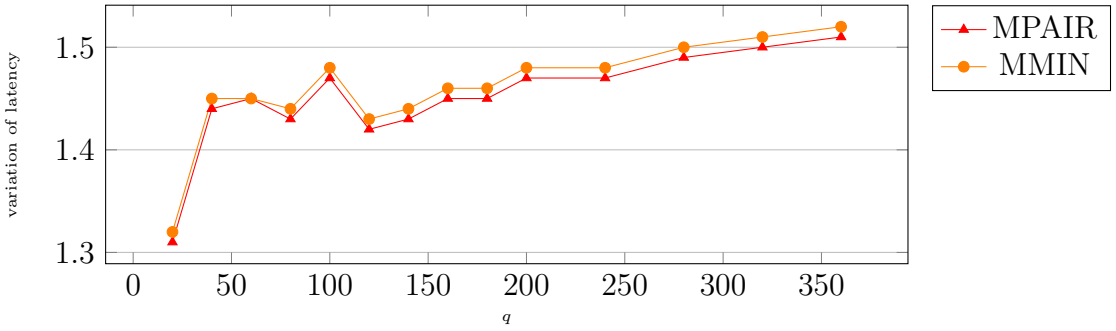


Figure 5.5: Variation of latency of MPAIR and MMIN in 2D torus

Table 5.4: Average time and traffic in 2D mesh and torus networks

q	**MDIAG average time**		**MDIAG average traffic**		**MPAIR average time**		**MPAIR average traffic**	
	M	**T**	**M**	**T**	**M**	**T**	**M**	**T**
20	34	18	128	101	35	18	88	81
40	36	19	200	152	36	19	123	118
60	36	19	245	192	36	19	150	146
80	37	20	281	226	37	20	172	170
100	37	20	301	254	37	20	187	190
120	37	20	322	275	37	20	206	207
140	37	20	336	294	37	20	219	223
160	38	20	344	312	38	20	234	238
180	38	20	355	324	38	20	246	251
200	38	20	363	337	38	20	261	265
240	38	20	374	356	38	20	285	290
280	38	20	383	371	38	20	311	315
320	38	20	389	383	38	20	337	342
360	38	20	395	392	38	20	366	369

5.2 Results on HMDIAG

Figure 5.6 shows that HMDIAG generates the least average time for all values of q. The maximum average time generated by HMDIAG is 40, TASNEM 243, and M-HCM 800. Moreover, HMDIAG time increases with q until $q \leq 160$, after which it is constant. In TASNEM, as the multicast set size increases, the main path length decreases, resulting in a more parallel tree. This is displayed in Figure 5.6, as the time decreases with increase in q. When $q \geq 800$, TASNEM generates on average approximately 61% more time than HMDIAG. In M-HCM as the multicast set size increases, almost all nodes of the 2D torus are in the destination set, resulting in longer paths and higher multicast time.

Figure 5.7 shows that HMDIAG generates the least average latency for all values of q. The maximum average latency generated by HMDIAG is 2 μs, TASNEM 7.7 μs, and M-HCM 20.5 μs. When $q \geq 800$, TASNEM generates on average approximately 31% more latency than HMDIAG.

Figure 5.8 shows that HMDIAG generates the least coefficient variation of multicast time for all values of q, proving that the T-s generated by HMDIAG are the most parallel. The maximum coefficient variation of multicast time generated by HMDIAG is 0.36 ns, TASNEM 0.58 ns, and M-HCM 0.60 ns. On average, the average coefficient variation of multicast time generated by TASNEM is approximately 19% more than that of HMDIAG.

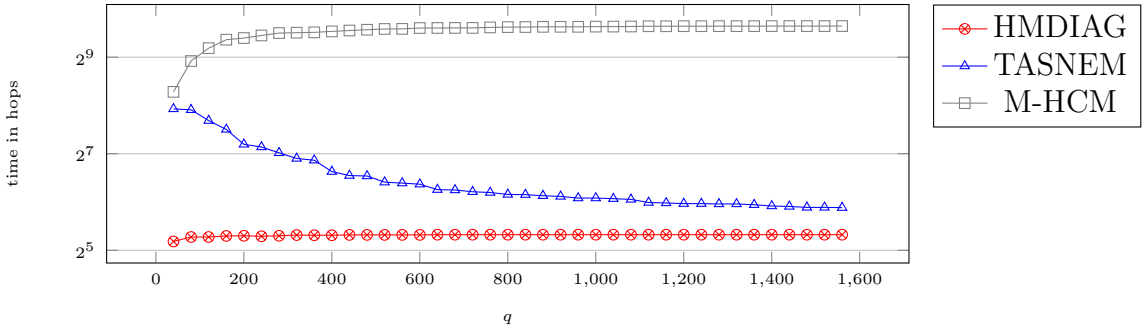


Figure 5.6: Average time

Figure 5.9 shows that HMDIAG generates on average approximately 16% less traffic than TASNEM when $q \leq 160$, after which HMDIAG generates on average approximately 14% more traffic. This is mainly because as the number of destination nodes increases, the probability of reaching a destination node from a path sending the message to another destination node increases and HMDIAG forces nodes to receive a message only from nodes on the PDP or SDP. Thus, although shorter paths might be available, to achieve a (2n - 1)-additive approximation on multicast time, HMDIAG does not consider them as every branching leads to a time unit of delay. M-HCM generates the most traffic.

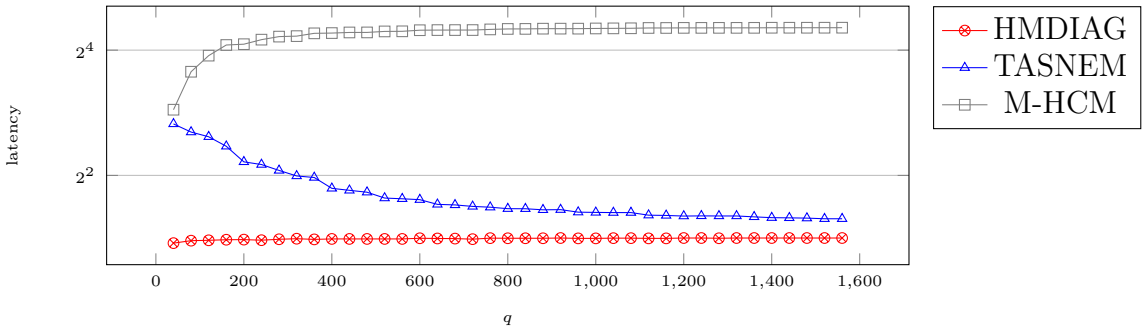


Figure 5.7: Average latency

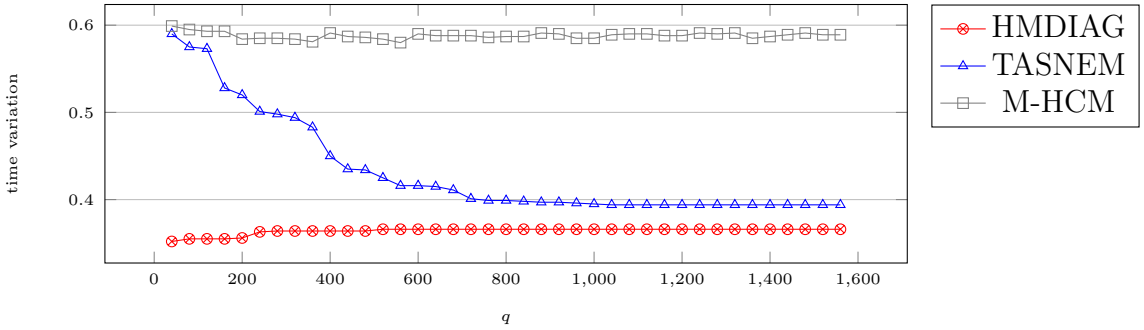


Figure 5.8: Average coefficient variation of multicast time

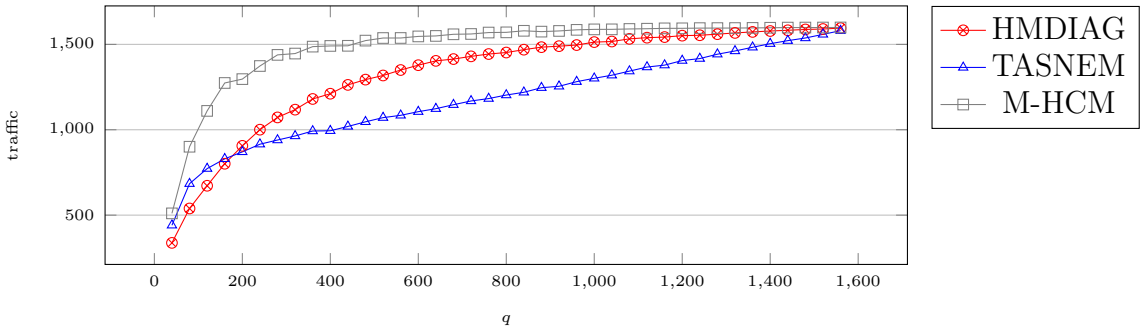


Figure 5.9: Average traffic

Figure 5.10 shows the average multicast time generated by HMDIAG in 2D and 3D torus networks. On average, the time generated in 3D torus is 55% less than that in 2D torus. This is because for the same number of compute nodes, as the dimension of a torus increases, the maximum possible path length to reach a destination node decreases.

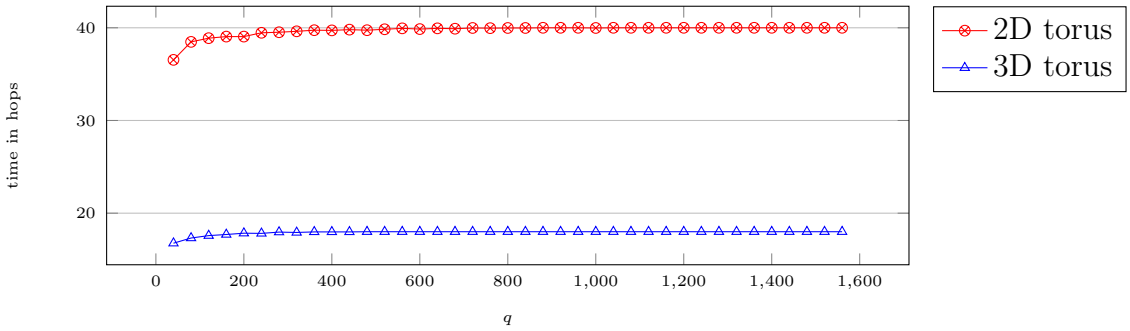


Figure 5.10: Average time in 2D and 3D torus networks

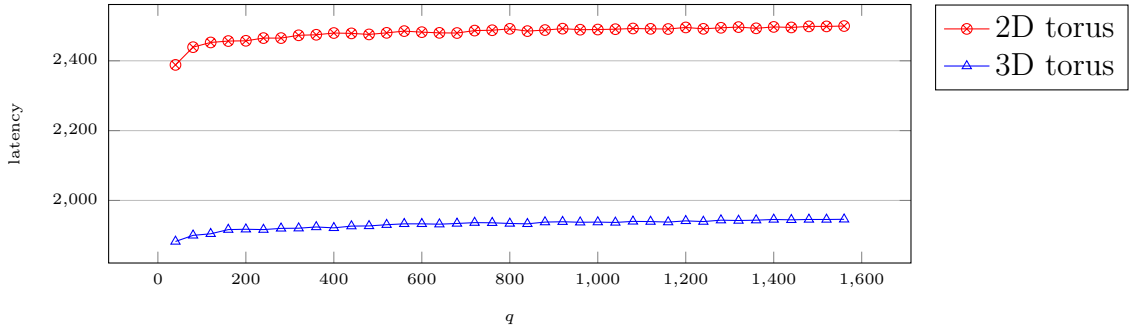


Figure 5.11: Average latency in 2D and 3D torus networks

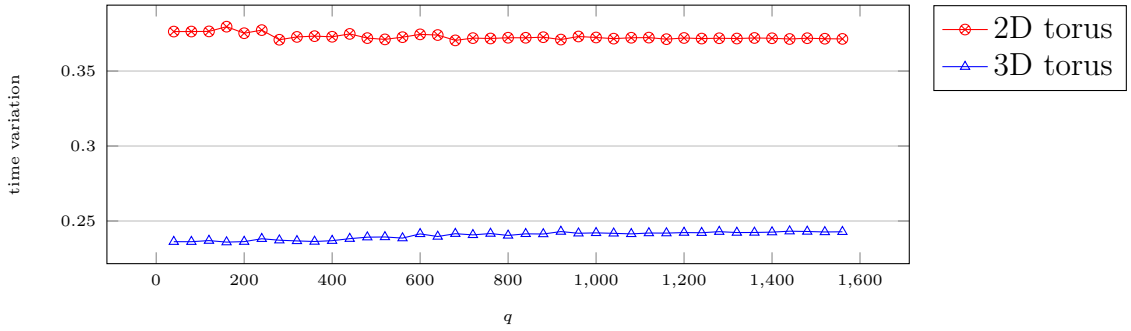


Figure 5.12: Average Coefficient variation of multicast time in 2D and 3D torus networks

Figure 5.11 shows the average latency generated by HMDIAG in 2D and 3D torus. On average, the latency generated in 3D torus is 22% less than that in 2D torus. This is also because for the same number of compute nodes, as the dimensions of a torus increases, the maximum possible path length to reach a destination node decreases.

Figure 5.12 shows the average coefficient variation of multicast time generated by HMDIAG in 2D and 3D torus networks. On average, the average coefficient variation of multicast time generated in 3D torus is 36% less than that in 2D torus. This is because, as the dimensions of a torus increases, the tree becomes more parallel.

Figure 5.13 shows the average traffic generated by HMDIAG in 2D and 3D torus networks. The average traffic generated in 3D torus is on average 12% less than the average traffic generated in 2D torus when $q \leq 480$, after which it is more on average by 22%. This is mainly because as the number of destination nodes and dimensions

increases, the probability of reaching a destination node from a path sending the message to another destination node increases and HMDIAG forces nodes to receive a message only from nodes on the PDP or SDP. Thus, although shorter paths might be available, to achieve a $(2n-1)$-additive approximation on multicast time, HMDIAG does not consider them as every branching leads to a time unit of delay.

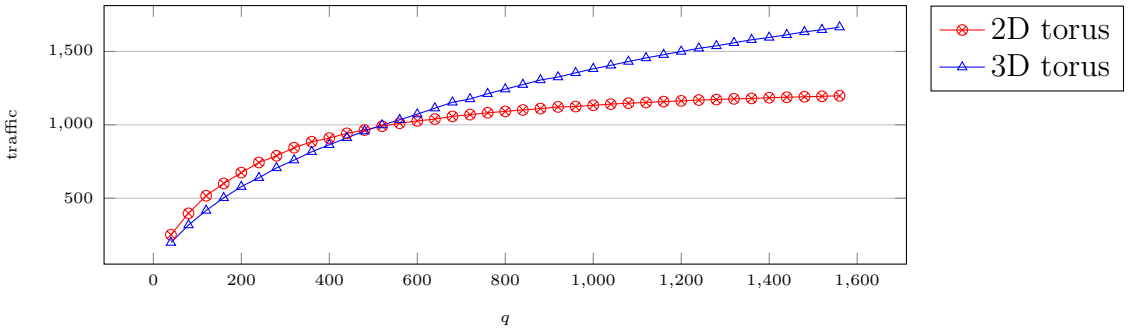


Figure 5.13: Average traffic in 2D and 3D torus networks

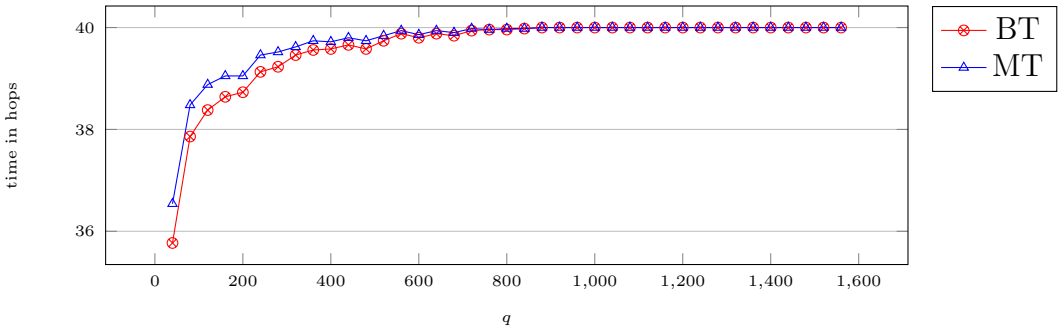


Figure 5.14: Average broadcast and multicast time in 2D torus

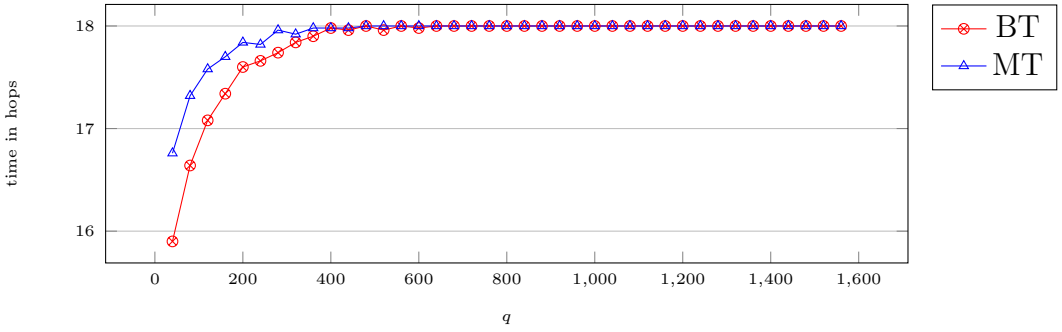


Figure 5.15: Average broadcast and multicast time in 3D torus

Figure 5.14 shows the broadcast time (BT) and multicast time (MT) of the trees generated by HMDIAG in 2D torus. The maximum difference between the BT and MT is two and when $q \geq 840$, BT and MT are equal.

Figure 5.15 shows the broadcast time (BT) and multicast time (MT) of the trees generated by HMDIAG in 3D torus. The maximum difference between the BT and MT is two and when $q \geq 640$, BT and MT are equal. Although theoretically it is possible to have 2n-1 extra time units in an nD Torus to achieve the multicast process, simulations showed that in a 3D torus the maximum extra time is 2. This is mainly because of the position of the primary Diagonal Path (PDP).

Figures 5.16 and 5.17 show the effect of network size on time and traffic generated by the three algorithms, respectively. The multicast set size is set to 20% of the maximum possible destination node set size. Figure 5.16 shows that increase in torus size increases the time generated by the three algorithms. The increase rate in time of HMDIAG is the least. The average increase in time of TASNEM is approximately three times the average increase of HMDIAG. The network size has the lowest effect on the time generated by HMDIAG. Figure 5.17 shows that increase in torus size increases the traffic. The increase rate of HMDIAG and TASNEM is almost the same when $T_{size} \leq 30 \times 30$, after which the traffic increase rate of HMDIAG is more by approximately 11%.

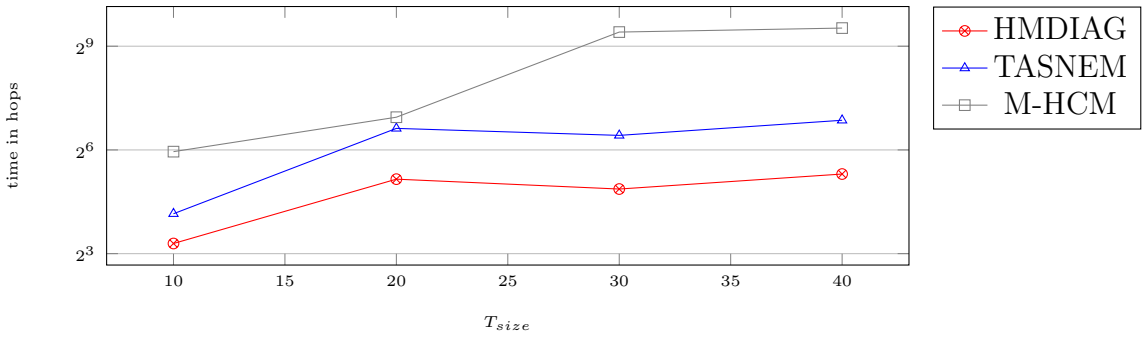


Figure 5.16: Average time as a function of torus size

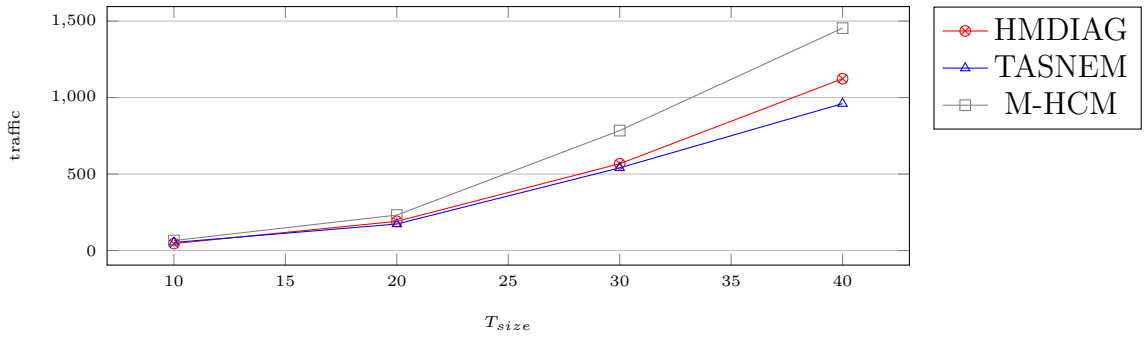


Figure 5.17: Average traffic as a function of torus size

5.3 Results on DCS and DCSBLVT

5.3.1 Failure Percentage

In table 5.5 the failure percentage of the algorithms when $\sigma = p$ is recorded. KMK fails when it does not return a tree. DDVCA, ESC, and ATabu fail when they generate a tree not complying with both constraints. KMK and KMKh have equal failure percentage. In the simulations, DCS and DCSBLVT generate trees complying with both constraints.

As the cardinality of the multicast set increases, all four algorithm failing percentages increases. For instance, when $n = 40$ and $q = 20\%$ of n, ESC creates trees not complying with both constraints 26% of the time. This value increases to 41% when $q = 50\%$ of n and to 42% when $q = 80\%$ of n.

On average KMK fails in 9% of the cases, DDVCA in 47%, ESC in 58%, and ATabu in 28%.

5.3.2 Inter-destination Delay Variation

Figure 5.18 presents the average inter-destination delay variation generated by the algorithms when $\sigma = p$.

Table 5.5: Failure percentage when $\sigma = p$

	KMK			DDVCA			ESC			ATabu		
n	q			q			q			q		
	20%	50%	80%	20%	50%	80%	20%	50%	80%	20%	50%	80%
20	6	12	11	18	29	34	26	41	42	14	21	29
40	4	10	12	21	42	59	29	55	69	14	33	48
60	7	12	13	23	54	69	39	62	76	10	39	41
80	6	8	13	27	50	69	41	66	79	11	30	46
100	4	8	13	34	58	78	47	73	86	8	23	43
120	4	8	11	31	68	82	50	80	90	18	25	43
avg	5.2	9.7	12.2	25.7	50.2	65.2	38.7	62.8	73.7	12.5	28.5	41.7

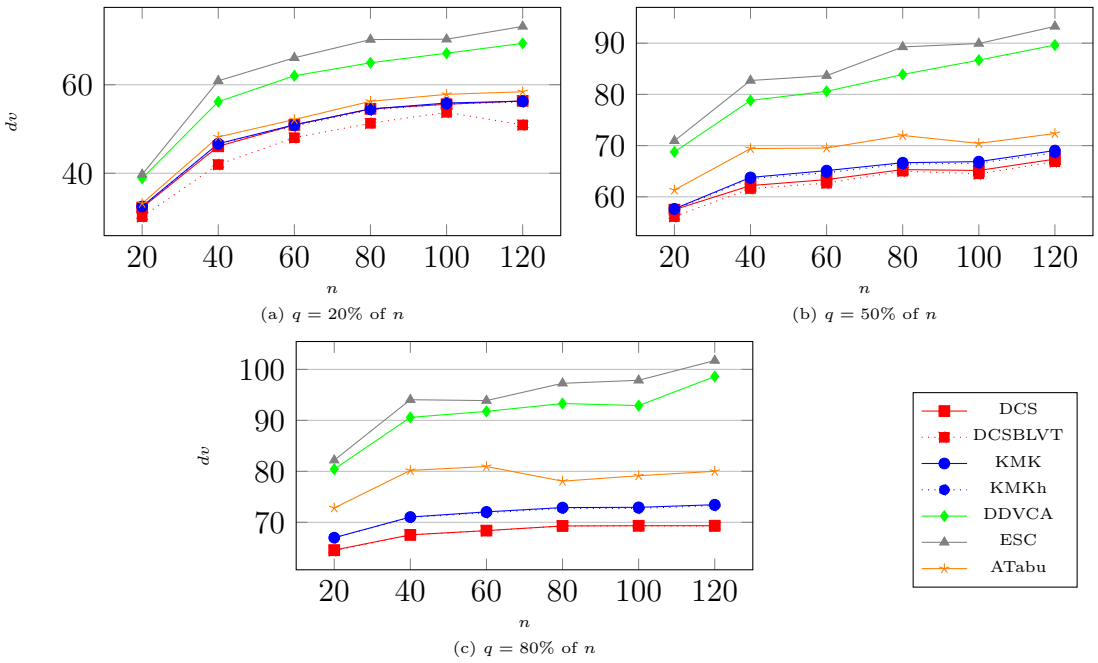


Figure 5.18: Inter-destination delay variation when $\sigma = p$

KMK, KMKh, and DCS have almost the same inter-destination delay variation when $q = 20\%$ of n. DDVCA, and ESC generate the highest inter-destination delay

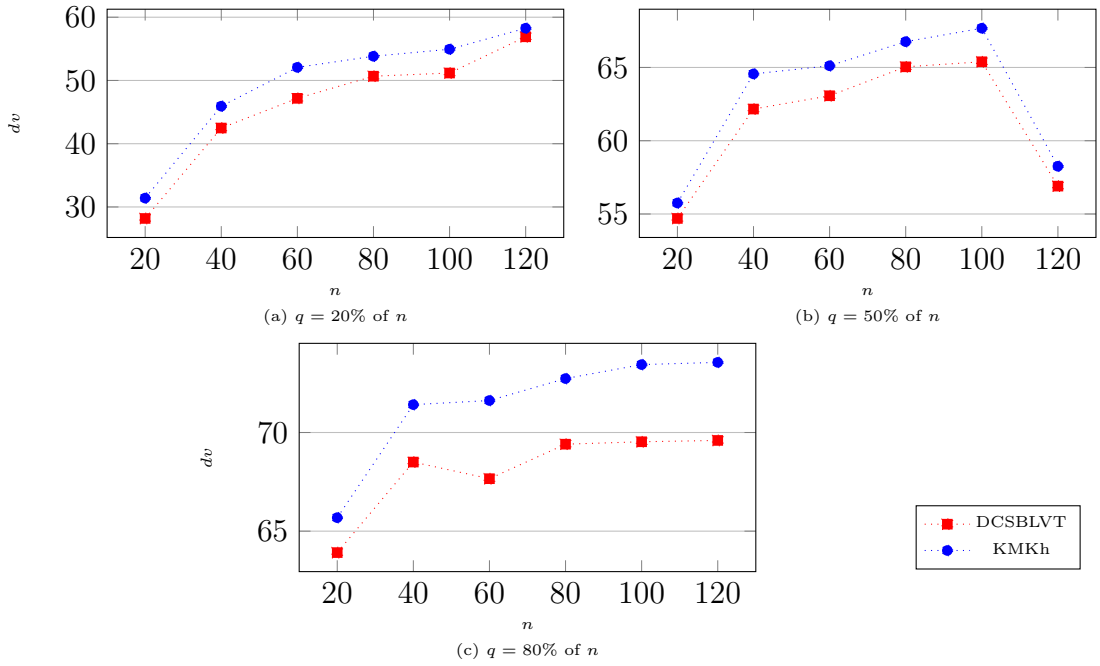


Figure 5.19: Inter-destination delay variation when $\sigma = 0$

variation. KMK has lower inter-destination delay variation than DDVCA, ESC, and ATabu.

DCS has lower inter-destination delay variation than KMK, KMKh, DDVCA, ESC, and ATabu when $q > 20\%$ of n. On average, it is 3.5% less than KMK, 3.2% less than KMKh, and 8.3% less than ATabu.

DCSBLVT has the least inter-destination delay variation for all values of n and q. DCSBLVT has at most 10% less inter-destination delay variation than KMK, 9.8% less than KMKh, and 15.8% less than ATabu.

DCSBLVT decreases the inter-destination delay variation of DCS the most when the multicast group is a small fraction of network nodes. This improvement percentage decreases with increase in the cardinality of the multicast group for both algorithms. As the cardinality of the multicast group increases, it is harder to replace paths to decrease the inter-destination delay variation of the tree. DCSBLVT

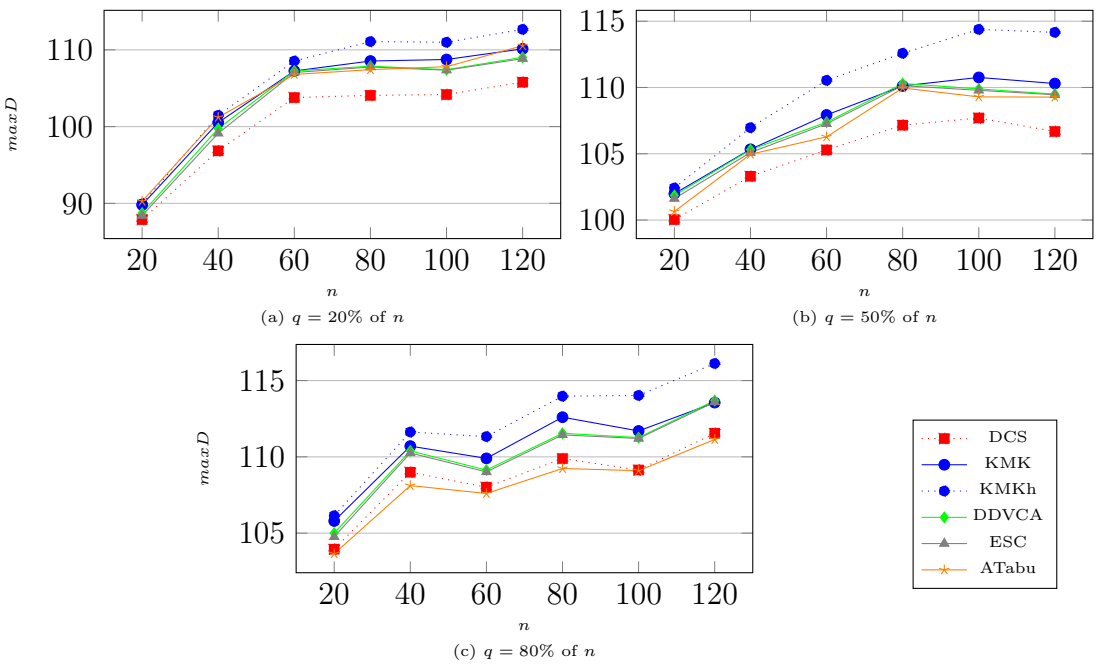


Figure 5.20: Maximum end-to-end delay when $\sigma = p$

decreases the inter-destination delay variation of DCS by at most 9.7%. On contrary, KMKh decreases the inter-destination delay variation of KMK by at most 0.6%. This improvement in DCSBLVT is on average in 38.2% of the cases. On contrary, in KMKh, it is in 14.9% of the cases. Thus, the strategy used in DCSBLVT is more powerful than the one used in KMKh.

Figure 5.19 displays the average inter-destination delay variation of DCSBLVT and KMKh when $\sigma = 0$. DCSBLVT has at most 10.5% less inter-destination delay variation than KMKh.

5.3.3 End-to-End Delay

Figure 5.20 displays the average end-to-end delay generated by the algorithms when $\sigma = p$. DCS and DCSBLVT generate the same end-to-end delay. KMKh generates more end-to-end delay than KMK, since it takes longer paths from the source node

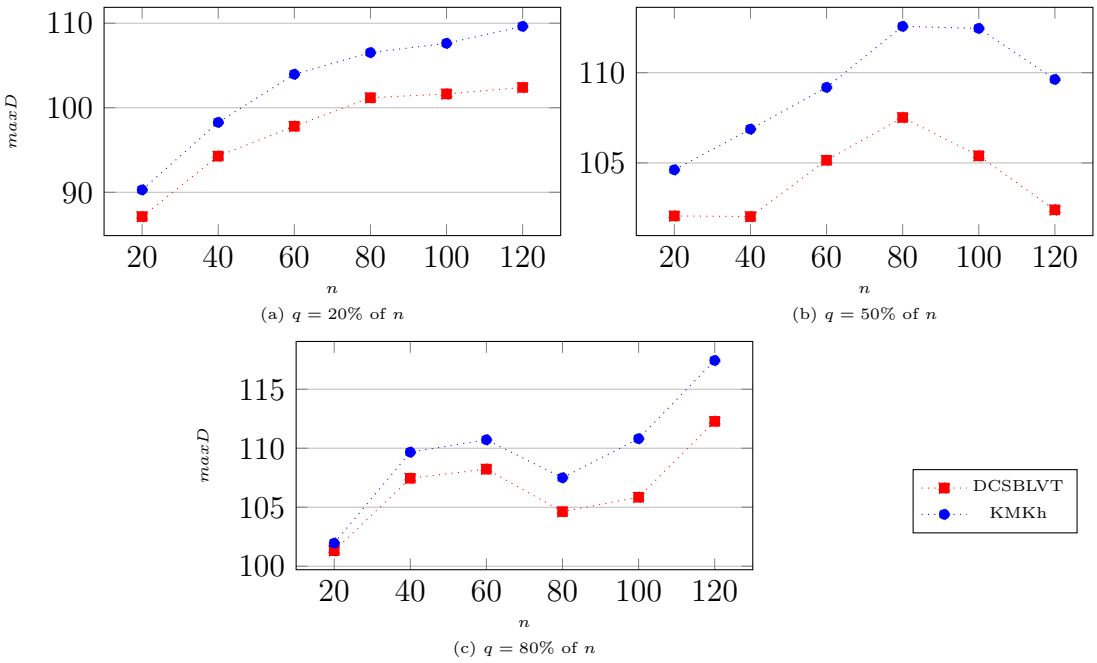


Figure 5.21: Maximum end-to-end delay when $\sigma = 0$

to the core node. When $q < 80\%$ of n the end-to-end delay of DCS and DCSBLVT is the lowest. However, when $q \geq 80\%$ of n, ATabu on average generates 0.4% lower end-to-end delay than DCS and DCSBLVT.

On average, DCS and DCSBLVT have 2.7% less end-to-end delay than KMK, 4.3 % less than KMKh, 2.1% less than DDVCA, 2 % less than ESC, and 1.5% less than ATabu.

Figure 5.21 displays the end-to-end delay of DCSBLVT and KMKh when $\sigma = 0$. The end-to-end delay of DCSBLVT is the least for all values of n and q. DCSBLVT has on average 4.2% less end-to-end delay than KMKh.

5.3.4 Execution Time

Table 5.6 compares the execution time of DCS to that of KMK when $\sigma = p$. When q is 20%, 50%, and 80% of n, DCS has 33.7%, 46.5%, and 58% the execution time of

KMK, respectively. On average, the execution time of DCS is 46% of that of KMK and they have the same time complexity.

Table 5.6 also compares the execution time of DCSBLVT to that of KMKh when $\sigma = 0$. When q is 20%, 50%, and 80% of n, DCSBLVT has 93.9%, 91.5%, and 83.8% the execution time of KMK, respectively. When $n < 80$, the execution time of DCSBLVT is lower than that of KMKh. When $n \geq 90$, there are cases where it is slightly higher.

5.3.5 Nodes Explored

Table 5.7 compares the number of nodes explored by DCS to that KMK. KMK finds the shortest path from the source to all nodes in the network and from every group member to all nodes, with a total of $qn+q$ nodes explored. Given that both algorithms have the same time complexity, DCS explores on average 67.5% of the nodes KMK explores, causing it to have execution time lower than that of KMK (Table 5.6).

Table 5.6: Execution time

	DCS versus KMK $\sigma = p$			**DCSBLVT versus KMKh** $\sigma = 0$		
$\boldsymbol{n}$	$\boldsymbol{q}$			$\boldsymbol{q}$		
	20%	50%	80%	20%	50%	80%
20	28.5	36.5	41.2	66.5	62	51.5
40	30.3	39.8	47.8	93.2	91.8	68.5
60	32.6	43.1	53.5	94.1	92.7	89.4
80	34.2	48.6	61.2	96.5	94.1	91.4
100	37.2	53.5	68.5	101.5	99	93.5
120	39.3	57.6	75.9	111.7	109.6	108.7
avg	33.7	46.5	58	93.9	91.5	83.8

Table 5.7: Nodes explored

n	q		
	20%	50%	80%
20	41.9	62.4	68.1
40	55.3	68.8	71.9
60	62.8	69.8	72.3
80	65.6	71.7	73
100	69.6	72.4	73.9
120	68.6	72.7	74.4
avg	60.6	69.6	72.3

5.3.6 Re-executions in Dynamic DCS and DCSBLVT

In this section, the dynamic approach for DVBMT tree reorganization is studied.

Table 5.8 shows the total number of requests performed, re-executions because of join requests, and their respective percentages for every combination of n and q value for DCS. Re-executions are the highest when $n = 20$ and $q = 20\%$ of n. For a given n value, the number of re-executions decreases as the multicast set size increases. This follows from the observation that as the percentage of group members increases, the values set for Δ and σ reflect a higher percentage of the n nodes. On average 3.4% of total requests triggered re-executions and 38.1% of the graphs suffered from a re-execution before successfully accepting the q join or leave requests.

Table 5.8: Re-executions in dynamic DCS

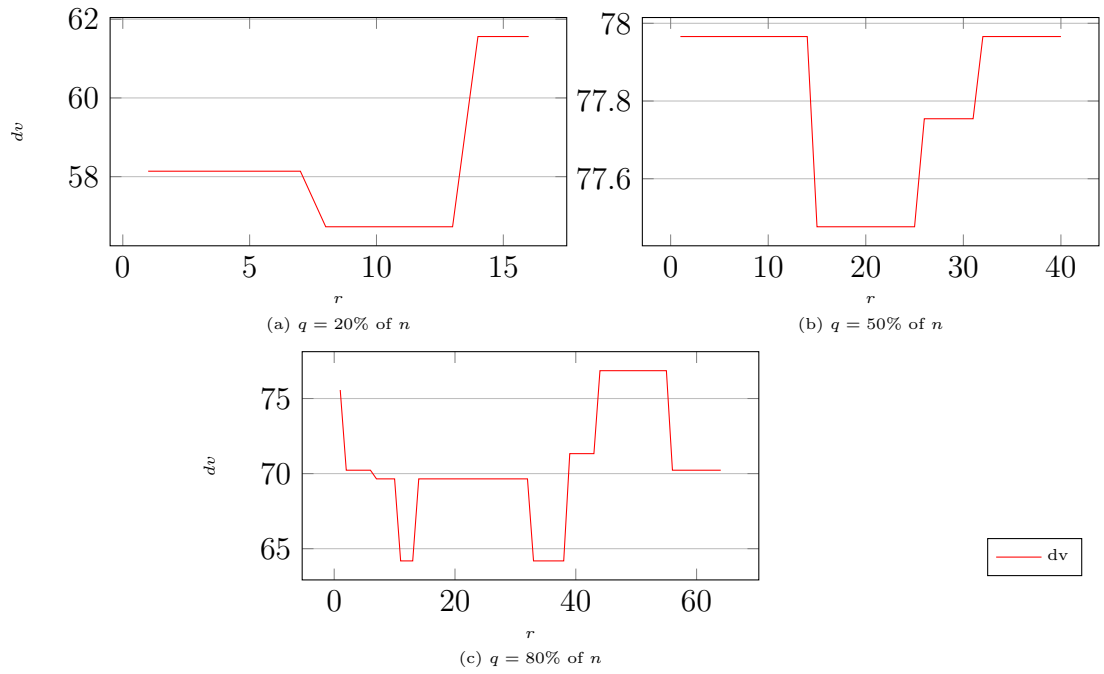
$n-q$	request #	re-execution #	re-execution%
20-20%	304	61	20.1
20-50%	758	41	5.4
20-80%	1454	13	0.9
40-20%	584	54	9.2
40-50%	1616	35	2.2
40-80%	2764	22	0.8
60-20%	854	45	5.3
60-50%	2418	34	1.4
60-80%	4004	22	0.5
80-20%	1084	55	5.1
80-50%	2986	41	1.4
80-80%	5146	25	0.5
100-20%	1452	47	3.2
100-50%	3614	35	1
100-80%	6666	26	0.4
120-20%	1498	55	3.7
120-50%	3996	47	1.2
120-80%	7508	28	0.4

Table 5.9 shows the total number of join/leave requests performed, re-executions as a result of join requests, and their respective percentages for every combination of n and q value for DCSBLVT. Re-executions are the highest when $n = 20$ and $q = 20\%$ of n. On average 2.8% of total requests triggered re-executions and 36.8% of the graphs suffered from a re-execution before successfully accepting the q join or leave requests.

Table 5.9: Re-executions in dynamic DCSBLVT

$n-q$	request #	re-execution #	re-execution %
20-20%	388	46	11.9
20-50%	776	44	5.7
20-80%	1356	29	2.1
40-20%	631	42	6.7
40-50%	1581	42	2.7
40-80%	3092	10	0.3
60-20%	927	39	4.2
60-50%	2416	39	1.6
60-80%	4263	19	0.4
80-20%	1128	55	4.9
80-50%	3074	40	1.3
80-80%	5607	20	0.4
100-20%	1442	52	3.6
100-50%	3512	44	1.3
100-80%	6646	28	0.4
120-20%	1729	48	2.8
120-50%	4344	41	0.9
120-80%	7952	26	0.3

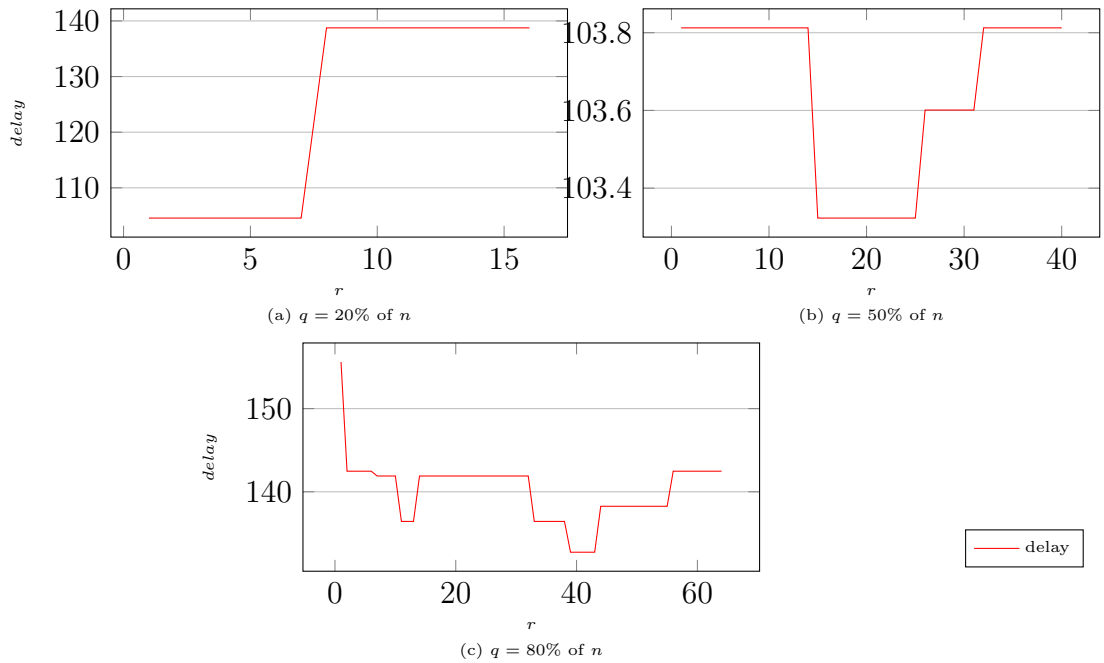
Figure 5.22 and Figure 5.23 show examples of change in inter-destination delay variation and end-to-end delay of the multicast tree as nodes join/leave the multicast group for DCS when $n = 80$, respectively. The values at $r = 0$ represent the tree of the current multicast session. At $r = i$ the values represent the reorganized tree after request number i.



(a) $q = 20\%$ of n (b) $q = 50\%$ of n

(c) $q = 80\%$ of n

Figure 5.22: Change in inter-destination delay variation in dynamic DCS for $n = 80$



(a) $q = 20\%$ of n (b) $q = 50\%$ of n

(c) $q = 80\%$ of n

Figure 5.23: Change in end-to-end delay in dynamic DCS for $n = 80$

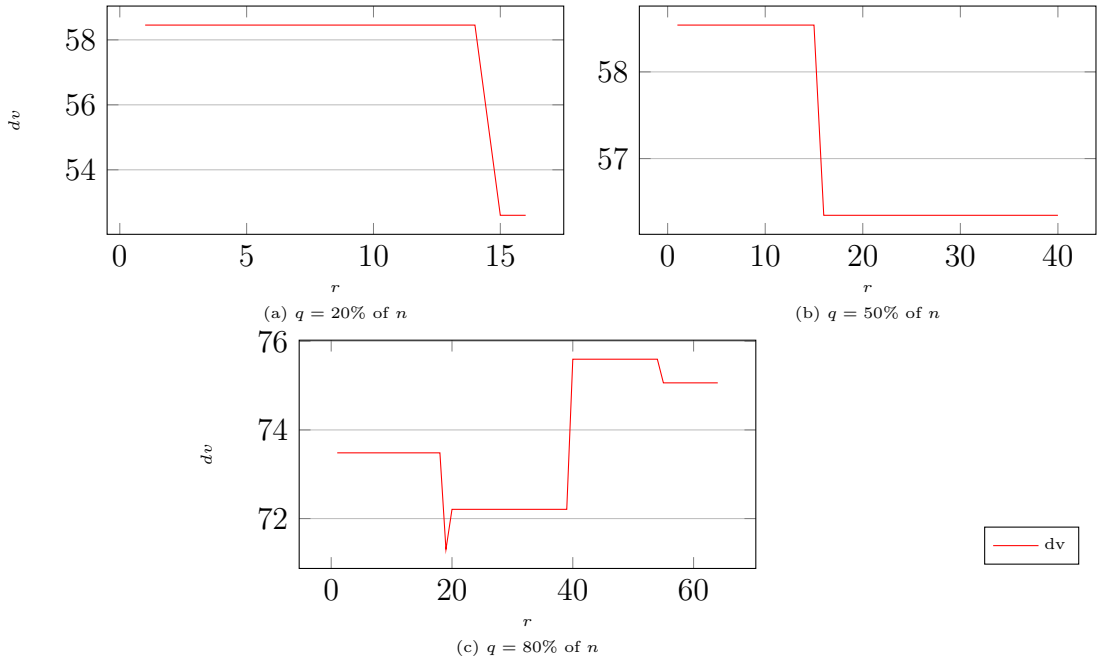


Figure 5.24: Change in inter-destination delay variation in dynamic DCSBLVT for $n = 80$

In Figure 5.22 and Figure 5.23 when $q = 20\%$ of n, 16 join/leave requests are performed with $\sigma = 65.6$ and $\Delta = 156.5$. At join request 8, re-execution occurs and a new core node is selected. This decreases the inter-destination delay variation and increases the end-to-end delay. In Figure 5.22 and Figure 5.23 when $q = 50\%$ of n, 40 join/leave requests are performed. No re-executions occur. However, both the inter-destination delay variation and the end-to-end delay vary as nodes join and leave the multicast group. In Figure 5.22 and Figure 5.23 when $q = 80\%$ of n, 64 join/leave requests are performed with $\sigma = 78.2$ and $\Delta = 197$. Two join re-executions occur at request 2 and 56.

Figure 5.24 and Figure 5.25 show examples of change in inter-destination delay variation and end-to-end delay of the multicast tree as nodes join/leave the multicast group for DCSBLVT when $n = 80$, respectively. The delay and inter-destination delay

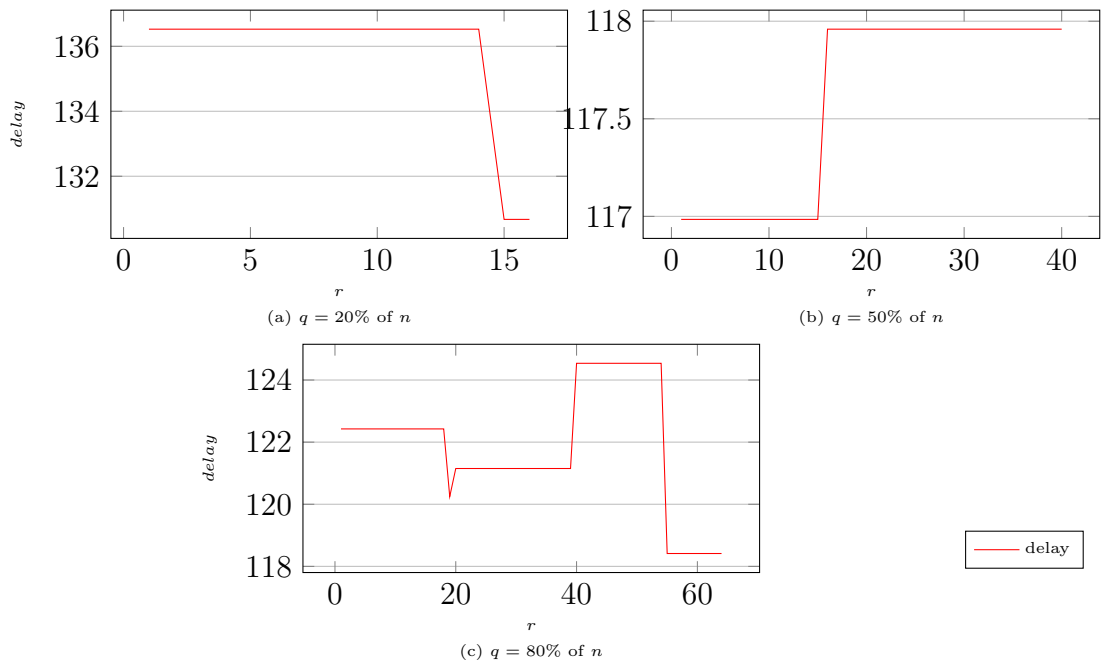


Figure 5.25: Change in end-to-end delay in dynamic DCSBLVT for $n = 80$

variation at point zero of the axis is the delay and inter-destination delay variation of the initial tree, T, before any nodes are added or deleted. In Figure 5.24 and Figure 5.25 when $q = 20\%$ of n, 16 join/leave requests are performed. No re-executions occur. In Figure 5.24 and Figure 5.25 when $q = 50\%$ of n, 40 join/leave requests are performed with $\sigma = 61.9$ and $\Delta = 175.2$. One re-execution occurs at join request 16, a new core is selected, and a new tree constructed. Thus, there is a slight decrease in inter-destination delay variation and a slight increase in end-to-end delay. In Figure 5.22 and Figure 5.23 when $q = 80\%$ of n, 64 join/leave requests are performed with $\sigma = 80.1$ and $\Delta = 158.9$. One re-execution occurs at join request 55.

5.4 Results on MCDVBMT

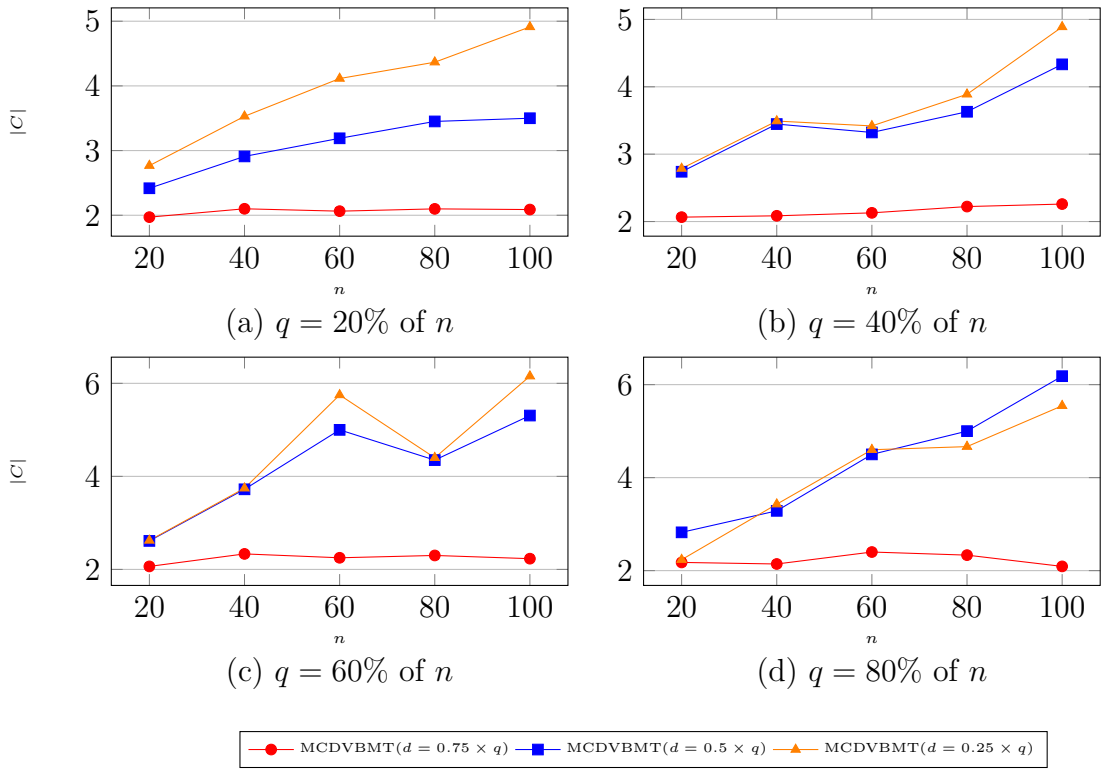


Figure 5.26: Cardinality of core nodes

5.4.1 Cardinality of Core Nodes Generated by MCDVBMT

The cardinality of core nodes selected by MCDVBMT with three d values is studied. d in Algorithm 4.8 is set to $0.75 \times q$, $0.5 \times q$, and $0.25 \times q$ (Figure 5.26).

MCDVBMT($d = 0.75 \times q$) selects the least number of cores for all values of n and q. On average, it selects 2 nodes. MCDVBMT($d = 0.5 \times q$) selects lower number of core nodes than MCDVBMT($d = 0.25 \times q$) when $q <= 60\%$ of n. However, when $q = 80\%$ of n on average MCDVBMT($d = 0.25 \times q$) selects less core nodes than MCDVBMT($d = 0.5 \times q$). This is because as d decreases and the percentage of q

increases, the algorithm uses the candidate core nodes with degree greater than d to cover remaining uncovered nodes and find a semi-matching.

5.4.2 Failure Percentage of Single-core Based Algorithms

Table 5.10 presents the number of times the algorithms failed. KMK and DCS failure reflect inability to generate a tree. DDVCA and ESC failure reflect tree generation with inter-destination delay variation greater than σ. Single core failure reflects the inability of single-core based algorithms to generate a tree satisfying both constraints.

On average KMK fails 20%, DCS 2%, DDVCA 47 %, and ESC 61% of the time. All single-core algorithms fail 2% of the time. In the cases where it is not possible to generate a single-core tree satisfying both constraints, MCDVBMT successfully generates a solution.

5.4.3 Inter-destination Delay Variation of MCDVBMT

Figure 5.27 displays the average inter-destination delay variation generated by the algorithms.

The variation of DCS is on average 3.6% less than KMK. DDVCA and ESC have the highest variation.

Table 5.10: Failure Cases

n	KMK	DCS	DDVCA	ESC	Single Core
20	43	4	61	85	4
40	39	3	85	114	3
60	43	5	103	129	5
80	38	5	109	138	5
100	37	4	113	144	4

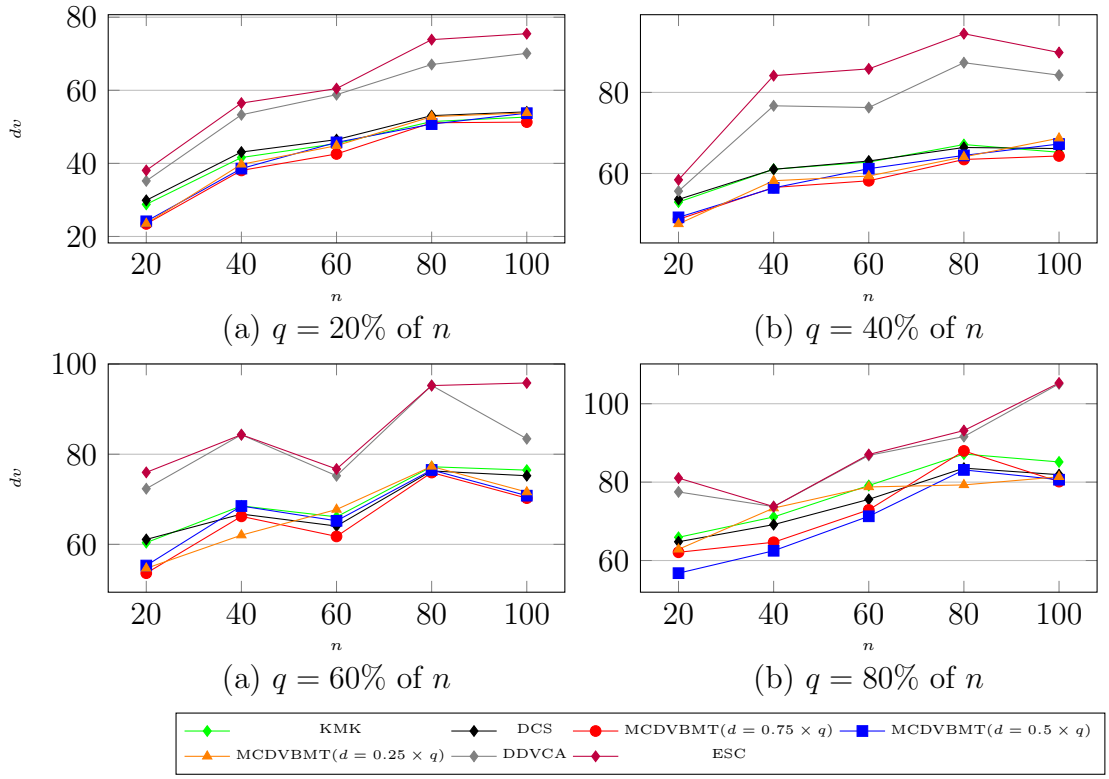


Figure 5.27: Inter-destination delay variation

When comparing the average delay variation generated by MCDVBMT with the three d values, MCDVBMT($d = 0.75 \times q$) generates the least value. Next lowest is MCDVBMT($d = 0.5 \times q$).

When $q = 20\%$ of n, MCDVBMT($d = 0.75 \times q$), MCDVBMT($d = 0.5 \times q$), and MCDVBMT($d = 0.25 \times q$) generate 10.1%, 7.3%, and 6.8% less delay variation than DCS, respectively. When $q = 40\%$ of n, MCDVBMT($d = 0.75 \times q$), MCDVBMT($d = 0.5 \times q$), and MCDVBMT($d = 0.25 \times q$) generate 6.3%, 5.4%, and 5% less delay variation than DCS, respectively. When $q = 60\%$ of n, MCDVBMT($d = 0.75 \times q$), MCDVBMT($d = 0.5 \times q$), and MCDVBMT($d = 0.25 \times q$) generate 6.6%, 5.9%, and 4.8% less delay variation than DCS, respectively. When $q = 80\%$ of n, MCDVBMT($d = 0.75 \times q$), MCDVBMT($d = 0.5 \times q$), and MCDVBMT($d = 0.25 \times q$)

generate 6%, 5.1%, and 5% less delay variation than DCS. On average, the delay variation of MCDVBMT is 6.2% less than DCS.

5.4.4 End-to-end Delay of MCDVBMT

Figure 5.28 displays the average maximum end-to-end delay generated by the algorithms.

On average, DCS has the lowest end-to-end delay. DDVCA, ESC, and KMK have almost the same end-to-end delay. When comparing the end-to-end delay generated by MCDVBMT with the three d values, on average MCDVBMT($d = 0.75 \times q$) generates the least value. Next lowest is MCDVBMT($d = 0.25 \times q$).

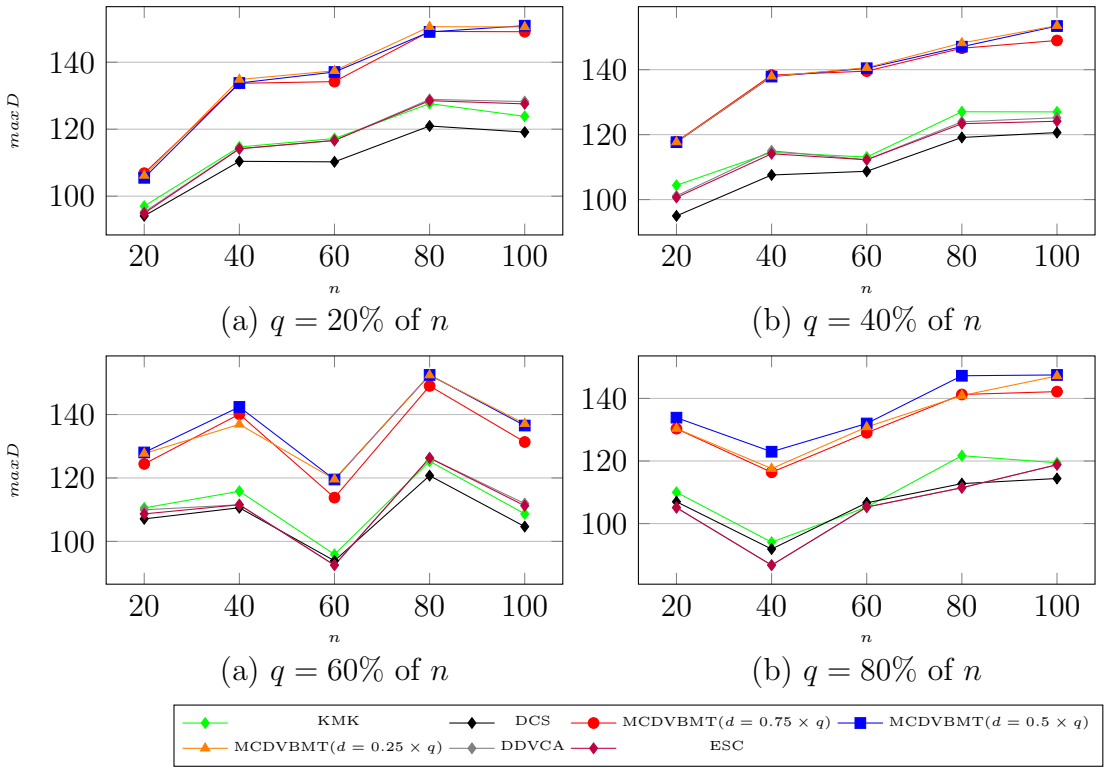


Figure 5.28: Maximum end-to-end delay

On average, the end-to-end delay of MCDVBMT is 25% more than that of DCS.

This is mainly because the algorithm is not trying to minimize the end-to-end delay. The objective is to select a reasonable number of core nodes satisfying both constraints.

5.4.5 Bandwidth Cost

The bandwidth cost, which is the sum of the delays of the path links to multicast group members, is evaluated. It represents the bandwidth consumed by one packet transmission (Figure 5.29).

DCS generates the least cost. Next least cost is KMK. DDVCA and ESC have almost the same cost.

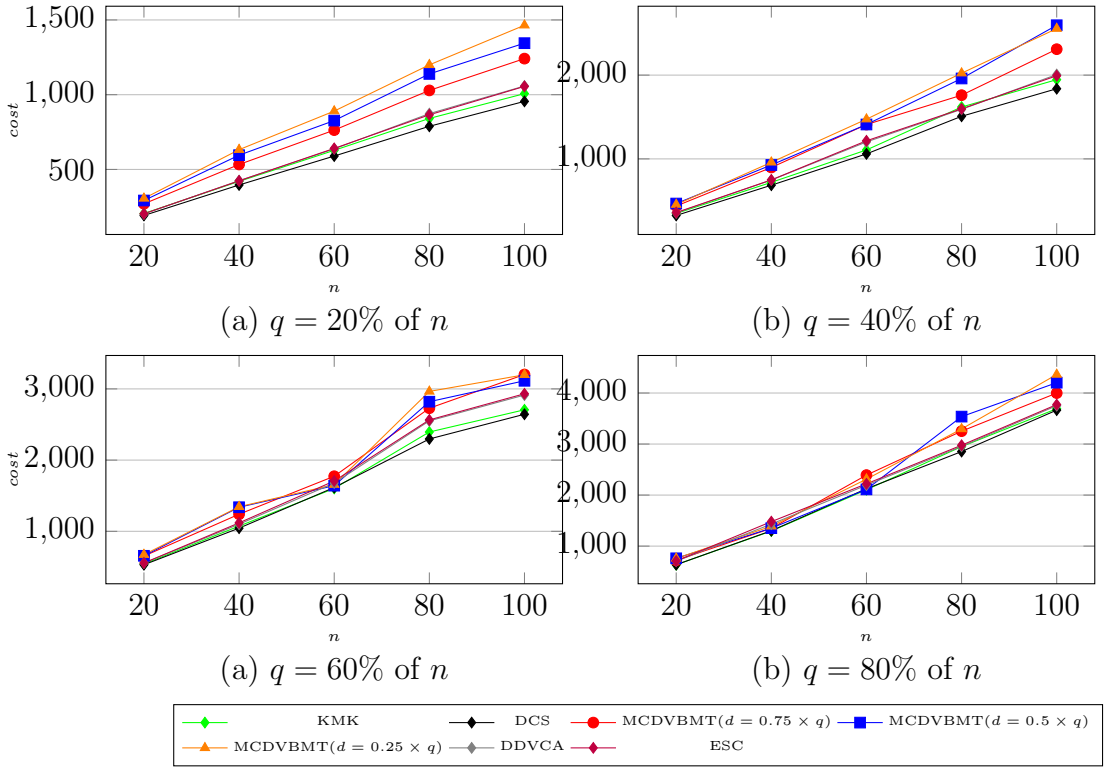


Figure 5.29: Bandwidth cost

When comparing the average cost generated by MCDVBMT with the three d

values, on average MCDVBMT($d = 0.75 \times q$) generates the least value. Next lowest is MCDVBMT($d = 0.5 \times q$). The cost of MCDVBMT is on average 20% more than that of DCS.

5.4.6 Traffic Concentration

To measure the traffic concentration, the number of times a node replicates a message is counted. Traffic concentration on nodes occurs when some nodes have very high degree and other nodes low.

Figure 5.30 displays the traffic concentration generated by the algorithms.

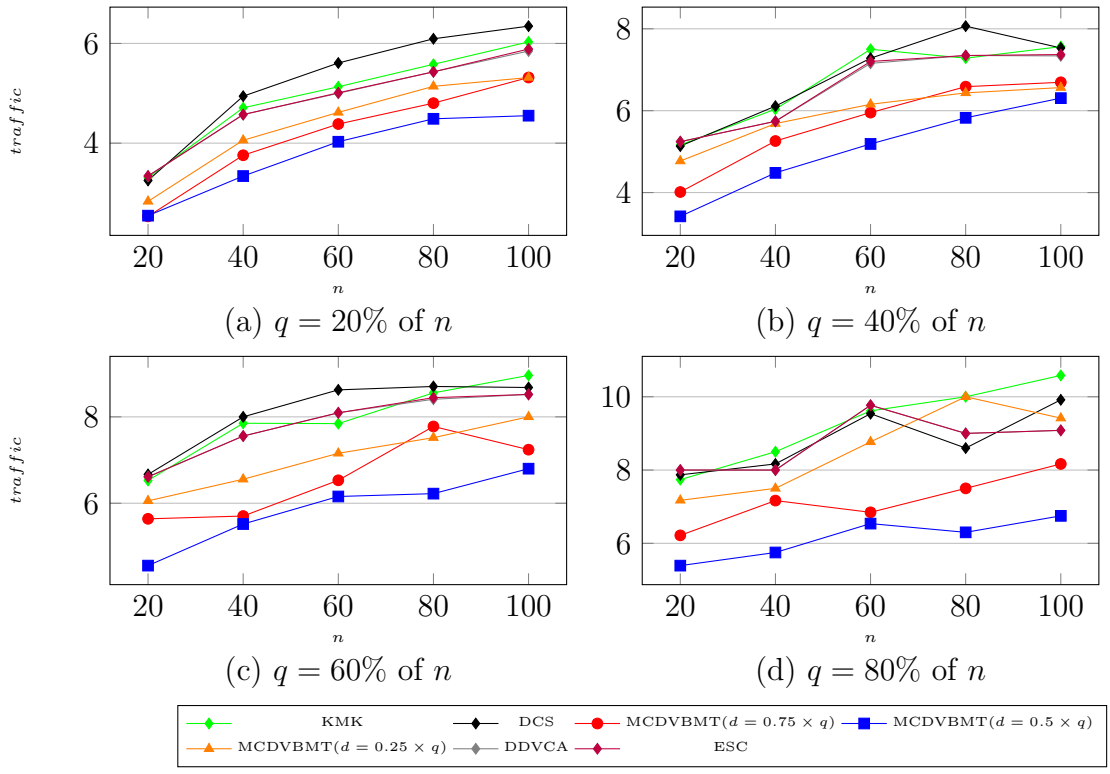


Figure 5.30: Traffic concentration

MCDVBMT($d = 0.5 \times q$) on average generates the least traffic concentration. Next on the list in increasing order are MCDVBMT($d = 0.75 \times q$), MCDVBMT($d = 0.25 \times q$), DDVCA, ESC, KMK, and DCS respectively.

5.4.7 Re-executions in Dynamic MCDVBMT

Table 5.11 presents the average number of requests received by the 200 graphs, percentage of re-executions, and percentage of trees suffering from a re-execution before receiving all q requests when $d = \{0.75 \times q, 0.5 \times q, 0.25 \times q\}$.

An increase in the number of core nodes, leads to a decrease in the average requests handled and an increase in the percentage of leave re-executions and percentage of trees re-executed before receiving all q requests. This is mainly because the trees become more complicated and augmenting paths becomes more difficult as the number of core nodes increases.

On average, 5.2% of the requests triggered re-executions. This value reflects the total percentage of re-executions compared to the total number of dynamic requests.

On average, 53.6% of the sessions suffered from a re-execution before receiving all q requests. This value reflects the percentage of the randomly generated graphs that were not able to withstand all q dynamic requests without a re-execution.

Table 5.11: Dynamic MCDVBMT re-executions

MCDVBMT	**requests #**	**re-execution%**	**re-executed trees%**
$d = 0.75 \times q$	2643	4.8	46.3
$d = 0.5 \times q$	2347	5.3	53.6
$d = 0.25 \times q$	2058	5.7	61

Chapter 6

Conclusion and Future Work

In the first part of the dissertation, time efficient multicast algorithms in torus networks are examined. Since the goal of this research is to develop efficient tree based multicast algorithms for time critical applications, the multicast time is first minimized and the traffic subsequently reduced to find near optimal solutions. Improvements on four tree based multicast algorithms are made. MDIAG algorithm has less time complexity $O(qn)$ compared to DIAG with time complexity $O(qn^2)$ and generates optimal or optimal plus one time in a 2D mesh. However, it increases the time on average by 35%. MPAIR and PAIR have $O(qn)$ time complexity and MPAIR generates on average 28% less traffic. However, it generates an extra time unit in 7% of the cases. MMIN has less time complexity $O(qn)$ compared to MIN with time complexity $O(qn^3)$ and generates on average 5% less time. However, it generates on average 4% more traffic. MDIST has less time complexity $O(qn)$ compared to DIST with time complexity $O(qn^3)$ and generates on average 6.5% less time. However, it increases the traffic on average by 15%. When comparing the three pro-traffic algorithms with DIAG, although MMIN generates the least traffic, it increases the time the most. However, MPAIR generates traffic comparable to MMIN's traffic, has better % increase in time, and lower average coefficient variation of multicast latency

than MMIN. Thus, we conclude that MDIAG is good for pro-time applications and MPAIR for pro-time and pro-traffic applications. The average time generated by the modified algorithms in a 2D torus is almost half the average time generated in a 2D mesh for the same algorithm. In addition, the average traffic generated by the modified algorithms in a 2D torus is less than the average traffic generated in a 2D mesh for the same algorithm in most cases or slightly more.

To tackle the disadvantages of centralized routing, the hybrid version of MDIAG, HMDIAG is designed. HMDIAG performs preprocessing at the source node. At the source node and every intermediate node, another process is performed to retransmit the message to another subset of destination nodes. The proof that HMDIAG is a 3-additive approximation for multicast time in 2D torus networks is given. HMDIAG is extended to be applicable in nD torus networks and the proof that it is a $(2n-1)$-additive approximation algorithm for multicast time is given. Simulation results show that HMDIAG generates less multicast time, latency and coefficient variation of multicast time than TASNEM and M-HCM. However, when the number of destination nodes is greater than 160, HMDIAG generates on average 14% more traffic than TASNEM. For the same cardinality of destination nodes, HMDIAG in 3D torus generates less multicast time and latency than in 2D torus. However, when $q \geq 500$, it generates 22% more traffic in 3D torus. Although HMDIAG is a $(2n-1)$-additive approximation algorithm for multicast time in nD torus networks, in our results the maximum difference between the broadcast and multicast time of the generated trees is two. This is mainly because of the construction of the Diagonal Paths and the relative location of nodes with maximum distance to the diagonal paths.

In the second part of this dissertation, multicast communication constrained by end-to-end delay and inter-destination delay variation, known as Delay and Delay

Variation Bounded Multicast (DVBM) is examined. DCS algorithm that uses a directional approach to select a core node and generate a DVBMT is designed. Another algorithm, based on k-shortest-paths, is designed to further decrease the inter-destination delay variation of the trees generated by DCS. In DCS and DCSBLVT, the source sends a unicast message to the core node, and the core sends the message to group members using the multicast tree. Thus, the end-to-end delay and inter-destination delay variation values set during the execution of the algorithms reflect the values of the generated tree. DCS and DCSBLVT surpass existing algorithms in efficiency since they always generate trees in the performed simulations. KMK, DDVCA, ESC, and ATabu fail in 9%, 47%, 58%, and 28% of the cases, respectively. DCS has lower inter-destination delay variation than KMK, KMKh, DDVCA, ESC, and ATabu. On average DCS has 3.5% less inter-destination delay variation than KMK, 3.2% less than KMKh, and 8.3% less than ATabu. It also has lower end-to-end delay. Although DCS and KMK have the same time complexity, DCS has 46% the execution time of KMK and explores 67.5% of the nodes KMK explores. DCSBLVT generates better results than KMKh in end-to-end-delay and inter-destination delay variation. The strategy DCSBLVT uses to lower the inter-destination delay variation is much more powerful than the one used in KMKh. DCSBLVT decreases the inter-destination delay variation of DCS by at most 9.7% in 38.2% of the cases. On contrary, KMKh decreases the inter-destination delay variation of KMK by at most 0.6% in 14.9% of the cases. DCSBLVT has a factor of $\log n$ more time complexity than KMKh, but they have comparable execution time. The dynamic version of both algorithms that responds to dynamic join and leave requests to the ongoing multicast session by reorganizing the tree and avoiding session disruption is given. On average, only 3.4% of the total requests in DCS triggered re-executions and 2.8% in DCSBLVT.

Next, a multi-core multicast approach for DVBMT is designed. When existing single-core based algorithms fail to generate a tree satisfying delay and delay variation constraints, MCDVBMT successfully selects multiple cores and generates trees rooted at the selected cores satisfying both constraints. MCDVBMT generates less inter-destination delay variation and load on nodes than existing single-core based algorithms. In addition, in MCDVBMT only group members receiving the message from the failing core node suffer from recovery delay when a core node fails. However, the end-to-end delay and cost of MCDVBMT is higher than single-core trees. Setting the maximum covering degree to 50% of q generates the most favorable results. The performance of MCDVBMT in response to dynamic requests is also average when the maximum covering degree is set to 50% of q. On average, only 5.2% of the requests trigger re-executions and 53.6% of trees generated by MCDVBMT suffer from re-execution before receiving all q requests. The results suggest that, if the number of join/leave requests is not very large, the dynamic approach performs well. However, as the change in the multicast group members increases, the tree stops reflecting the initial group and it may be necessary to periodically run the algorithms to keep the inter-destination delay variation low.

This thesis presented contributions in the area of multicast in torus networks and Delay and Delay Variation Bound Multicast. Yet, there exists several interesting future research directions that can be explored:

- Extend the suggested single core algorithms to generate DVBM trees when there exists more than one source node (many-many communication).
- Extend the suggested single core algorithms to generate DVBM trees when all multicast group members are senders and receivers. This variant of the problem is used in many applications including videoconferencing and teleconferencing.

- Develop multi-core algorithms for DVBM using different viable techniques that further lower the inter-destination delay variation of the tree. On such technique could be clustering of multicast group members into groups and then selecting core nodes for every group. Once the multicast group members clustered into groups, DCS could be applied on every subset.

- Develop multi-core algorithms for DVBM that also tries to optimize on delay and bandwidth.

Bibliography

[1] M. Abdel-Baky. New routing techniques for high message passing systems performance, 2000.

[2] S. Ahn, M. Kim, and H. Choo. Efficient algorithm for reducing delay variation on delay-bounded multicast trees in heterogeneous networks. In *Wireless Communications and Networking Conference*, pages 2741–2746. IEEE, 2008.

[3] A. Al-Dubai, M. Ould-Khaoua, and I. Romdhani. On high performance multicast algorithms for interconnection networks. *High Performance Computing and Communications*, pages 330–339, 2006.

[4] Y. Al-Dubai, M. Ould-Khaoua, and L. M. Mackenzie. An efficient path-based multicast algorithm for mesh networks. In *International Parallel and Distributed Processing Symposium*, pages 8–pp. IEEE, 2003.

[5] R. Alverson, D. Roweth, and L. Kaplan. The gemini system interconnect. In *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pages 83–87. IEEE, 2010.

[6] Y. Baddi and M. D. E.-C. El Kettani. Vns-rp algorithm for rp selection in multicast routing protocol pim-sm. In *International Conference on Multimedia Computing and Systems*, pages 595–600. IEEE, 2012.

[7] Y. Baddi and M. D. E.-C. El-Kettani. Parallel grasp algorithm with delay and delay variation for core selection in shared tree based multicast routing protocols. In *Third International Conference on Innovative Computing Technology*, pages 227–232. IEEE, 2013.

[8] S. M. Banik, S. Radhakrishnan, and C. N. Sekharan. Multicast routing with delay and delay variation constraints for collaborative applications on overlay networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):421–431, 2007.

[9] L. Blazević and J.-Y. Le Boudec. Distributed core multicast (dcm): a multicast routing protocol for many groups with few receivers. *ACM SIGCOMM Computer Communication Review*, 29(5):6–21, 1999.

[10] R. V. Boppana, S. Chalasani, and C. Raghavendra. Resource deadlocks and performance of wormhole multicast routing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):535–549, 1998.

[11] K. L. Calvert, E. W. Zegura, and M. J. Donahoo. Core selection methods for multicast routing. In *Fourth International Conference on Computer Communications and Networks*, pages 638–642. IEEE, 1995.

[12] S. Chen and K. Nahrstedt. An overview of quality of service routing for next-generation high-speed networks: problems and solutions. *IEEE network*, 12(6):64–79, 1998.

[13] H. Cheng, J. Cao, and X. Wang. A fast and efficient multicast algorithm for qos group communications in heterogeneous network. *Computer communications*, 30(10):2225–2235, 2007.

[14] S. M. Chung and C.-H. Youn. Core selection algorithm for multicast routing under multiple qos constraints. *Electronics Letters*, 36(4):378–379, 2000.

[15] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE transactions on Parallel and Distributed Systems*, 4(4):466–475, 1993.

[16] M. Darwish, A. Radwan, M. A. El-Baky, and K. Hamed. Gttpm-an efficient deadlock-free multicast wormhold algorithm for communication in 2d torus multicomputers. In *IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 707–712, 2005.

[17] M. Darwish, A. Radwan, M. A. El-Baky, and K. Hamed. Ttpm–an efficient deadlock-free algorithm for multicast communication in 2d torus networks. *Journal of Systems Architecture*, 54(10):919–928, 2008.

[18] M. Darwish, A. Radwan, M. A. El-Baky, and K. Hamed. Ready groups: A path-based multicast algorithm for 2d torus networks. In *7th International Conference on Informatics and Systems*, pages 1–9. IEEE, 2010.

[19] M. A. El-Baky. A tree-based algorithm for multicasting in 2d torus networks. *Egyptian Informatics Journal*, 16(1):45–53, 2015.

[20] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A dynamic bootstrap mechanism for rendezvous-based multicast routing. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1090–1098. IEEE, 1999.

[21] E. Fleury and P. Fraigniaud. Strategies for path-based multicasting in wormhole-routed meshes. *Journal of parallel and distributed computing*, 53(1):26–62, 1998.

[22] F. Font and D. Mlynek. Applying clustering algorithms as core selection methods for multiple core trees. In *International Conference on Communications*, volume 4, pages 2030–2035. IEEE, 2004.

[23] M. R. Garey and D. S. Johnson. A guide to the theory of np-completeness. *WH Freemann, New York*, 70, 1979.

[24] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *ACM SIGARCH Computer Architecture News*, 20(2):278–287, 1992.

[25] H. A. Harutyunyan and X. Dong. A new algorithm for rp selection in pim-sm multicast routing. In *IASTED International Conference on Wireless and Optical Communications*, pages 208–216, 2003.

[26] H. A. Harutyunyan and A. Malani. Efficient multicast algorithms for mesh and torus networks. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 231–236. IEEE, 2014.

[27] H. A. Harutyunyan and M. Terzian. A multi-core multicast tree approach for delay and delay variation multicast routing. In *Eighteenth International Conference on High Performance Computing and Communications*, page to be published. IEEE, Dec 2017.

[28] H. A. Harutyunyan and S. Wang. Efficient multicast algorithms for mesh-connected multicomputers. In *Tenth International Conference on Information Visualization*, pages 504–510. IEEE, 2006.

[29] N. J. Harvey, R. E. Ladner, L. Lovász, and T. Tamir. Semi-matchings for bipartite graphs and load balancing. In *Algorithms and Data Structures Symposium*, pages 294–308. Springer, 2003.

[30] M. R. Kabat, M. K. Patel, and C. R. Tripathy. A heuristic algorithm for core selection in multicast routing. *Journal of Computer Science and Technology*, 26(6):954–961, 2011.

[31] M. R. Kabat and S. P. Sahoo. Adaptive tabu search algorithm for rp selection in protocol independent multicast-sparse mode. In *World Congress on Information and Communication Technologies*, pages 634–638. IEEE, 2011.

[32] S. Kapoor and S. Raghavan. Improved multicast routing with delay and delay variation constraints. In *Global Telecommunications Conference*, volume 1, pages 476–480. IEEE, 2000.

[33] A. Karaman and H. Hassanein. Dcmc-delay-constrained multipoint communication with multiple sources. In *Eighth International Symposium on Computers and Communication*, pages 467–472. IEEE, 2003.

[34] A. Karaman and H. Hassanein. Core-selection algorithms in multicast routing-comparative and complexity analysis. *Computer Communications*, 29(8):998–1014, 2006.

[35] M. Kim, M. W. Mutka, and H.-Y. Kim. Esc: Estimation of selecting core for reducing multicast delay variation under delay constraints. *International Journal of Communication Systems*, 24(1):40–52, 2011.

[36] Z. Kun, W. Heng, and L. Feng-Yu. Distributed multicast routing for delay and delay variation-bounded steiner tree using simulated annealing. *Computer Communications*, 28(11):1356–1370, 2005.

[37] Z. Kun, Q. Yong, and Z. Hong. Dynamic multicast routing algorithm for delay and delay variation-bounded steiner tree problem. *Knowledge-Based Systems*, 19(7):554–564, 2006.

[38] Y. Lan, L. Li, and A. Esfahanian. Distributed multi-destination routing in hypercube multiprocessors. In *Third conference on Hypercube concurrent computers and applications: Architecture, software, computer systems, and general issues*, pages 631–639. ACM, 1988.

[39] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.

[40] X. Lin and L. M. Ni. Deadlock-free multicast wormhole routing in multicomputer networks. In *ACM SIGARCH Computer Architecture News*, volume 19, pages 116–125. ACM, 1991.

[41] X. Lin and L. M. Ni. Multicast communication in multicomputer networks. *IEEE transactions on Parallel and Distributed Systems*, 4(10):1105–1117, 1993.

[42] Y.-D. Lin, N.-B. Hsu, and R.-H. Hwang. Rpim-sm: extending pim-sm for rp relocation. *Computer Communications*, 25(18):1774–1781, 2002.

[43] D. H. Linder and J. C. Harden. An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on computers*, 40(1):2–12, 1991.

[44] X. Liu. Multicasting algorithms for mesh and torus networks. Master's thesis, Concordia University, Canada, 2003.

[45] A. Malani. Efficient multicast algorithms for mesh and torus networks. Master's thesis, Concordia University, 2012.

[46] P. K. McKinley, H. Xu, A.-H. Esfahanian, and L. M. Ni. Unicast-based multicast communication in wormhole-routed networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1252–1265, 1994.

[47] P. Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Computing Surveys (CSUR)*, 30(3):374–410, 1998.

[48] P. Mohapatra and V. Varavithya. A hardware multicast routing algorithm for two-dimensional meshes. In *Eighth IEEE Symposium on Parallel and Distributed Processing*, pages 198–205. IEEE, 1996.

[49] R. Mukherjee and J. W. Atwood. Rendezvous point relocation in protocol independent multicast-sparse mode. *Telecommunication Systems*, 24(2):207–220, 2003.

[50] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[51] W. Putthividhya, W. Tavanapong, M. Tran, and J. Wong. Distributed core selection with qos support. In *International Conference on Communications*, volume 4, pages 2132–2137. IEEE, 2004.

[52] W. Putthividhya, M. Tran, W. Tavanapong, and J. Wong. Core selection with end-to-end qos support. In *ACM symposium on Applied computing*, pages 328–333. ACM, 2004.

[53] G. N. Rouskas and I. Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in communications*, 15(3):346–356, 1997.

[54] S. P. Sahoo and M. R. Kabat. Adaptive tabu search-based core selection algorithm in heterogeneous network. *International Journal of Swarm Intelligence*, 1(3):209–225, 2014.

[55] H. F. Salama. *Multicast routing for real-time communication of high-speed networks*. North Carolina State University, 1996.

[56] P.-R. Sheu and S.-T. Chen. A fast and efficient heuristic algorithm for the delay-and delay variation-bounded multicast tree problem. *Computer Communications*, 25(8):825–833, 2002.

[57] C. Shields and J. Garcia-Luna-Aceves. The ordered core based tree protocol. In *Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 884–891. IEEE, 1997.

[58] P. J. Slater, E. J. Cockayne, and S. T. Hedetniemi. Information dissemination in trees. *SIAM Journal on Computing*, 10(4):692–701, 1981.

[59] T. Takabatake. A hybrid bootstrap mechanism for multicast routing in pim-sm. In *Australasian Telecommunication Networks and Applications Conference*, pages 332–336. IEEE, 2007.

[60] Top500. Top500 lists june 2017, 2017.

[61] A. Vishnu, J. Daily, and B. Palmer. Designing scalable pgas communication subsystems on cray gemini interconnect. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, 2012.

[62] D. W. Wall. Mechanisms for broadcast and selective broadcast. 1980.

[63] H. Wang, X. Meng, M. Zhang, and Y. Li. Tabu search algorithm for rp selection in pim-sm multicast routing. *Computer Communications*, 33(1):35–42, 2010.

[64] N.-C. Wang and Y.-P. Hung. Multicast communication in wormhole-routed 2d torus networks with hamiltonian cycle model. *Journal of Systems Architecture*, 55(1):70–78, 2009.

[65] N.-C. Wang, C.-P. Yen, and C.-P. Chu. Multicast communication in wormhole-routed symmetric networks with hamiltonian cycle model. *Journal of Systems Architecture*, 51(3):165–183, 2005.

[66] S. Wang. Efficient multicast routing algorithms for mesh-connected multicomputers. Master's thesis, Concordia University, Canada, 2005.

[67] B. M. Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

[68] L. Wei and D. Estrin. The trade-offs of multicast trees and algorithms. In *International Conference on Computer Communications and Networks*, pages 17–24, 1994.

[69] X. Wu and V. Taylor. Performance characteristics of hybrid mpi/openmp scientific applications on a large-scale multithreaded bluegene/q supercomputer. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on*, pages 303–309. IEEE, 2013.

[70] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[71] D. Zappala, A. Fabbri, and V. Lo. An evaluation of shared multicast trees with multiple cores. *Telecommunication Systems*, 19(3):461–479, 2002.

[72] W. Zhengying, S. Bingxin, and Z. Erdun. Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm. *Computer communications*, 24(7):685–692, 2001.