INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

Quality of Service Routing On Wide Area Networks

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirement for the

degree of

Doctor of Philosophy

By

Young-Cheol Bang

Norman, Oklahoma

2000

Quality of Service Routing on Wide Area Networks

A DISSERTATION APPROVED FOR THE

SCHOOL OF COMPUTER SCIENCE

BY

Sridhar Radhakrishnan, Chair

S. Lakshmivarahan

Sudarshan K. Dhall

A. Das

M. J. Breen

Acknowledgments

First of all, I profoundly thank my father, Hong-Sik Bang, and mother, Il-Bong Park, who have been a tower of strength all throughout my education. This work is entirely dedicated to both of them.

I would like to express my sincere and heartfelt gratitude to Dr. Sridhar Radhakrishnan for guiding my research efforts. He guided me on various ideas in graph algorithms and computer networks. I am very grateful to him for his encouragement, kindness, and support during my research work. I am also indebted to Dr N. S. Rao for his technical support on my research.

I would also like to acknowledge Dr. S. Lakshmivarahan, Dr. S. Dhall, Dr. A. Das, and Dr. M. Breen,the other members of the advisory committee for their guidance and assistance. This acknowledgement is also extended to my friends Jong-Jun Sim, Jong-Hyun Kim, and Dr Man-Hoon Lee who had worked togather with me.

Saving the best for the last, I would also like to express my deepest gratitude to my family, my wife JaeHee Hwang, my elder son SungJe Bang, and my younger son Minje Bang for motivating me throughout my research.

Contents

List of Figures

Abstract

Routing is the process of sending a message from a source node to the destination node and the routing algorithm is a method to determine links that a message should be transmitted in order to reach the destination. The routing algorithm can be classified into the following three categories: unicast, multicast, and broadcast. Unicast involves sending a message from a given source to a destination; multicasting is a mechanism to send a message from a given source to a chosen set of destinations; broadcasting is sending a message from a given source to all the nodes in the network. Clearly, unicast and broadcast are special cases of multicast. The path selected by a routing algorithm depends on the application's Quality-of-Service (QoS) demands such as, end-to-end delay time, cost, delay jitter, and other factors.

Moore [20] introduced the quickest path problem and it has been studied extensively in recent times. The quickest path problem is to determine a routing path to minimize end-to-end delay from the source to the destination node taking into account message size, and propagation delay and bandwidth on the links of the network. Thus the quickest path is a path with minimum end-to-end delay time required to send σ units of message from a source node to the destination node.

The main theme of this dissertation is to investigate unicast and multicast routing algorithms in wide area networks. Towards this end, first we present a unifying quickest path algorithm for different message transfer modes at intermediate nodes. The source to destination path varies with message sizes. Quickest path algorithms build a table called the Path-Table that when searched with message size gives the minimum end-to-end delay path for that message size. Our second result deals with efficient construction of the Path-Table when a link or path bandwidth changes,

where path bandwidth is defined as the minimum of the bandwidths on the links of the path. Third, we present efficient algorithms for all-to-all quickest path problems in the presence of unreliable links in the network. By assigning probability of link failure to each link we can cast two problems namely, quickest most reliable path and most reliable quickest path.

Our fourth result deals with multicast routing in wide area networks. We have developed several heuristics for the construction of a multicast tree that minimizes end-to-end delay time taking into account message size, and propagation delay and bandwidths on links. We consider different modes of message transfers at intermediate nodes and for each type of intermediate node architecture we present heuristics for the multicast tree construction. The heuristics are simulated on large networks that are generated using different network generation models including Waxman I and II, Locality, and Transit-Stub. Our heuristics are shown to outperform existing heuristics that are based on shortest path and minimum spanning tree for multicast tree construction. Finally, we introduce a novel heuristic for the construction of a multicast tree with minimum cost in Internet like topologies. Our algorithm on directed asymmetric networks is shown to have a performance gain in terms of tree costs over existing algorithms.

Chapter 1

Introduction

1.1 Introduction to Routing in Wide Area Networks

Routing is the process of sending a message from a source node to the destination node and the routing algorithm is a method to determine links that a message should be transmitted in order to reach the destination. The path selected by a routing algorithm depends on the application's Quality-of-Service (QoS) demands such as, end-to-end delay time, cost, delay jitter, and other factors. There are several key characteristics [13], that are considered by a routing algorithm.

- **Optimality** Optimality is the condition by which the routing algorithm is capable of selecting the best route depending on the metrics and QoS.
- **Robustness and Stability** Routing algorithms must be able to withstand any unusual or unforeseen circumstances for which they must be robust and highly stable. If not, it can lead to conditions such as hardware failures,

high load conditions and incorrect implementations. As routers are located at network junction points, it should be make sure that they do not fail as this may cause considerable problems. Routing algorithms that can withstand the test of time and remain stable under a variety of network conditions are considered to be the best.

- **Convergence** Convergence is the process in which all the routers agree on the route to be followed which is the optimal route. It is a must that routing algorithms converge rapidly as otherwise they can lead to routing loops or network outages. When a network event goes down or becomes available, the routers distribute the routing update messages as a result of which new optimal routes are calculated which is agreed upon by all the routers.

- **Flexibility** Routing algorithms that are flexible adapt to a variety of network circumstances very quickly and at the same time accurately. Thus, when a network segment goes down, a flexible routing algorithm will quickly select the next best path for all routes using the same segment.

- **Simplicity** Routing algorithms designed should be as simple as possible for ease of utilization. Routing algorithm is considered to be efficient if the software and utilization overhead are minimal. Routing algorithm needs to be highly efficient when run on a computer with limited physical resources.

The routing algorithms can be broadly classified into the following three categories: unicast, multicast, and broadcast routing algorithms. The unicast routing algorithm is to find a path from single source to single destination, and the broadcast routing algorithm is to construct a broadcast tree rooted at the source node

spanning all destinations (all other nodes in a network), where broadcasting is to send the same message to all destinations. Multicast in a network is the process of sending the same information from a source node to a set of destination nodes called multicasting group. There are two types of multicast: one-to-many and many-to-many. The typical algorithm for one-to-many multicast is to construct a spanning tree rooted at the source covering all destinations. In the case of many-to-many multicast, the following two alternative approaches are generally used [38]. One approach called *Source-specific multicast* is to construct one-to-many multicast tree for each source, and alternative method named *Shared multicast* is to construct only one multicast tree, where traffic streams from multiple sources share the links of the same tree.

In recent years, the emergence of cost effective audio and video hardwares and high speed networks have introduced multimedia computer applications, such as teleconference and remote education which require real-time transmission as a QoS. So in new communication services involving multimedia applications, it is critical to minimize the end-to-end delay for sending messages in real-time.

Consider a link, connecting a node s with another node t, having a bandwidth B bps and a propagation delay D. Even though it is very short, there is always finite time delay on the link to propagate message from one node to the other node, which is known as the propagation time [21]. As explained in [21], if data propagates through medium without any resistance, the speed is $3 \times 10^8\ ms^{-1}$ which is the speed of the light. If twisted-pair wire or coaxial cable is used, the speed is typically in

the region of 2 x 10^8 ms^{-1} which is the speed of the electrical signal. Thus,

$$D = \frac{distance\ in\ meter}{velocity\ of\ propagation\ of\ medium\ used}$$

For example, if 1000 bits of data is transferred from node a to node b via twisted-pair wire or coaxial cable with transmission rate of 10 kbps, and distance between a and b is 100 m, then

$$D = \frac{100\ m}{2\ \text{x}\ 10^8 ms^{-1}} = 5\ \text{x}\ 10^{-7}\ s.$$

If s wants to send a packet with σ bits to t using this link, then it takes σ/B seconds for s to transmit the packet. It takes D seconds for the last bit of the packet to arrive at t. Thus, the last bit of the packet reaches t at time

$$T = \sigma/B + D,$$

where T is the end-to-end delay from s to t [42, 43]. For this reason, routing algorithms should take into account message size, propagation time, and bandwidth on each link over the computer network. If a path with multiple links is used to send a packet, then the end-to-end delay of the path is dependent on the transfer mechanism used by intermediate switches along the path (details are presented in section 2).

To transfer messages, there are two basic switching systems that are used: circuit switching (pipeline) and packet switching (store-and-forward). When data is delivered via circuit switching, bit streams of data are transferred with fixed rate from a source to a destination without buffering time. If data is routed over the packet switched network, entire data is stored at every intermediate node before forwarding to next node. The telephone networks belong to circuit switching, and the IP (Internet Protocol) computer networks belong to store-and-forward. The end-to-end

delay of a path can be computed by the formula $T = \sigma/B + D$ in the circuit switching. Since the circuit switching transfers the data along the route with fixed rate, the minimum bandwidth of link along the path is selected as the path-bandwidth B. In the case of packet switching, since incoming data is stored temporarily at each node and then transmitted to outgoing link, transmission time $\sigma/B(e)$ at each node v is required, where e is the outgoing link of v. Thus the path-bandwidth B is $\frac{1}{\sum_{e \in P} \frac{1}{B(e)}}$, where P is the routing path and e is the link on P. The propagation delay D of the path, is computed by adding all propagation delays of links along the path in both circuit switching and packet switching. Since path-bandwidth B is computed in a different way for each switching system, transfer mechanism of the switching system being used should be considered in order to find the optimal path. With the advent of ATM (Asynchronous Transmission Mode) and active network technologies, combinations of circuit switching and packet switching are presently considered, and this is the motivation for the other modes having different transfer mechanisms.

Before we introduce the message transfer modes, the terminologies used in this dissertation is first introduced. We consider a network represented by a graph $G = (V, E)$ with n nodes and m edges or links. Each link $e = (i, j) \in E$ has a *bandwidth* or *capacity* $B(e) \geq 0$ and a $link - delay$ $D(e) \geq 0$, and we assume that the source has adequate global information involving the topology of the network, the characteristics of links such as delay and bandwidth, and all links in the network are error-free. A message is sent as a continuous stream along the edge e at a constant flow rate denoted by $f_e \leq B(e)$. A message of length σ units can be sent along the edge e at flow rate f_e in $\sigma/f_e + D(e)$. The flow rates can be different in

different edges, and the message can be delayed at the nodes.

Consider a *simple path* P from $s = i_0$ to $t = i_k$ given by $(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)$, where $(i_j, i_{j+1}) \in E$, for $j = 0, 1, \ldots, (k = 1)$, and all i_0, i_1, i_k are distinct. The delay experienced by a message sent via P depends on the *message forwarding mechanism* used by the intermediate nodes. For a node v on P, let $B_{in}(v)$ and $B_{out}(v)$ be the bandwidths of incoming and outgoing edges, respectively, and $f_{in}(v)$ and $f_{out}(v)$ be the flow rates of incoming and outgoing message. We consider the following five modes. The timing diagrams of arrival and departure of a message at an intermediate node v are shown for all modes in Figure 1.1. The *delay* of P is $D(P) = \sum_{j=0}^{k-1} D(e_j)$, where $e_j = (i_j, i_{j+1})$. Let $|P|$ denote the number of nodes of P.

The objective of our study is to develop unicast and one-to-many multicast routing algorithms with respect to optimality (the end-to-end delay, the minimum cost, etc) taking into account message size, and propagation delay and bandwidth on each link over the network.

1.2 The Transfer Modes

The various modes abstract different mechanisms used in the data networks. The classical circuit switching involves no buffering, and the store-and-forward is the other extreme wherein the message is buffered in its entirety at every intermediate node. With the emergence of ATM (Asynchronous Transmission Mode) and active network technologies, the transfer mechanisms take into account combinations of these modes presently, which motivates the other modes. Transfer modes we consider in our research as follow:

- **I. Circuit Switching:** In mode I, the message is sent at a constant rate from s to d with no buffering at intermediate nodes. The *bandwidth of type* I of P is $B^I(P) = min_{j=0,1,\ldots,k-1} B(e_j)$. The *end-to-end delay* in mode I of path P in transmitting a message of size σ is $T^I(P) = \sigma/B^I(P) + D(P)$. Thus, $f_e = B^I(P)$ for all e on P, and $f_{in}(v) = f_{out}(v) = B^I(P)$ for all v on P except the end nodes.

- **II. Earliest Departure:** A message received at an intermediate node v is sent out at a rate equal to the minimum of the incoming rate and outgoing bandwidth, i.e. $f_{out}(v) = min\{f_{in}(v), B_{out}(v)\}$. If the outgoing bandwidth is smaller than the incoming rate, then the message is suitably buffered and sent at a lower rate without any delay. In mode IIa, the retransmission at a lower bandwidth starts only after the entire message is received at v under the condition $f_{in}(v) > B_{out}(v)$; but, the retransmission at the incoming rate is without any delay under the condition $f_{in}(v) \leq B_{out}(v)$.

- **III. Full Outgoing Bandwidth:** A message received at an intermediate node v is retransmitted immediately at the outgoing rate if $f_{in}(v) = B_{in}(v) \geq B_{out}(v)$, and is buffered and suitably delayed to be sent at the rate of the outgoing bandwidth otherwise, i.e. $f_{out}(v) = B_{out}(v)$. In this mode the flow in any edge is equal to its bandwidth, i.e. $f_e = B(e)$ for all e on P. Mode IIIa is same as III, except when the outgoing bandwidth is higher than incoming flow rate, in which case the message is completely buffered at v before it is sent out at the rate of $B_{out}(v)$.

- **Store-Forward:** A message sent along an edge (u, v) will be received in its entirety at v before it is sent from v such that $f_e = B_e$ for all e on P. The *bandwidth of type* IV of P is $B^{IV}(P) = \frac{1}{\sum_{j=0}^{k-1} \frac{1}{B(e_j)}}$. The *end-to-end delay* in mode IV of path P in transmitting a message of size σ is $T^{IV}(P) = \sigma/B^{IV}(P) + D(P)$.

1.3 The Quickest Paths

Mode I has been studied under the title *quickest path problem* by Chen and Chin [11], Rosen *et al* [37], Rao and Batsell [32, 34, 33], Bang *et al* [5], Hung and Chen [9, 22], Martins and Santos [17]. The *quickest path problem* is to find a routing path in a network G such that the end-to-end delay time required to send σ units of message from a source to a destination is minimum. The quickest path problem is very similar to the shortest path problem and can be computed by using the shortest path algorithm; however, the quickest path problem does not have the optimal-substructure property of shortest paths such that any subpath of a shortest path is a shortest path [1]. Figure 1.2 shows such an example and how to compute the quickest paths.

Let a and f be a source and a destination, respectively. Each link is associated with two numbers; the first number is the link delay and the second number is the link bandwidth. Let $\sigma = 100$. Then the time required to be arrived at node f from a through path (a, b, c, f) ,$T(a, b, c, f)$, is $100/2 + 30 = 80$. The time through path (a, b, e, f), $T(a, b, e, f)$, is $100/4 + 35 = 60$. $T(a, d, e, f)$ is $100/2 + 40 = 90$, $T(a, g, e, f)$ is $100/4 + 40 = 65$, and $T(a, g, h, f)$ is $100/4 + 50 = 75$. Thus the quickest

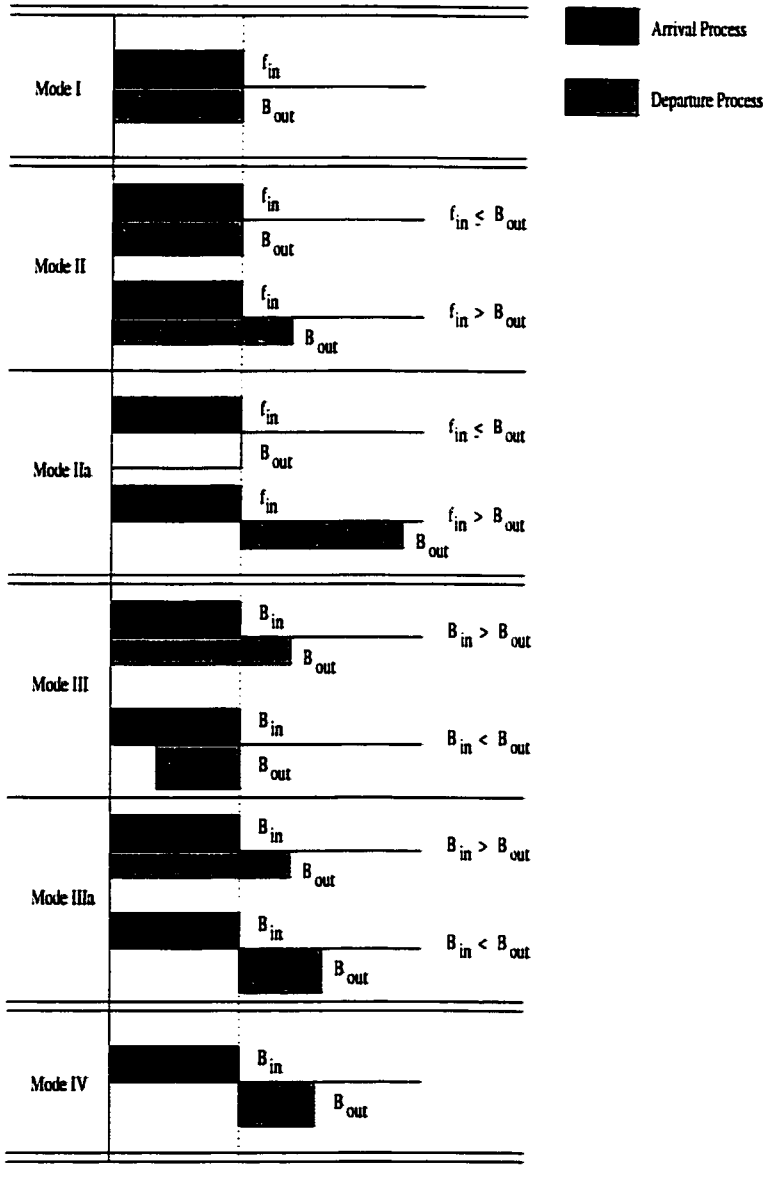

Figure 1.1: Arrival and departure timing diagrams at an intermediate node for various modes.

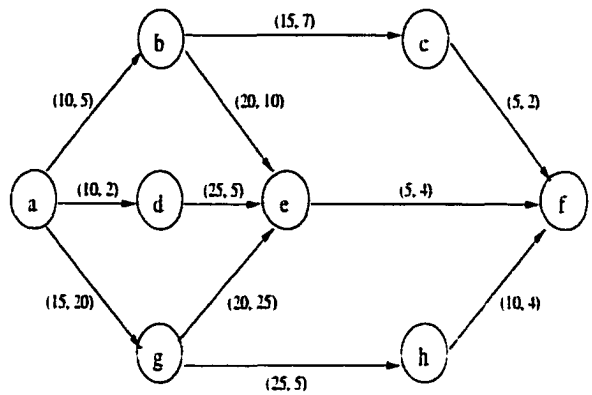

Figure 1.2: An example of the quickest path problem. .

path from a to f is the path (a, b, e, f). Even though the quickest path from a to f is (a, b, e, f), the path (a, b, e) is not the quickest path from a to e, since $T(a, b, e) = 100/5 + 30 = 50$, $T(a, d, e) = 100/2 + 35 = 85$, and $T(a, g, e) = 100/20 + 35 = 40$. Thus the quickest path from a to e is the path (a, g, e).

Assume that $B_1 < B_2 < \ldots < B_r$ denote the distinct values of the bandwidth $B(e)$, $e \in E$ and G_{B_i} represents the subnetwork where bandwidth of all links in the subnetwork are greater than or equal to B_i with $1 \leq i \leq r$. The following observation was made in [32].

Observation 1.3.1 *Let $\mathcal{P} = \{P_{B_1}, P_{B_2}, \ldots, P_{B_r}\}$ denote the set of shortest path, where P_{B_i} is the shortest path in G_{B_i} with $1 \leq i \leq r$. Then we have $D(P_{B_1}) \leq D(P_{B_2}) \leq \ldots \leq D(P_{B_r})$, and $B(P_{B_1}) < B(P_{B_2}) < \ldots < B(P_{B_r})$.*

Consider two paths P_1 and P_2. If $D(P_1) = D(P_2)$ and $B(P_1) < B(P_2)$ then $\sigma/B(P_1) + D(P_1) < \sigma/B(P_2) + D(P_2)$. This means that P_1 is always slower than P_2 to send σ units. Thus, by above observation [32], we can assume that $D(P_{B_1}) < D(P_{B_2}) < \ldots < D(P_{B_r})$ and $B(P_{B_1}) < B(P_{B_2}) < \ldots < B(P_{B_r})$ by removing the equality. In addition, if $D(P_1) > D(P_2)$ and $B(P_1) = B(P_2)$ then P_1 cannot be the

quickest path for any σ units. Thus, the quickest path P to transmit σ units is nothing but the shortest path in $G_{B(P)}$. Observation in [32] and following theorem in [37] lead to the efficient quickest path algorithm.

Theorem 1 *Let P be a quickest path from s to t in G to send σ units of data. Then*

(1) P is a shortest path from s to t in $G_{B(P)}$;

(2) Any subpath of P must itself be a shortest path in $G_{B(P)}$.

Thus, algorithm to compute the quickest path from s to t is as follows;

Algorithm Single Pair Quickest Path [37]

INPUT : Network, Source s, Destination d, and Message Size σ

OUTPUT : Quickest Path from s to d. 1. **for** $i = 1, \ldots, r$ **do**

2. compute a shortest path P_i from s to t in $G_{B(P_i)}$.

3. compute the path-delay $D(P_j)$ and the path-bandwidth $B(P_j)$.

4. **endfor**

5. compute index k which minimizes $\{\sigma/B(P_i) + D(P_i) | i = 1, 2, \ldots, r\}$

6. return P_k as the quickest path in G to send σ units of data.

7. **end Algorithm**

When a shortest path P_i is computed in step 2, any shortest path algorithm can be used so that Dijkstra's shortest path algorithm can be adapted. Since Dijkstra's algorithm results in one-to-all shortest paths, the time taken to compute one-to-all quickest paths is the same as the time for single pair quickest path. Thus, in the

case of all-to-all quickest paths, it can be computed in $O(nm^2 + mn^2 logn)$ using algorithm Single Pair Quickest Path. However, according to algorithms introduced in [29, 10], the time complexity can be further reduced to $O(n^2 m)$.

1.4 Multicast Routing

Multicast in a network is the act of sending the same information from a source node to a set of destination nodes called *multicasting group*. New communication services involving multicast and multimedia applications (**High-Bandwidth** applications) are becoming widespread. In applications such as the retrieval of medical images from remote repositories, and remote teleoperated robots, it is critical to minimize the end-to-end delays in transmitting messages. Furthermore, if a variety of applications are supported over the same network, it is important to achieve minimum end-to-end delays for small messages such as robot control packets as well as large messages such as image/video data. These applications also need very large reserved bandwidth to satisfy their Quality-of-Service (QoS) requirements. In our study of *multicast problem*, we consider limitations in network resources that make it critical to design multicast routing paths using the resources such as optimal bandwidth and the minimum end-to-end delays to transmit messages. To achieve the minimum end-to-end delays, we consider a framework wherein bandwidth can be reserved on the communication links, and, once reserved, is guaranteed for the required time period. At the additional cost of bandwidth reservation and guarantees, our framework enables *deterministic* end-to-end delay guarantees. Such mechanisms can be naturally supported in ATM networks [28] whereas in situations such as the

Internet, require additional mechanisms (for example, RSVP [51] and/or specific queuing implementations at the routers [8]).

The general problem of multicasting is well-studied in the areas of computer networks and algorithmic network theory. Depending on the specific cost or criterion, the multicast problem can be of varying levels of complexity. The Steiner tree studied extensively in network theory deals with minimizing the "cost" of a tree that connects a source to a subset of nodes of a network [46, 23, 36, 39, 50]. The Steiner tree has a natural analog of the general multicast-tree in computer networks [24, 45, 26, 52]. The computation of Steiner tree is NP-complete, and an interesting polynomial-time algorithm for a fixed parameter has been proposed in [31] (wherein an overview of several other approximation algorithms are also provided); distributed algorithms based on Steiner heuristics are provided in [6]. More generally, the minimization problems where the cost is based on "entire subtrees", such as the Steiner tree problems, are computationally intractable [19]. On the other hand, the end-to-end delay between s a given source and all $d \in MC$, is significantly easier algorithmically, where MC is the multicast group. Typically, the shortest path based algorithms are used to generate the best paths with the minimum-end-to-end delay, and Dijkstra's shortest path algorithm [1] is widely used. Using the shortest path based algorithms, a multicast tree is constructed by merging optimal paths from the source to each of the destinations. Such a tree can be constructed in polynomial time [15]. Also note that our framework is different from the dynamic frameworks which utilize feedback mechanisms [25, 12, 4] to provide only "soft" guarantees.

Following two algorithms called KMB [27] and SPH [31] are the most well known heuristics based on Steiner tree and shortest paths.

Algorithm KMB [27]

INPUT : An undirected graph G, source s, and multicast group MC

OUTPUT : A Steiner tree T_{KMB}

1. Construct the complete undirected distance graph G' with s and MC,
   where cost of each link (u, v) is equal to the cost of the minimum cost path
   from u to v.
2. Find the minimum spanning tree T' from G'.
   (If there are several minimal spanning tree, pick an arbitrary one.)
3. Construct G_{SG} by replacing each edge in T'
   by one of its corresponding minimum cost paths in G.
4. Find the minimum spanning tree T_{SG} of G_{SG}.
   (If there are several minimal spanning trees, pick an arbitrary one.)
5. Construct a Steiner tree T_{KMB} from T_{SG} by deleting edges in T_{SG}, if necessary.
6. **end Algorithm**

As an example of KMB, consider Figure 1.3. Given a network $G = (V, E)$ in Figure 1.3-(a), where V is the set of nodes and E is the set of edges, the number on each edge represents a cost. Let the source and MC be 1 and $\{6, 7, 8\}$, respectively. A complete graph G' in Figure 1.3-(b) is constructed by step 1 in algorithm KMB. Figure 1.3-(c) shows the minimum spanning tree T' of G' by step 2. G_{SG} in Figure 1.3-(d) is constructed by replacing each edge in T' by its one of corresponding minimum cost paths in G. The minimal spanning tree T_{SG} of G_{SG} is shown in Figure

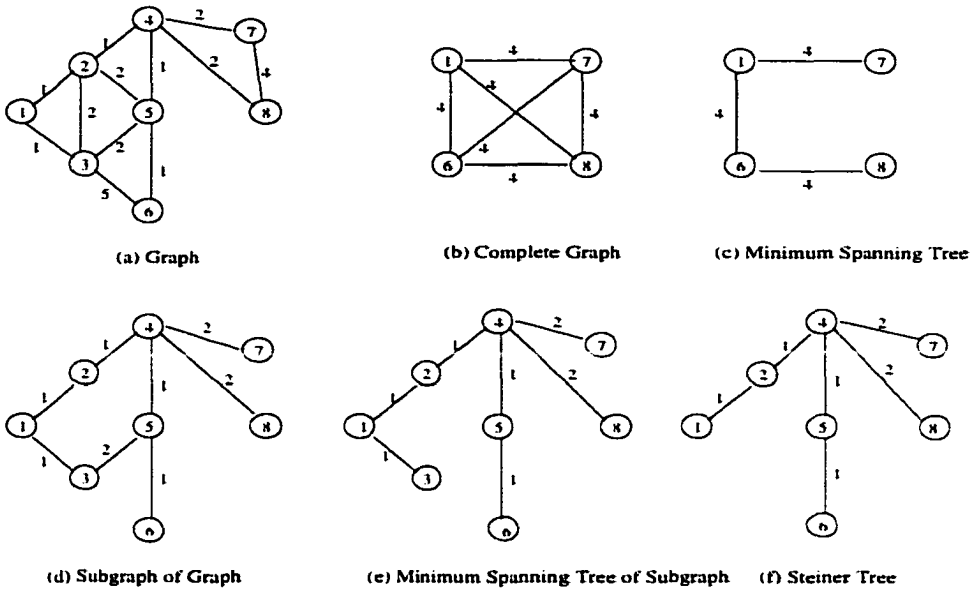

Figure 1.3: Illustrative example of KMB.

1.3-(e). After removing unnecessary edges in T_{SG} by step 5, the final Steiner tree T_{KMB} is constructed and Figure 1.3-(f) shows T_{KMB}.

Algorithm SPH

INPUT : Any network G, source s, and multicast group MC

OUTPUT : A multicast tree T_{SPH}

1. Find the single source shortest path tree T for G, rooted at s,
   using Dijkstra's shortest path algorithm [1].
2. Delete unnecessary edges and nodes from T.
3. Return the multicast tree T_{SPH}
4. **end Algorithm**

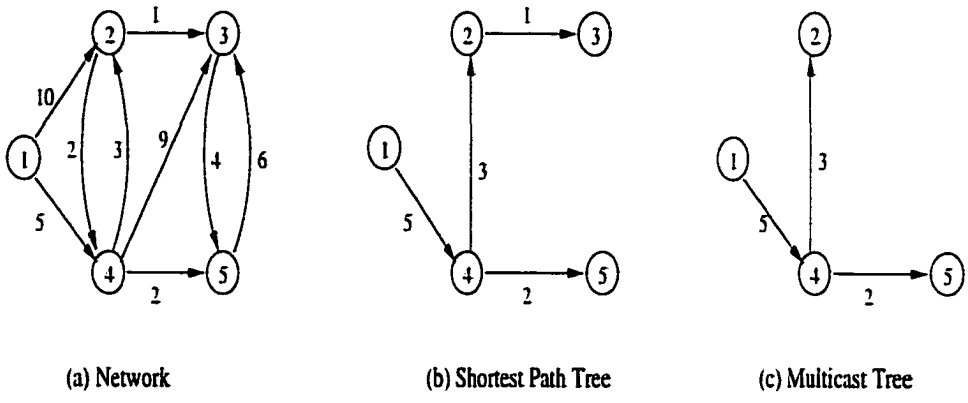

Figure 1.4: Illustrative example of SPH.

On the other hand, in Figure 1.4, (a) represents the given network. The shortest path tree T for G is shown in (b). Since T contains unnecessary link $(2,3)$ and node 3, T_{SPH} can be constructed by removing the link $(2,3)$ and the node 3.

The construction methods for the multicast trees can be summarized as follows: In most cases, the Steiner tree based algorithms have been used to construct the multicast tree with the minimum costs, and the shortest path based algorithms have been used to construct the multicast tree with the minimum end-to-end delays.

1.5 Topology Models

In the literature, many network models are introduced to represent actual networks. Broadly, these network models include regular topologies such as trees, rings, and stars, Well-known topologies such as the **ARPA**net or **NSF**net backbone, and randomly generated topologies. Here, to implement the simulation, four different randomly generated topology models to represent networks have been used. These four models are called WaxmanI [45], WaxmanII [45], Locality [49], and Transit-Stub

[49]. Since WaxmanI and WaxmanII have almost similar characteristics, these two models are explained together.

1.5.1 Waxman Models

To generate a network using WaxmanI method proposed by Waxman, n nodes are randomly distributed in a plane which can represent the size of the area to be simulated. Then links are placed with the probability

$$P(u, v) = \alpha e^{\frac{-l}{(\beta L)}}$$

, where L is the maximum possible distance between any two nodes in the network and $l(u, v)$ represents the Euclidean distance from u to v. The parameter α is selected in the range (0, 1), β can be chosen in the range(0, 1]. Since an increase in value of α and a large value of β increases the number of links and the number of long links in the network respectively, the varying parameter values should be appropriately chosen to obtain the desired random networks.

The WaxmanII model can be obtained by replacing l by a random number in the range (0, L). Thus, the probability of an link between nodes, u and v, is given by

$$P(u, v) = \alpha e^{\frac{-rand(0,L)}{(\beta L)}}.$$

1.5.2 Locality Model

To obtain Locality model, discrete links are partitioned based on length. Then, locality can be captured by relating the link probability to the distance between any two nodes, i.e, we can obtain locality by assigning a different fixed probability for

each equivalence class of link lengths. Thus, for given set of nodes, links are placed with the probability

$$P(u,v) = \begin{cases} \alpha & \text{if } d < r \\ \beta & \text{if } d \geq r \end{cases}$$

1.5.3 Transit-Stub Model

This model proposed by [49] describes the current Internet very well. It first constructs a connected random network using any random network model and each node in the random network generated represents an entire transit domain. To create the backbone topology for every transit domain, each node, which represents an entire transit domain, is replaced by a newly generated random network. For each node in the backbone of its transit domain, a number of random networks, called stub domains which are connected to that node, are generated. Then, additional links are generated between any pair of nodes, where one is from a transit domain and the other is from a stub domain, or both nodes are from two different stub domains. Figure 1.5 shows the example of Transit-Stub model.

As shown in Figure 1.5, transit domain represents the backbone of the Internet and each backbone node in a transit domain connects to a number of stub domains through nodes, called gateway, in the stub domains.

1.6 Organization of the dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we present an unifying minimum end-to-end delay routing algorithm for different message transfer modes which runs in $O(rm + rnlogn)$, where r is the number of distinct link-

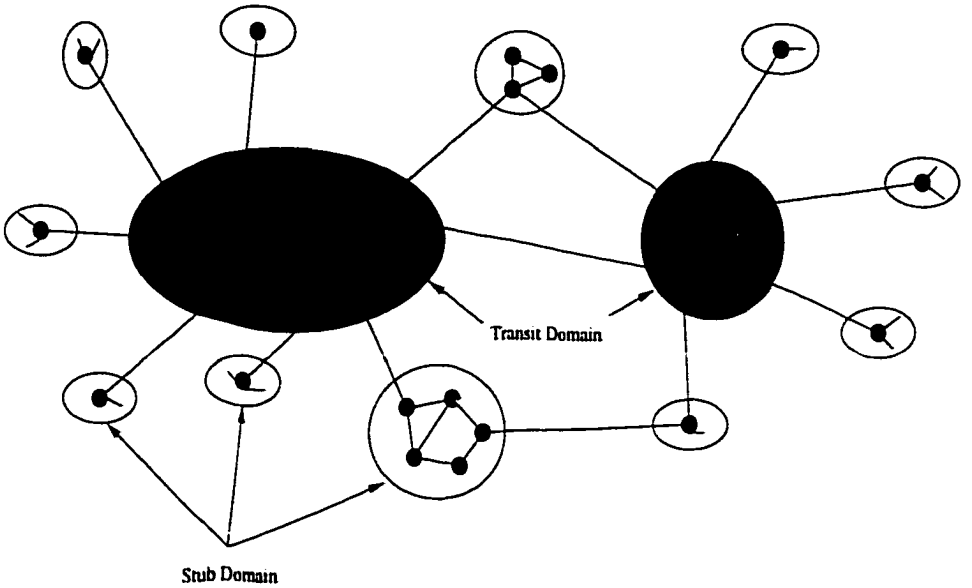

Figure 1.5: Example of Transit-Stub Model.

bandwidth, and n and m are the number of nodes and links in the network, respectively. Chapter 3 contains two newly developed algorithms. The first one is used for constructing the path-table containing the minimum end-to-end delay path for any particular message size, and the second one is to update the path-table when a link or path bandwidth changes, where the path bandwidth is defined as the minimum of the bandwidths on the links of the path. In chapter 4, we present efficient algorithms for all-to-all quickest path problems in the presence of unreliable links in the network. By assigning probability of link failure to each link we can cast two problems namely, quickest most reliable path and most reliable quickest path. Chapter 5 deals with multicast routing in wide area networks. We have developed several heuristics to build a multicast tree that minimize end-to-end delay time taking into account message size, and propagation delay and bandwidths on links. We consider different modes of message transfers at intermediate nodes and for each type of inter-

mediate node architecture we present heuristics for the multicast tree construction. The heuristics are simulated on large networks that are generated using different network generation models including Waxman I and II, Locality, and Transit-Stub. Our heuristics are shown to outperform existing heuristics that are based on shortest path and minimum spanning tree for multicast tree construction. In chapter 6, we introduce a novel heuristic for the construction of a multicast tree with minimum cost in Internet like topologies. Our algorithm on directed asymmetric networks is shown to have a performance gain in terms of tree costs over existing algorithms. Finally, conclusions are presented in chapter 7.

Chapter 2

Quickest Paths Under Different Transfer Modes

2.1 Introduction

In this chapter we present an unifying algorithm to compute the end-to-end delay of the quickest paths under different message transfer modes for given message size σ. As explained in chapter 1, mode I corresponds to the classical circuit switching which involves no buffering, and mode IV is the store-and-forward wherein the message is buffered in its entirety at every intermediate node. In mode II the message is circuit switched if there is sufficient outgoing bandwidth, and sent a reduced flow level otherwise. Such reduction of flow in circuit switching is sometimes difficult, and the message may have to be received in full before the retransmission as in mode IIa. In modes III and IIIa, the main focus is to avoid the fragmentation of the bandwidth, and hence the entire bandwidth of each edge is utilized during the

message transmission along the edge.

A path of mode i is referred to as i-path, and a path with the least *end-to-end delay* among all i-paths is referred to as the i-quickest path, for i = I, II, IIa, IIIa, IV. Let P_σ^i denote the i-quickest path in mode i for the message size σ.

A II-path can be converted into I-path by utilizing the flow rate corresponding to the edge with the minimum bandwidth. A I-path can be converted into II-path by suitably increasing the flow rate starting from s and repeatedly moving along P until the next edge with a lower bandwidth is reached and then lowering the flow. During these flow reductions, the *end-to-end delays* of the path P remain the same. In general for any path P, we have the following inequalities on the *end-to-end delays* of various modes:

$$T^I(P) = T^{II}(P) \leq T^{III}(P) \leq T^{IIIa}(P) \leq T^{IV}(P)$$

and

$$T^{II}(P) \leq T_{IIa}(P) \leq T^{IV}(P)$$

Example 2.1 Consider the network in Figure 2.1-(a) which consists of two paths P_1 and P_2 from s to d. The *end-to-end delays* of paths under various modes are given the following table.

Path	Mode I	Mode II	Mode IIa	Mode III	Mode IIIa	Mode IV
P_1	$\frac{\sigma}{1}+5$	$\frac{\sigma}{1}+5$	$\frac{\sigma}{4/5}+6$	$\frac{\sigma}{1}+5$	$\frac{\sigma}{4/5}+5$	$\frac{\sigma}{2/3}+5$
P_2	$\frac{\sigma}{4}+6$	$\frac{\sigma}{4}+6$	$\frac{\sigma}{20/7}+6$	$\frac{\sigma}{20/6}+6$	$\frac{\sigma}{20/7}+6$	$\frac{\sigma}{5/2}+6$

For this case, we have

$$T^{III} < T^{IIa}(P) = T^{IIIa}(P)$$

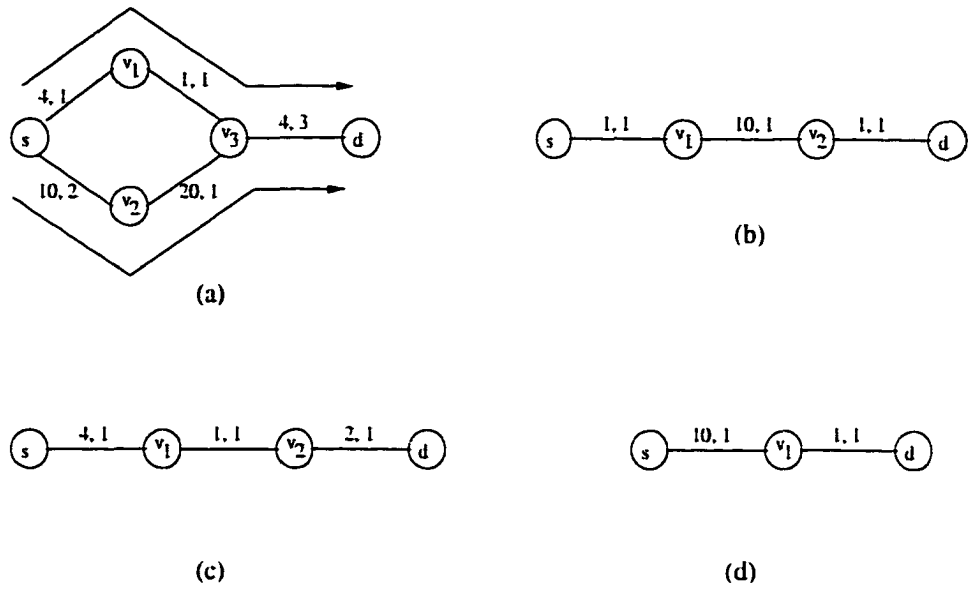

Figure 2.1: Example 2.1 .

for $P = P_1, P_2$. In general, however, the *end-to-end delays* in modes IIa and III (or IIIa) do not obey ordering as illustrated below. For the network in Figure 2.1-(b), we have $T^{IIa}(P) = \sigma + 3$, $T^{III}(P) = \frac{\sigma}{10/19} + 3$, and $T^{IIIa}(P) = \frac{\sigma}{1/2} + 3$. Hence, we have

$$T^{IIa}(P) < T^{III}(P) < T^{IIIa}(P).$$

For the network in Figure 2.3-(c), we have $T^{IIa}(P) = \frac{\sigma}{4/5} + 3$, $T^{III}(P) = \frac{\sigma}{1} + 3$, and $T^{IIIa}(P) = \frac{\sigma}{2/3} + 3$. Hence, we have

$$T^{III}(P) < T^{IIa}(P) < T^{IIIa}(P).$$

For the network in Figure 2-3(d), we have $T^{IIa}(P) = \frac{\sigma}{10/11} + 2$, $T^{III}(P) = \sigma + 2$, and $T^{IIIa}(P) = \sigma + 2$. Hence, we have

$$T^{III}(P) = T^{IIIa}(P) < T^{IIa}(P).$$

2.2 Computation of Quickest Paths

We now present an algorithm to compute the end-to-end delay of the II-quickest path for given message size σ. The path P_σ^{II} itself can be constructed by suitably maintaining the predecessor pointers as in the case of Dijkstra's algorithm [16]. The other quickest paths (except IV) can be computed using minor variations of this algorithm. For initialization, consider distinct $u, v \in V$. If (u, v) is an edge e then define $DE(u, v) = D(e)$, and if not $DE(u, v) = \infty$. Similarly, if (u, v) is an edge e then define $BE(u, v) = B(e)$, and if not $BE(u, v) = 0$. Let $b_1, b_2, \ldots, b_c$ denote the distinct values of the bandwidths $B(e)$, $e \in E$. Each node v is represented by an array $TE[v][.]$ such that $TE[v][b]$, for $b = b_1, b_2, \ldots, b_c$, is the time at which the trailing edge of the message reaches v at a flow rate b. Note that the flow rate at nodes in between s and v must be at least b in this mode. Since the message is not delayed at any intermediate node, the flow rate once reduced will stay at this value or further reduced subsequently. As a result, if the message is received at a flow rate of b at d, then $T^{II}(P) = \sigma/b + D(P)$.

Algorithm Quick-II(σ)

1. A $\leftarrow$ s

2. **for** each vertex $v \in V$ - s **do**

3. **for** $b = b_1, b_2, \ldots, b_c$ **do**

4. $TE[v][b] \leftarrow \infty$;

5. $TE[v][B(s, v)] \leftarrow \sigma/BE(s, v) + DE(s, v)$;

6. $CT[v] = TE[v][BE(s, v)]$;

7. $BT[v] = BE(s, v)$;

8. **while** $A \neq V$ **do**

9. choose a vertex $v \in V - A$ such that $CT[v]$ is minimum;

10. $b_v \leftarrow BT[v]$;

11. **if** $CT[v] \geq TE[v][b]$ for all $b = b_1, b_2, \ldots, b_c$ then add w to A;

12. **for** each $w \in V - A$ **do**

13. **if** $b_v \leq BE(v, w)$ **then**

14. $TE[w][b_v] \leftarrow \min\{TE[w][b_v], TE[v][b_v] + DE(v, w)\}$

15. **if** $TE[w][b_v] < CT[w]$ **then**

16. $CT[w] \leftarrow TE[w][b_v]$;

17. $BT[w] \leftarrow b_v$;

18. **else**

19. $TE[w][B(v, w)] \leftarrow \{TE[w][B(v, w)], TE[v][b_v] - \sigma/b_v + DE(v, w) + \sigma/B(v, w)\}$

20. **if** $TE[w][BE(v, w)] < CT[w]$ **then**

21. $CT[w] \leftarrow TE[w][BE(v, w)]$;

22. $BE[w] \leftarrow BE(v, w)$;

23. return($min_b\{TE[d][b]\}$)

24. **end Algorithm**

The outline of algorithm is as follows. The algorithm has at most $(n - 1)c$ iterations, and in each iteration a node v with bandwidth b_i for some $b_i = b$, is selected such that the path from s to v, denoted by P_v, has the least end-to-end

delay (line 9). Once v is selected, every vertex $w \in V - A$, is examined to see if the path P_v can be extended to w to result in a smaller end-to-end delay in lines 13-22. In particular, if the bandwidth of (v, w) is higher than b, then flow rate of b is used along (v, w) (line 14), and a lower flow rate of $B(v, w)$ is used otherwise (line 19). In either case the extension of the path from v to w can result in a flow into w that is no more than b. Once the end-to-end delay of v is known for all $b = b_1, b_2, \ldots, b_c$, it is added to A and is not considered further (line 11). For the sake of this algorithm, the infinites are handled according to the following rules: $1 \ / \ 0 = \infty$, $\infty + \infty = \infty$, and $\infty \leq \infty$ are true.

Theorem 2 *Algorithm Quick-II computes the end-to-end delay of II-quickest path for transmitting a message of size σ from s to every node in $O(m^2 + mn\log n)$ time.*

Proof: Let *flow* of a II-path P at d be defined as $f(P) = f_{in}(d)$. Let A_b denote all nodes that have been selected in line 9 with $BT[v] \geq b$. We establish the correctness by induction on the size of the set A_b. Specifically, for each $v \in A_b$, we show that $TE[v][b]$ is the shortest end-to-end delay from s to v among all paths with flow rate of b at v. Moreover, for all $v \in V - A_b$, $TE[v][b]$ is the lowest end-to-end delay from s to v with flow rate of b at v such that the path is wholly within A_b, except for v itself. The II-path with the lowest end-to-end delay among such paths is chosen in line 23. Then, the correctness of the algorithm follows by noting that II-quickest path must have a bandwidth equal to one of b_i's at v.

The claim is true for $|A_b| = 1$ since the message is already at s, it takes 0 time for the transmission. Among the paths with bandwidth b_v, the edge (s, v) is the quickest since any other path with the same bandwidth will not have shorter delay.

Moreover, a path from s to $v \in V - A_b$, that is wholly contained in A_b except for v, is the single edge from s.

Assume that the hypothesis is true for the current set A_b, when v is chosen in line 9. Then we have $TE[v][b_v] \leq TE[w][b_i]$ for any $w \in V$ - A_b and any $i = 1, 2, \ldots, c$. Consider that $TE[v][b]$, corresponding to path P_v, is not the minimum end-to-end delay. Then, there must be a II-path P with $f(P) = b$ with a smaller end-to-end delay. Furthermore, P must contain a node not contained in A_b, and let w be the first node on P from $V - A_b$ while moving from s to v. Now let us split the path P into the path from s to w, denoted by P_w, and the path from w to v. Since P is II-path, $f(P_w) \geq f(P)$, and by the positiveness of link-delays we have $D(P_w < D(P_v)$. Then, we have

$T^{II}(P_w) = D(P_w) + \sigma/f(P_w) < D(P) + \sigma/f(P)$.

Since P has shorter end-to-end delay than P_v, we have $D(P_w) + \sigma/f(P_w) < D(P) + \sigma/f(P)$. Thus we have $TE[w][b'] < TE[v][b]$ for some $b' \geq b$, which is contradiction.

This algorithm consist of at most $c \leq m$ instances of the Dijkstra's each corresponding to $b = b_1, b_2, \ldots, b_c$ algorithm with interleaved steps. The time complexity follows directly from that of Dijkstra's algorithm [16]. □

The algorithm for mode IIa is obtained by replacing the line 19 of Quick-II by the following line. 19. $TE[w][B(v,w)] \leftarrow \min\{TE[w][b_v], TE[v][b_v] + D(v,w)\}$;

Even though mode I and mode II use different transfer mechanisms, since both modes use the pipeline scheme, Algorithm Quick-I can be obtained by using Quick-II. Quick-I can be viewed as a variation of the algorithm of [37], which is based on c instances of Dijkstra's algorithm, executed separately. Let $G(b) = (V, E(b))$ denote the subnetwork where $e \in E(b)$ if and only if $B(e) \geq b$. Let a $s - d$ *path* in $G(b)$

denote the shortest delay path based only on the link-delays. The $s-d$ *path* are independently computed in each $G(b)$ for each $b = b_1, b_2, \ldots, b_c$. Then I-quickest path is selected to be the one with lowest-end-to-end delay among the c paths. The algorithm Quick-I intermingles the path computations in $G(b)$'s by expanding a path from v to w only if $BE(v,w) \geq b$.

The algorithm for mode III is obtained by replacing the lines 13 through 23 of Quick-II by following lines.

---

13. **if** $b_v \leq BE(v,w)$ **then**
14. $TE[w][b_v] \leftarrow \min\{TE[w][b_v], TE[v][b_v] + DE(v,w)\ \}$
15. **else**
16. $TE[w][B(v,w)] \leftarrow \{TE[w][B(v,w)], TE[v][b_v] - \sigma/b_v + DE(v,w) + \sigma/B(v,w)$
17. **if** $TE[w][BE(v,w)] < CT[w]$ **then**
18. $CT[w] \leftarrow TE[w][BE(v,w)]$;
19. $BE[w] \leftarrow BE(v,w)$;
20. return$(min_b\{TE[d][b]\})$

The algorithm for mode IIIa is obtained by replacing the line 14 of Quick-III by the following line. 14. $TE[w][B(v,w)] \leftarrow \min\{TE[w][B(v,w)], TE[v][b_v] + \sigma/B(v,w) + D(v,w)\}$;

The correctness proof and the time complexity of the corresponding algorithms,

namely Quick-i for $i = I, IIa, III, IIIa$, can be established with minor modifications to Theorem 2.1.

2.3 Concluding Remarks

We have presented five variations of the quickest path algorithms that reflect mechanisms such as circuit witching, Internet protocol and their combinations. We presented a basic algorithm variations of which compute the quickest paths in the first four modes with the time complexity, $O(m^2 + mnlogn)$, and for the last mode, Dijkstra's algorithm computes the quickest path, where the time complexity is $O(m + nlogn)$.

Future research directions include the computation of path-tables for all modes. For mode I, the path-table is of size $O(m)$ and can be easily computed [11, 37, 32], and such result can be very useful in the other modes. Another direction is the computation of multiple paths in various modes (as in [34, 47] for mode I).

Chapter 3

On Updating Algorithm on Quickest Paths

3.1 Introduction

The *quickest path problem* is to compute a path P with the minimum end-to-end delay for a message of size σ from source node s to destination node t. In this chapter we consider the quickest path problem under the transfer mode I corresponding to the circuit switching.

If the quickest paths are required for many values of σ, it is more efficient to compute the $path - table$ that specifies P_σ^I for each value of σ. To illustrate this point, we consider a network which consists exactly three disjoint paths P_1, P_2, and P_3 from s to t, where $D(P_1) < D(P_2) < D(P_3)$ and $B(P_1) < B(P_2) < B(P_3)$. Then the end-to-end delay is minimized by P_2 when σ lies in the intervals $[\frac{B(P_2)B(P_3)(D(P_2)-D(P_1))}{(B(P_2)-B(P_1))}, \frac{B(P_3)B(P_2)(D(P_3)-D(P_2))}{(B(P_3)-B(P_2))}]$. For example, a path P_1 (a, b, c, f) is

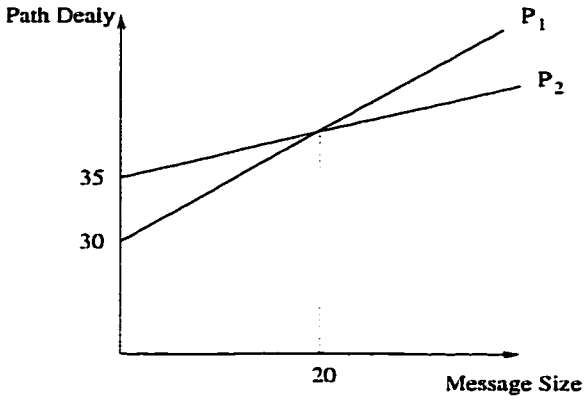

Figure 3.1: Line representation of the quickest paths. .

the shortest path from a to f in G_2 and a path P_2 (a, b, e, f) is the shortest path between a and f in G_4 in Figure 1.2 (on page 10 in chapter 1). The delay profile of P_i can be visualized as a line (see Figure 3.1), where slope is $1/B(P_i)$. For $D(P_1) < D(P_2)$ and $B(P_1) < B(P_2)$, lines are intersected at $R = R_I = \frac{(D(P_2)-D(P_1))B(P_1)B(P_2)}{B(P_2)-B(P_1)}$. Thus, if $\sigma \in (0, R_I]$ then the minimum end-to-end delay is achieved by P_1. Otherwise P_2 is used to achieve the minimum end-to-end delay. Since $R_I = \frac{(35-30)*2*4}{4-2} = 20$, we may obtain following path table;

Message Size	Path	Path Delay	Path Bandwidth
(0, 20]	(a, b, c, f)	30	2
[20, ∞]	(a, b, e, f)	35	4

For any message size σ, I-quickest path can be computed in $O(m^2 + mnlogn)$ time, and the path-table for mode I contains at most $O(m)$ intervals, each of which corresponds to a single quickest path and no two intervals have the same path. Two previous methods for computing the path-table for mode I are presented in [11, 32] and both algorithms have the time complexity of $O(m^2)$.

Furthermore, since change of link bandwidths that make path bandwidths changed

cause the path-table to be updated, it is critical to develop an algorithm to update the path-table.

In this chapter, we propose fast algorithms to update the path-table after a change in link or path bandwidth. Existing algorithms for computing the path-table or the quickest paths for all ranges of the size σ require two steps. First, let $B_1, B_2, ..., B_r$ be the number of distinct bandwidths with $B_1 < B_2 < ... < B_r$ in the network and clearly $r \leq m$, where m is the number of links in the networks. The first step constructs the shortest paths from source s to destination t in each of the graphs G_i, where G_i is a graph constructed by removing links with bandwidth less than B_i, $1 \leq i \leq r$. The best known algorithm for the first step requires $O(rm + rn \log n)$ time [32]. The second step is the computation of the path-table (presented in [32]), and requires $O(rq)$ time, where q is the number of intervals of message sizes in the path-table and $q \leq m$.

This chapter is organized as follows. Section 3.2 defines the terminologies used in this chapter. In Section 3.3, we present an algorithm to perform the second step, that is, to compute the path-table in $O(r)$ time. Section 3.4 discusses the dynamic algorithms that recompute the path-table in $O(r)$ time after a change in a link bandwidth or a path bandwidth (which involves changing bandwidths on several links in a path).

3.2 Preliminaries

Let $\mathcal{P}_N = \{P_1, P_2, \ldots, P_r\}$ be a set of paths such that P_i, $1 \leq i \leq r$ is the shortest path from s to t in the graph G_i defined earlier. Note that as mentioned earlier with

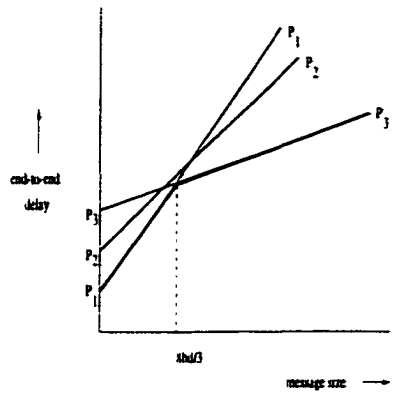

Figure 3.2: Lower envelope of $\{P_1, P_2, P_3\}$ contains only P_1 and P_2.

$B_1 < B_2 < ... < B_r$ we have $D(P_1) \leq D(P_2) \leq \ldots \leq D(P_r)$ and $B(P_1) \leq B(P_2) \leq \ldots \leq B(P_r)$. We will denote $B(G_i)$ to indicate the smallest link bandwidth in the graph G_i and $B(G_i) = B_i$. To facilitate our following algorithms, we define $P_i \in \mathcal{P}_N$, $1 \leq i \leq r$, to be *redundant* if it is not a quickest path for any σ otherwise, it is called *non-redundant*. Let $\mathcal{P}_N^*$ be the set of non-redundant paths with $\mathcal{P}_N^* \subseteq \mathcal{P}_N$. For $\mathcal{P}_N^* = \{P_1, P_2, \ldots, P_r\}$, $P_i, 1 < i < r$, is a quickest path for

$$\sigma \in \left[\frac{D(P_i) - D(P_{i-1})}{B(P_i) - B(P_{i-1})} B(P_i)B(P_{i-1}), \frac{D(P_{i+1}) - D(P_i)}{B(P_{i+1}) - B(P_i)} B(P_{i+1})B(P_{i1})\right] \qquad (3.1)$$

with appropriate boundary intervals specified for $i = 1$ and $i = r$.

Consider a network with three disjoint paths P_1, P_2 and P_3 with bandwidth and delay pairs given by (b, d), $(3b/2, 2d)$, and $(4b, 3d)$, respectively. We have $\mathcal{P}_N = \{P_1, P_2, P_3\}$, but P_1 is the quickest path for $\sigma \in [0, 8bd/3]$, and P_3 is the quickest path for $\sigma \geq 8bd/3$ as illustrated in Figure 3.2. Using P_2 - although it is not a quickest path for any σ, and the expression Eq (3.1) evaluates to $[3bd, 12bd/5]$ which is not an interval.

A link e on a path P is said to be *critical* if $B(P) = B(e)$ and it is *uniquely critical* if there exists no other link e' on P with $B(P) = B(e')$.

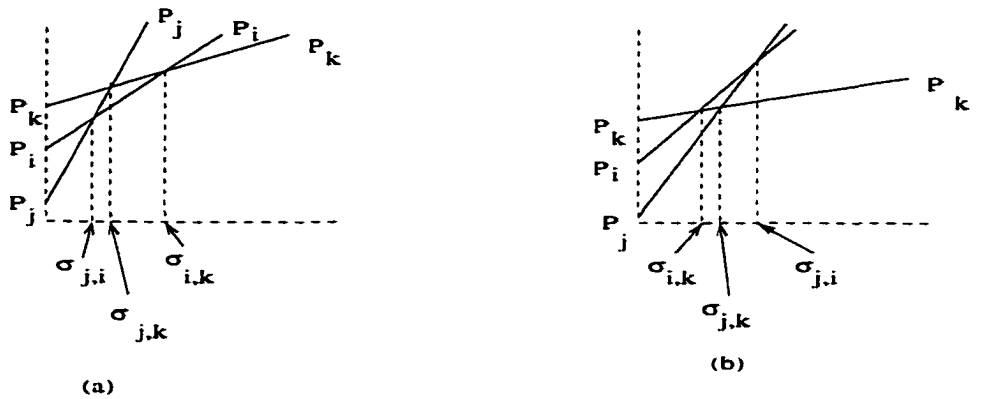

Figure 3.3: The condition $\sigma_{i,k} \leq \sigma_{j,k}$ is not satisfied in (a), and is satisfied in (b).

In the next section, we present an algorithm to compute the set of all non-redundant paths $\mathcal{P}_N^* \subseteq \mathcal{P}_N$ in $O(r)$ time given $\mathcal{P}_N$, and also compute the path-table with no additional time-complexity.

3.3 Path-Table Computation

For the path-table, the range of σ is partitioned into at most m intervals, each of which corresponds to a unique quickest path. It is known that no two distinct intervals have the same quickest path (see [32] for details). We note that for every $P \in \mathcal{P}_N^*$ the expression Eq (3.1) evaluates to a valid interval, but not necessarily for $P \in \mathcal{P}_N$.

We are given $\mathcal{P}_N = \{P_1, P_2, \ldots, P_r\}$, $r \leq m$. For simplicity, let $D_i = D(P_i)$, and $B_i = B(P_i)$, and $T_i(\sigma) = \frac{\sigma}{B_i} + D_i$. For $\mathcal{P}_N$ generated by algorithm of [32], we have $D_1 \leq D_2 \leq \ldots \leq D_r$ and $B_1 \leq B_2 \leq \ldots \leq B_r$. For distinct $P_i, P_j \in \mathcal{P}_N$, let $\sigma = \sigma_{i,j}$ denote the intersecting point satisfying the condition $\frac{\sigma_{i,j}}{B_i} + D_i = \frac{\sigma_{i,j}}{B_j} + D_j$ or equivalently $\sigma_{i,j} = \frac{D_j - D_i}{B_j - B_i} B_j B_i$.

We first note that P_l and P_r are not redundant. For $\sigma = 0$, the bandwidth plays no role, and the quickest path is the path with minimum total link-delay, namely P_l. For large value of σ (for example $\sigma = n^2(B_r + D_r)$), the delay plays no role, and the quickest path is P_r decided by the largest bandwidth alone [32]. Then we have the following properties illustrated in Figure 3.3.

Lemma 3.3.1 *For $|\mathcal{P}_N| > 2$, we have: (i) P_i is redundant if and only if for P_j and P_k such that $j < i < k$, we have $T_i(\sigma_{j,k}) \geq T_j(\sigma_{j,k}) = T_k(\sigma_{j,k})$; and (ii) P_i is redundant if and only if $\sigma_{i,k} \leq \sigma_{j,k}$ for $j < i < k$.*

Proof: Part (i): Consider the lower envelope of the lines corresponding to P_l, P_2, ..., P_r as shown in Figure 3.2. Since P_i is redundant, it lies above the lower envelope, and thus any two adjacent lines of the envelope serve as P_j and P_k satisfying $T_i(\sigma_{j,k}) \geq T_j(\sigma_{j,k}) = T_k(\sigma_{j,k})$ as shown in Figure 3.3(b). On the other hand, if P_j and P_k exist, it is easy to see that $T_j(\sigma) \leq T_i(\sigma)$ for $\sigma \in [0, \sigma_{j,k}]$ and $T_k(\sigma) \leq T_i(\sigma)$ for $\sigma \in [\sigma_{j,k}, \infty]$, which implies that P_i is redundant.

Part(ii): P_i is redundant if and only if $T_i(\sigma_{j,k}) \geq T_k(\sigma_{j,k})$ by Part (i). This condition is equivalent to $D_k - D_i \leq (1/B_i - 1/B_k)\sigma_{j,k}$. By substituting the expression $\sigma_{j,k} = \frac{D_j - D_i}{B_j - B_i} B_j B_i$, this condition in turn is equivalent to $\sigma_{i,k} \leq \sigma_{j,k}$. □

We now present our algorithm to compute $\mathcal{P}_N^*$ from $\mathcal{P}_N$, which utilizes a stack S of entities of the form $[\sigma, i, j]$ where σ is a suitable message size and i (j) is the index of path P_i (P_j).

---

algorithm COMPUTE-TABLE($\mathcal{P}_N$)

1. initialize stack S; $\sigma_L = 0$; Push($S, [\sigma_L, 0, 1]$); Push($S, [\sigma_{1,2}, 1, 2]$);

2. $k = 3$;

3. **while** $k \leq r$ **do**

4. $\quad [\sigma_{j,i}, j, i] =$ Top(S);

5. $\quad$ compute $\sigma_{i,k}$, $\sigma_{j,k}$;

6. $\quad$ **while** $((\sigma_{i,k} \leq \sigma_{j,k})$ **and** $(\sigma_{ji} \neq 0))$ **do**

7. $\quad\quad$ Pop(S); $[\sigma_{j,i}, j, i] =$ Top(S); compute σ_{jk};

8. $\quad$ Push($S, [\sigma_{j,k}, j, k]$); $k = k + 1$;

9. $\sigma_R = \infty$;

10. **while** not(Empty(S)) **do**

11. $\quad [\sigma_L, j, i] =$ Top(S);

12. $\quad$ make P_i quickest path entry for the interval $[\sigma_L, \sigma_R]$;

13. $\quad \sigma_R = \sigma_L$; Pop ($S$);

Our algorithm computes the path-table from left to right by identifying $\mathcal{P}_N^*$ from $\mathcal{P}_N$. Let $\mathcal{P}_N^* = \{P_{1^*}, P_{2^*}, \ldots, P_{q^*}\}$ such that $D(P_1^*) < D(P_1^*) < \ldots < D(P_q^*)$, and P_i^* is the quickest path for $\sigma \in [\sigma_{(i-1)^*,i^*}, \sigma_{i^*,(i+1)^*}]$. At the end of execution of Steps 1-8, we will show that stack S contains the entries $[\sigma_{q^*}, q^*]$, $[\sigma_{(q-1)^*}, (q-1)^*]$, ..., $[\sigma_{2^*}, 2^*]$, $[\sigma_{1^*}, 1^*]$, listed top-to-bottom.

First note that every path is pushed onto the stack S exactly once (Step 1, Step 8). The path P_i is removed from the stack in Step 7 only if $\sigma_{i,k} \leq \sigma_{j,k}$ which means that P_i is redundant (Lemma 3.3.1, Part (ii)). Now consider P_i is a redundant path and let $P_{j^*}, P_{k^*} \in \mathcal{P}_N^*$ such that j^* and k^* are the largest and smallest indices, respectively, such that $j^* < i < k^*$. The entry j^* is pushed onto stack and is never

removed from it, since P_{j^*} is non-redundant. When $k = k^*$ in the **while** loop of Steps 3-8, the condition $\sigma_{i,k^*} \leq \sigma_{j^*,k^*}$ is satisfied (since P_i is redundant). Hence, P_i will be removed from S if it is not removed earlier. Thus, only paths the remain on S are the non-redundant. By noting that the entries on S are of the form

$$\ldots [\sigma_{(i-1)^*,i^*}, i^*], [\sigma_{i^*,(i+1)^*}, (i+1)^*], \ldots$$

the intervals are correctly associated with the quickest paths, since P_{i^*} will be entered into path-table for the interval $[\sigma_{(i-1)^*,i^*}, \sigma_{i^*,(i+1)^*}]$.

For **while** loop of Steps 3-8, there are $O(r)$ iterations, with a single push operation and zero or more pop operations in each iteration. Since each path is pushed onto stack exactly once and only the paths on the stack are poped, the total complexity of this **while** loop is $O(r)$. The time complexity of **while** loop of Steps 10-13 is $O(r)$ since complexity of each iteration is $O(1)$. Thus, the time complexity of COMPUTE-TABLE is $O(r) \leq O(m)$.

3.4 Update Algorithms

In this section, we will present two algorithms to recompute the path-table after a change in bandwidth of a link and path. Note that the latter might involve changing the bandwidths of several links.

3.4.1 Change Link Bandwidth

Increasing the bandwidth of a single link can dramatically affect the precomputed path-table. It may require recomputation of the shortest paths $\mathcal{P}_N$. In this sub-

section, we present algorithms that adjusts the shortest path distance in $O(r)$ time without having to recompute the shortest paths.

Consider that the bandwidth B_i on a link e is increased to $B_i+\Delta$, for $\Delta > 0$. Now, we have $B(e) = B(P_i) = B_i$ and since G_i contains all links e' with $B(e') \geq B(e)$, the shortest path P_i remains unchanged. But the graphs G_j such that $B(G_i) < B(G_j) \leq B(G_{B_i+\Delta})$ might have to changed to include the new link e thus affecting certain shortest paths.

For the algorithm INCREASE-LINK-BANDWIDTH specified below, we apply the *forward Dijkstra's algorithm* and the *reverse Dijkstra's algorithm*. Let $T_{fwd}^{B_i}$ be a spanning tree induced by the *forward Dijkstra's algorithm* with G_{B_i}, and $T_{rvs}^{B_i}$ be induced by the *reverse Dijkstra's algorithm*. The tree $T_{fwd}^{B_i}$ represents the shortest paths from a given source to all other nodes and $T_{rvs}^{B_i}$ denotes the shortest paths from all other nodes to a given destination node. Suppose a new link $e = (u, v)$ is added to the network. To find the new shortest distance from a source to a given destination, we just need to compare $d[s,t]$ and $d[s,u] + D(e) + d[v,t]$ where $d[s,t]$ and $d[s,u]$ is obtained from $T_{fwd}^{B_i}$ and $d[v,t]$ is obtained from $T_{rvs}^{B_i}$.

It should be noted that the time to construct the reverse shortest path tree is same as the time for forward shortest path tree [1]. Hence the trees $T_{fwd}^{B_i}$ and $T_{rvs}^{B_i}$ for all graphs G_i with $1 \leq i \leq r$ can be constructed in $O(rm + rn \log n)$ time [32] as the first step in the algorithm. The computation of the new shortest distance given the forward and reverse trees after addition of an edge takes only $O(1)$ for each graph G_i.

Given $\mathcal{P}_N^*$ the following algorithm recomputes the path-table after the bandwidth B_i on a link $e = (u, v)$ is increased to $B_i + \Delta$.

algorithm INCREASE-LINK-BANDWIDTH($B_i + \Delta, u, v$)

Comment: Compute path-table after increasing link bandwidth

1. k = i;
2. **while** ($B_k[s,t] < B_i + \Delta$) **do**
3. **if** ($d[s,u] + D(u,v) + d[v,t]$) $< d[s,t]$
4. with $d[s,u]$ from $T^{B_i}_{fwd}$ and $d[v,t]$ from $T^{B_i}_{rvs}$
5. **then** $d[s,t] = d[s,u] + D(u,v) + d[v,t]$;
6. $B_k[s,t] = \min\{B_k[s,u], B_k[u,v], B_k[v,t]\}$
7. **else** $k = k + 1$
8. $\mathcal{P}^*_N = \mathcal{P}^*_N / \{P_B |\ B_k < B \leq B_i + \Delta$ if any$\}$
9. COMPUTE-TABLE ($\mathcal{P}^*_N$);

Theorem 3 *The time taken by the algorithm INCREASE-LINK-BANDWIDTH is* $O(r)$.

□.

Let B_i be the original bandwidth on a link e in the input network that was increased to $B_i + \Delta$. The decrease operation on the same link e by an amount τ is same as the increase operation on the link e by an amount $\Delta - \tau$. Let G^e represent the network with $B(e)$ set to 1. Let the $\mathcal{P}_N$ and $\mathcal{P}_N(e)$ be computed on G and G^e, respectively. The increase operation is performed on the paths $\mathcal{P}^*_N$. The paths $\mathcal{P}^*_N(e)$ are used to perform the decrease operation and this is done by increasing the

bandwidth on the link e by $B(e) - \tau - 1$, where τ is the amount of decrease. In order to perform the decrease operation on an arbitrary link in the graph, we need to compute the two sets of paths discussed above for every link and this will increase the time-complexity of the preprocessing step (step 1) by $O(m)$ in magnitude. After the preprocessing step increase and decrease operations can be completed in $O(r)$ time.

3.4.2 Increasing Path Bandwidth

Consider the case in which the bandwidth of the quickest path Q_i is to be increased from $B(Q_i)$ to $B(Q_i) + \Delta$. This operation could involve increasing bandwidths on links in the path Q_i by varying amounts. Performing such an increase could require recomputation of the path table. Let $\mathcal{P}_N^* = \{P_1, P_2, ..., P_N\}$ be the set of non-redundant paths with $B(P_1) < B(P_2) < ... < B(P_N)$. Increasing the path bandwidth of P_i from $B(P_i)$ to $B(P_i) + \Delta$ does not affect the non-redundant paths $P_1, ... P_{i-1}$, since the graphs $G_1, ..., G_{i-1}$ already contain the links whose bandwidths are to be increased. Similarly, the paths whose bandwidths are greater than $B(P_i) + \Delta$ are also not effected since their corresponding graphs anyway do not contain the links whose bandwidths are to be increased. We have to concern ourselves only with graphs with bandwidths on links greater than or equal to $B(P_i)$ and less than or equal to $B(P_i) + \Delta$.

The paths on graphs with bandwidth B such that $B(P_i) < B \leq B(P_i) + \Delta$ are redundant and can be removed from $\mathcal{P}_N^*$ to determine the new path-table. The reason behind this is as follows. Consider two paths P_i and P_j such that $D(P_i) < D(P_j)$ and $B(P_i) + \Delta \geq B(P_j)$. Now the path P_j is redundant by definition.

We can also increase the bandwidth of a deleted path, if we keep track of the deleted paths at every stage of deletion. For example, let $P_N^* = \{P_1, P_2, P_3, P_4, P_5\}$. If we increase $B(P_1)$ to $B(P_3)$, P_2 and P_3 will become redundant. Later, if we increase $B(P_2)$ to make P_2 *non-redundant*, we have to increase $B(P_2)$ to some bandwidth such that the intersection point $\sigma_{2,4} > \sigma_{1,4}$ by Lemma 3.1. This implies that to make P_2 non-redundant $B(P_2)$ we have $\frac{D_4-D_2}{B_4-B_2}B_4B_2 > \frac{D_4-D_1}{B_4-B_1}B_4B_1$. This gives rise to the fact that $B(P_2) > \frac{(D_4-D_1)B_4}{(B_4-B_1)(D_4-D_2)+(D_4-D_1)B_1}$. The decrease operation on path bandwidth can be performed very similar to the approach used for decrease of link bandwidth.

Since the number of deleted paths is no more than $O(r)$, the time it takes to recompute that path table after an increase in path bandwidth is $O(r)$.

3.5 Concluding Remarks

We introduced efficient algorithms to construct and update the path-table. The path-table that maps all intervals for σ to the corresponding quickest paths can be computed in $O(m^2 + mnlogn)$ time with n number of nodes and m number of links. Any change in the bandwidth of a single link or the path bandwidth can dramatically affect the computed path-table and may require to recompute the shortest path $\mathcal{P}_N$. Algorithms we proposed run in linear-time to construct and update the path-table.

Chapter 4

Reliability Problem on Quickest Paths

4.1 Introduction

The reliability problem of the quickest path deals with the transmission of a message of size σ from a source to a destination with both the minimum end-to-end delay and the reliability of the path over a network with bandwidth, delay, and probability of fault free on the links.

We consider a computer network represented by a graph $G = (V, A)$ with n nodes and m arcs or links. Each link $l = (i, j) \in A$ has a *bandwidth* $B(l) \geq 0$, *delay* $D(l) \geq 0$, and *probability of fault free* $0 \leq \pi(l) \leq 1$. The link delay includes the preparation and propagation time of the link. By pipelining, a message of σ units can be sent along the link l in $\sigma/B(l) + D(l)$ time. As in [48], we assume that only links are subject to failure but not nodes.

Consider a *simple path* P from i_0 to i_k given by $(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)$, where $(i_j, i_{j+1}) \in A$, for $j = 0, 1, \ldots, (k-1)$, and all $i_0, i_1, \ldots, i_k$ are distinct. Subsequently, a simple path is referred to simply as a path. The delay of this path P, denoted by $D[P]$, is given by $\sum_{j=0}^{k-1} D(l_j)$, where $l_j = (i_j, i_{j+1})$. The *bandwidth* of this path, denoded by $B(P)$, is given by $\min_{j=0}^{k-1} B(l_j)$. The reliability of this path, denoted by $R(P)$, is $\prod_{j=0}^{k-1} \pi(i_j, i_{j+1})$, and the *end-to-end delay* of the path P in transmitting a message of size σ is $\sigma/B(P) + D[P]$.

Following definitions are given by Xue[48].

Definition 4.1.1 *The path P from a source to a destination is called a most reliable path if $R(P)$ is the maximum among all paths P from a source to a destintion.*

Definition 4.1.2 *The path P is called a quickest most reliable path if P is the quickest path to transmit σ units among all most reliable paths from a source to a destination.*

Definition 4.1.3 *The path P is called a most reliable quickest path if P is the most reliable path to transmit σ units among all quickest paths from a source to a destination.*

In the field of the computer network, the *end-to-end delay* time of the routing path is very critical in wide area network. One of the well-known routing algorithm to support this Quality-of-Servie is the quickest path algorithm. However, since communation links may fail to transmit a message during the routing in the real network, its reliability is also very critical to guarantee to send a message from a source to a destination. The quickest path problem was first introduced by Moore

[30] and formulated by Chen and Chin [11]. Due to its importance in the computer network, the problem received more attention in the area of *operations research* and was extensively studied by Rosen *et al.* [37], Rao and Batsell [32], D.T. Lee [29], and Bang *et al.* [5]. Despite of the importance of the reliability on routing, only *end-to-end delay* is exhaustively studied. Recently, the reliability problem of the quickest path was first introduced by Xue [48] and two $O(rm+rn\log n)$ time algorithms were suggested to compute the quickest most reliable path and the most reliable quickest path from a source to a destination when a message size σ is given.

As indicated in [48], it is very straightforward to compute one-to-all most reliable quickest paths and quickest most reliable paths, and easy to see that all-to-all quickest most reliable paths and most reliable quickest paths can be computed in $O(rnm+rn^2\log n)$ time by applying algorithms by Xue [48], where n is the number of nodes, m is the number of links, and r is the number of distinct bandwidth in a network. Also, as will be shown in section 2, direct applying any all-to-all quickest path algorithm and Most Reliable Path Network (will be defined in next section) to compute all-to-all the quickest most reliable paths and the most reliable quickest paths may lead to incorrect results. Thus, as one can expect, it requires special cares to reduce the time complexity to $O(n^2m)$ and to achieve correct results.

In this chapter, we show that both all-to-all the quickest most reliable paths and the most reliable quickest paths can be computed in $O(n^2m)$ time.

The rest of the chapter is organized as follows. In section 4.2, we present details of algorithm to compute the quickest most reliable paths for all pairs of nodes in a given network. We discuss the algorithm for computing the most reliable quickest paths for all pairs of nodes in a given network in section 4.3.

4.2 The All-to-All Quickest Most Reliable Paths

Most Reliable Path Network(MRPN), originally called Shortest Path Network(SPN) [48], is a directed and acyclic subnetwork of G such that any path from source s to any node t in MRPN is the most reliable path, and it contains all the most reliable paths from s to any t. Since the most reliable path P is a path such that $R(P)$ is maximum among all paths from s to t, P can be found using a shortest path algorithm with new weight π' such that $\pi'(l) = \log(1/\pi(l))$. According to definition [48], the *quickest most reliable path* P is a path that is quickest among all $P \in \{P|\ P$ is a most reliable path from s to $t\ \}$. However, the number of most reliable paths may be exponential so that MRPN should be constructed to facilitate our work. Once MRPN is constructed by merging all the most reliable paths from s to every other node t, any quickest most reliable path from s to t can be computed by selecting a quickest path from s to t in MRPN.

In the case of one-to-all quickest most reliable paths by Xue [48], as explained. merging all the most reliable paths from s to any t creates the directed and acyclic subnetwork MRPN, and any path from s in MRPN is the most reliable path. Figure 1-(b) illustrates one-to-all MRPN by merging all the most reliable paths from a to other nodes, and shows the property such that any path from a is the most reliable path. However, since merging all the most reliable paths for all pairs of nodes in a network G can create some unexpected paths which are not most reliable, this property does not hold in all-to-all most reliable paths, . In Figure 2, a path $P(a,b)$ is the most reliable path from a to b and a path $P(b,c)$ is the most reliable path from b to c, but $P(a,b,c)$ is not the most reliable path.

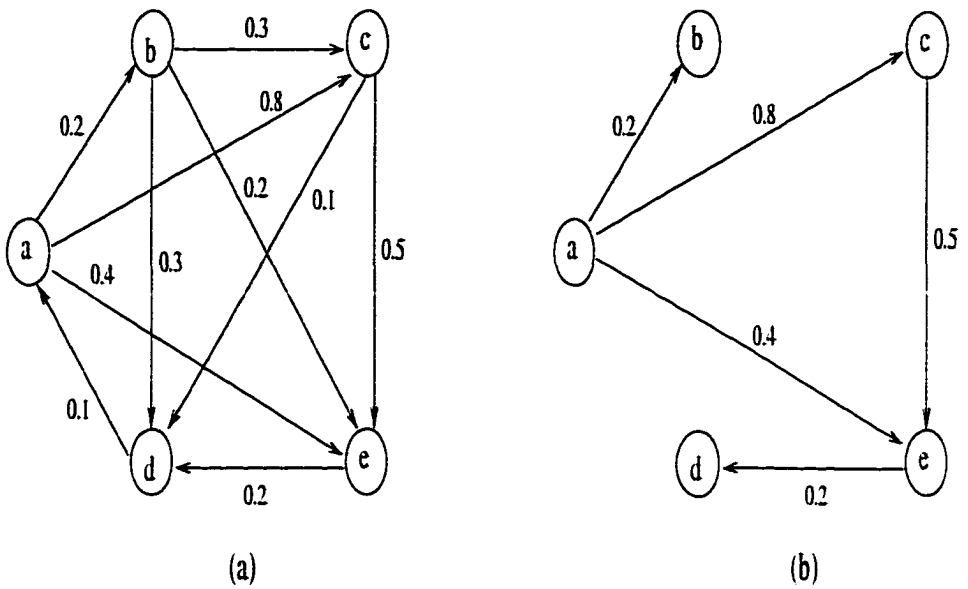

Figure 4.1: (a) Network G (b) Subnetwork after merging all the most reliable paths from a to every other node in G; Number in each link represents $\pi_I(e)$.

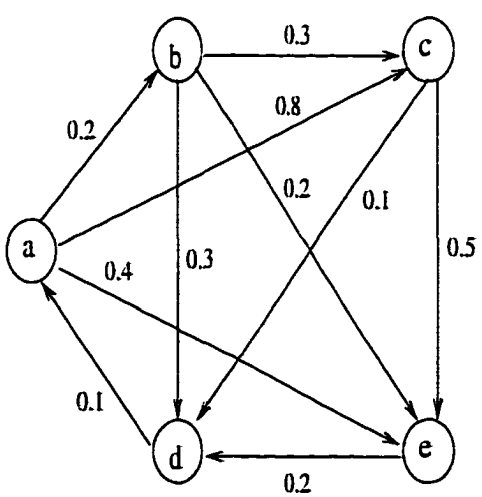

Figure 4.2: Subnetwork after merging all the most reliable paths for all pairs of nodes in G; Number in each link represents $\pi_I(e)$.

To solve the problem caused by unexpected paths, we may apply Dijkstra's shortest path algorithm to every node $i \in V$ as a source. Predecessor indices called $pred[i,j]$ are maintained in $n \times n$ two dimensional matrix *pred*, where each element $pred[i,j]$ is a list. Each index k in $pred[i,j]$ represents the last node k prior to node j in the shortest path P from i to j. Thus, if we backtrack along P with $pred[i,j]$, all the actual shortest path can be found. For example, suppose $P_1 = (i_0,i_1)(i_1,i_2)(i_2,i_4)(i_4,i_5)$ and $P_2 = (i_0,i_1)(i_1,i_2)(i_2,i_3)(i_3,i_4)(i_4,i_5)$ are two most reliable paths from i_0 to i_5. Then $pred[i_0,i_5] \rightarrow i_4$, $pred[i_0,i_4] \rightarrow i_2 \rightarrow i_3$, $pred[i_0,i_3] \rightarrow i_2$, $pred[i_0,i_2] \rightarrow i_1$, $pred[i_0,i_1] \rightarrow i_0$, and $pred[i_0,i_0] \rightarrow Null$. Thus, if we backtrack from $pred[i_0,i_5] \rightarrow i_4$, we can obtain both P_1 and P_2.

By implementing Dijkstra's shortest path, we can compute only one most reliable path from s to any particular $t \in V$. For this reason, we need to modify Dijkstra's shortest path algorithm to compute all the most reliable paths from s to any t. To do this, let $\Phi[i,j] = \sum_{j=0}^{k-1} \pi\prime(l_j)$, where $l_j = (i_j, i_{j+1})$. Let $\tau[i,j]$ be minimum among all $\Phi[i,j]$. We first compute $\tau[i,j]$ for all pairs of nodes (i,j) with $i \in V$ and $j \in V$. As shown in algorithm below, whenever we find $\Phi[i,j]$ such that $\Phi[i,j] = \tau[i,j]$, new predecessor of node j is added to the list of $pred[i,j]$. Since at least one of the most reliable paths from s to any t passes through a link $(pred[s,j] \rightarrow tail, j)$, we may construct a MRPN based on a root s by merging links $(pred[s,j] \rightarrow tail, j)$ with $j \in V$.

After all MRPNs based on each source $i \in V$ are constructed, links on each MRPN are labeled with a pair of nodes (a source, a destination) such that the link with (s,t) is on one of the most reliable paths from s to t. This labeling process can be easily done by backtracking each path from every $t \in V$ to s. To label, each link

(i,j) on G keeps $n \times n$ matrix called $Label_{ij}$. If a link (i,j) is on the most reliable path from s to t, then $Label_{ij}[s,t]$ is marked. By doing so, we may distinguish each link used by different most reliabl paths. Since there are at most n nodes in each MRPN, it takes $O(nm)$ to label all links in each MRPN. Thus, the time-complexity of this process for all MRPN is $O(n^2m)$.

To compute all-to-all quickest the most reliable paths, we merge all links in every MRPN to construct all-to-all MRPN, and apply all-to-all quickest path algorithm (AQP) [29] on all-to-all MRPN. As shown in Figure 2, since it may create unexpected paths to merge all the most reliable paths for all pairs of nodes (i,j) with $i \in V$ and $j \in V$, all-to-all MRPN does not hold the property such that any path in MRPN is the most reliable path. Thus, the quickest path computed by AQP may not be the most reliable path. For this reason, we need to modify AQP as follows; In AQP [29], if $D[u,i] + D(i,j) + D[j,v] < D[u,v]$ then the quickest path P_{uv} from u to v is updated with the path-bandwidth $B(i,j)$ which is $min\{B(u,i), B(i,j), andB(j,v)\}$. Although this updated path is the quickest path, it cannot be guaranteed that it is the most reliable path. However, if every link selected for a quickest path from s to t is from links which are used for the most reliable path, then it is the quickest the most reliable path. This implies that if the quickest path P_{st} is updated only when the link (i,j) is used for the most reliable path from s to t, we can compute the quickest the most reliable path without having problem caused by unexpected paths in all-to-all MRPN. If the above step in AQP is changed into **if** $(D[u,i] + D(i,j) + D[j,v] < D[u,v])$ and $(Label_{ij}[u,v]$ is marked) **then** update the quickest path, then we can find all-to-all quickest the most reliable paths correctly.

The algorithm to compute all-to-all MRPN is as follows;

Algorithm COMPUTE ALL-TO-ALL Quickest The Most Reliable Paths

Comment : Suppose $\tau[u, v]$ for all pairs of nodes (u, v) with $u \in V$ and $v \in V$ are precomputed.

Input : Network G

Output : All-to-All Most Reliable Path Network

1. $\Phi[u, v] = \infty$ for all pairs of nodes (u, v) with $u \in V$ and $v \in V$.
2. **for** $s = 1$ to n **do**
3. $\quad \Phi[s, s] = 0$
4. $\quad pred[s, v] =$ NULL for all $v \in V$
5. $\quad Q \leftarrow V[G]$
6. $\quad$ **while** $Q \neq \pi$ **do**
7. $\quad\quad u \leftarrow$ a node with minimum of $\Phi[s, v]$ from Q
8. $\quad\quad$ **for** each vertex $v \in Adj[u]$ **do**
9. $\quad\quad\quad$ **if** $\Phi[s, v] < \Phi[s, u] + \pi'[u, v]$ **then**
10. $\quad\quad\quad\quad \Phi[s, v] = \Phi[s, u] + \pi'[u, v]$
11. $\quad\quad\quad\quad$ **if** $\Phi[s, v] = \tau[s, v]$ **then**
12. $\quad\quad\quad\quad\quad pred[s, v] \rightarrow$ node $= u$
13. $\quad\quad\quad\quad$ **endif**
14. $\quad\quad\quad$ **endif**
15. $\quad\quad$ **endfor**
16. $\quad$ **endwhile**
17. **endfor**

18. **for** $s = 1$ to n **do**

19. **for** $t = 1$ to n **do**

20. backtrack every path from t to s.

21. **if** a link (i, j) is on a path from s to t **then**

22. mark $Label_{ij}[s, t]$.

23. **endif**

24. **endfor**

25. **endfor**

26. **for** $u = 1$ to n **do**

27. **for** $v = 1$ to n **do**

28. **while** $pred[u, v] \neq$ NULL **do**

29. add lind $(pred[u, v] \rightarrow tail, v)$ to AMRPN

30. **endwhile**

31. **endfor**

32. **endfor**

33. execute the modified AQP

34. **End of COMPUTE ALL-TO-ALL Quickest The Most Reliable Paths**

Theorem 4 *The all-to-all quickest most reliable paths to transmit a message can be computed in* $O(n^2m)$ *for all range of messige size* σ.

proof Clearly, Step 1 takes $O(n^2)$ time. From step2 to step17, all the quickest paths for all pairs of nodes in G are computed. Since Dijkstra's shortest path algorithm is invoked n times, it takes $O(nm + nnlogn)$ time. For labeling from step 18 to step 25, each path is backtracked n^2 times. Since each path may have at most $n-1$ links, it takes $O(n)$ time to backtrack any path. Thus, the time complexity for labeling of links is $O(n^3)$. To compute all-to-all quickest the most reliable paths, we apply modified AQP. The step 9 in modified AQP takes $O(1)$ time with the label of each link, the time complexity of the modified AQP is $O(n^2m)$. Therefore, algorithm, COMPUTE ALL-TO-ALL Quickest The Most Reliable Paths, takes $O(n^2m)$ running time □

4.3 The All To All Most Reliable Quickest Paths

Assume that $B_1 < B_2 < \ldots < B_r$ denote the distinct values of the bandwidth $B(l)$, $l \in A$ and G_{B_i} represents the subnetwork where bandwidth of all links in the subnetwork are greater than or equal to B_i with $1 \leq i \leq r$.

Since the most reliable quickest path from s to t is a path P such that $R(P)$ is maximum, where $P \in \{P \mid P$ is the quickest path from s to t to transmit σ units$\}$, we have to find all the quickest path from s to t with a given message of σ units. To develop an efficient algorithm to compute the most reliable quickest path, following observation in [32] leads to our efficient algorithm.

Observation 4.3.1 *Let $\mathcal{P} = \{P_{B_1}, P_{B_2}, \ldots, P_{B_r}\}$ denote the set of shortest path, where P_{B_i} is the shortest path in G_{B_i} with $1 \leq i \leq r$. Then we have $D[P_{B_1}] \leq D[P_{B_2}] \leq \ldots \leq D[P_{B_r}]$, and $B(P_{B_1}) < B(P_{B_2}) < \ldots < B(P_{B_r})$.*

Consider two path P_1 and P_2. If $D[P_1] = D[P_2]$ and $B(P_1) < B(P_2)$ then $\sigma/B(P_1) + D[P_1] < \sigma/B(P_2) + D[P_2]$. This means that P_1 is always slower than P_2 to sent σ units. Thus, by above observation [32], we can assume that $D[P_{B_1}] < D[P_{B_2}] < \ldots < D[P_{B_r}]$ and $B(P_{B_1}) < B(P_{B_2}) < \ldots < B(P_{B_r})$ by removing the equality. In addition, if $D[P_1] > D[P_2]$ and $B(P_1) = B(P_2)$ then P_1 never becomes the quickest path for any σ units. Thus, the quickest path P to transmit σ units is nothing but the shortest path in $G_{B(P)}$.

To derive the most reliable quickest paths, we need to compute the quickest paths for all pairs of nodes in G with any value of message size σ. In addition, we need to maintain following information by implementing AQP. Let $d_B[u, v]$ be the propagation time of the quickest path from u to v with the path bandwidth B. For each pairs of node (u, v), we define $d_{B_i}[u, v]$ with $B_1 < B_2 < \ldots < B_k$ and $d_{B_1} < d_{B_2} < \ldots < d_{B_k}$. Let $\mathcal{B}_{uv} = \{B_1, B_2, \ldots, B_k\}$ be a set of path bandwidths for quickest paths from u to v, where any bandwidth B_i is used for a particular value of σ with $1 \leq i \leq k$. Let $\tau_B[u, v] = \sum_{e \in P_B[u,v]} \pi l(e)$, where $P_B[u, v]$ be the quickest path from u to v computed by AQP. We define $Pl[u, v]$ which is a set of links used by an actual shortest path from u to v. By adding additional data structures to AQP, it can be easily done to maintain above information.

The algorithm COMPUTE-ALL-TO-ALL-MOST-RELIABLE-QUICKEST-PATH is as follows;

Algorithm COMPUTE-ALL-TO-ALL-MOST-RELIABLE-QUICKEST-PATH

Assumption1 : $D[u, v]$ is the delay time for a path from u to v.

Assumption2 : $D(u, v)$ is the delay time of a link (u, v).

Assumption3 : The stack arclist is sorted by non-increasing order with link-bandwidth.

Assumption4 : Top(arclist) is the link with highest link-bandwdith in current stack.

Assumption5 : $P\prime[u,v]$ is initially empty.

Input : Network G

Output : All-to-All Most reliable quickest path with respect to any vaule of σ

1. Compute all to all quickest paths by implementing the modified AQP with extra data structures
2. **while** arclist $\neq \emptyset$ **do**
3. (i,j) = pop(arclist)
4. **for** each pairs of nodes (u,v) **do**
5. **if** $D[u,i] + D(i,j) + D[j,v] < D[u,v]$ **then**
6. $D[u,v] = D[u,i] + D(i,j) + D[j,v]$
7. **endif**
8. $\Phi[u,v] = \Phi[u,i] + \pi\prime(i,j) + \Phi[j,v]$
9. **if** $B(i,j) \in \mathcal{B}_{uv}$ and $D[u,i] + D(i,j) + D[j,v] = d_{B(i,j)}[u,v]$ **then**
10. **if** $\Phi[u,v] < \tau[u,v]$ **then**
11. $P\prime[u,v] = P\prime[u,i] + (i,j) + P\prime[j,v]$
12. $\tau[u,v] = \Phi[u,v]$
13. replace $P_{B[i,j]}[u,v]$ by $P\prime[u,v]$
14. **endif**
15. **endif**
16. **if** $B(top(arclist)) < B(i,j)$ **then**
17. $P\prime[u,v] = P_{B[i,j]}[u,v]$

18. **endif**

19. **endfor**

20. **endwhile**

21. **COMPUTE-ALL-TO-ALL-MOST-RELIABLE-QUICKEST-PATH**

Correctness of algorithm as follows; In steps 5, 6, and 7, the delay of path from u to v is updated if needed. Since the quickest path with path bandwidth B is the shorest path in G_B, $d_{B(i,j)}[u,v]$ is the shortest delay for a path from u to v in $G_{B(i,j)}$. Thus, the most reliable quickest path from u to v with path bandwidth B is the most reliable path among all shortest paths from u to v in G_B. Step 9 checks if new computed path is the quickest path, and step 10 checks if new computed path is more reliable than the precomputed quickest path. If so, the quickest path is replaced by new computed path in step 11 ,step 12, and step 13. We can easily see that each $P\prime[u,v]$ computed in step 11 maintains the most reliable path among all shortest paths from u to v in each subnetwork G_{B_i}, with $B_i \in \mathcal{B}_{uv}$ through the algorithm. Since links in arclist are sorted in non-increasing order with link-bandwidth, subnetwork G_{B_i} is considered in order of $B_i = B_r, B_{r-1}, \ldots, B_1$ with $B_r > B_{r-1} > \ldots > B_1$. Since the quickest path is replaced whenever the more reliabe quickest path is found, all quickest paths after the algorithm is completed are the most reliable quickest paths.

Theorem 5 *All-to-all most reliable quickest paths can be computed in $O(n^2m)$ time.*

proof Obviously, each step which is nested in step 4 takes $O(1)$. It seems to take

more than $O(1)$ time to implement step 9. However, by indexing path-bandwidths $B[u, v] \in \mathcal{B}_{uv}$, the step 9 can be computed in $O(1)$ time, either. Thus, it takes $O(n^2)$ time to implement step 4 with its nested steps. Since while loop in step 2 is invoked for no more than m times and the time complexity of step 1 is $O(n^2m)$, the total time complexity of algorithm is $O(n^2m)$. □

4.4 Concluding Remarks

We studied the reliability problem of the quickest paths. The reliability problem of the quickest paths deals with the transmission of a message of size σ from a source to a destination with both the minimum end-to-end delay and the reliability of the path over a network with bandwidth, delay, and probability of fault free on the links. For any value of message size σ, we proposed $O(n^2m)$ time algorithms for all-to-all the quickest most reliable paths and the most reliable quickest paths, where n and m are the number of nodes and the number of edges or links in the network, respectively.

Chapter 5

On Multicasting With Minimum End-to-End Delays

5.1 Introduction

The minimum delay multicast problem that deals with the special case when $|MC| = 1$ was first formulated and solved with a time complexity $O(mn \log n + m^2)$ by Chen and Chin [11], in the area of operations research (also see [37, 32, 5]). This is called *quickest path problem* using circuit switch (Mode I). The multicast algorithms available in the computer networks literature solve our problem only in two important cases corresponding to the extreme values of message size r:

(a) When r is significantly small compared to the bandwidth of any link, the end-to-end delay is entirely controlled by the link-delays, and the Dijkstra's shortest path algorithm based only on link-delays [18, 7] minimizes the end-to-end delay.

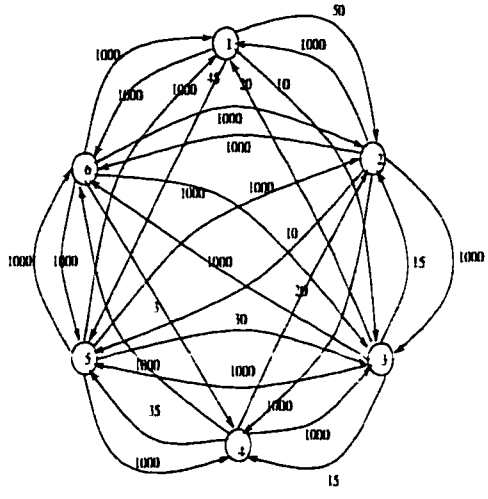

Figure 5.1: Illustrative example.

(b) If r is sufficiently large, the end-to-end delay is entirely controlled by the bandwidth, and thus a shortest-widest path [44, 3] yields the minimum end-to-end delay. Recall that a shortest-widest path [44] is the shortest among all paths from s to d with the largest bandwidth.

For a general value of r both bandwidth and delay constraints are active, and hence neither of the above algorithms is adequate as we will demonstrate using the following example network.

Consider the network shown in Figure 5.1, where the link delay and the bandwidth for each link are represented by the same number. Thus, in an overall sense, paths with higher delays also have higher bandwidths. Consider that $s = 5$ and $MC = \{1, 2, 3, 4\}$. As r is varied from 10 to 10^6, the corresponding multicast-trees with minimum end-to-end delays corresponding to Mode I migrate from low bandwidth to higher bandwidth paths as shown in Figure 5.2. For small message sizes, for example $r = 10$, the multicast-tree consists of exclusively shortest delay paths based only on link-delays. For example, the shortest delay path from 5 to 2 is $5-3-4-2$

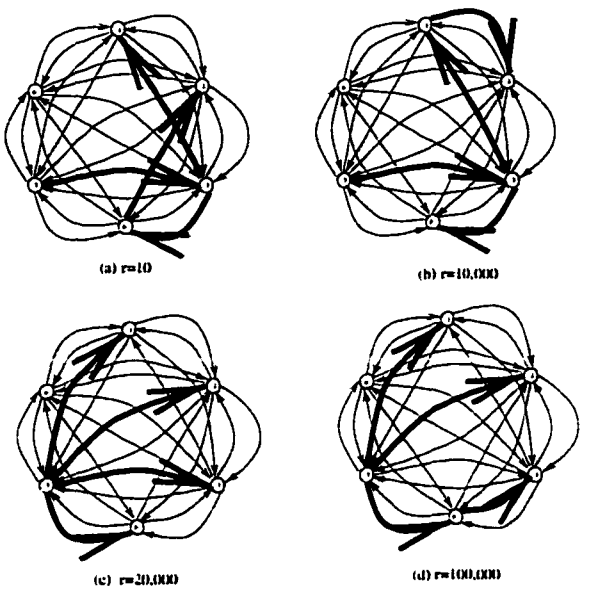

Figure 5.2: Migration of minimum end-to-end delay multicast-trees with Mode I.

with a delay of 65 and bandwidth of 15 (Figure 5.2(a)). As r is increase to large values, for example $r = 100,000$, the multicast-tree consists of shortest-widest paths only; the shortest-widest path from 5 to 2 is $5 - 2$ with a delay of 1000 and bandwidth of 1000 (Figure 5.2(c)). In between the two extreme values of r, note that multicast-tree takes two different forms for $r = 10,000$ and $r = 20,000$. Thus, this example illustrates that the message sizes can significantly effect the profile of the multicast-trees.

We are given the computer network $G = (V, E)$, and the delays $D(e)$ and bandwidths $B(e)$, for all $e \in E$. The task is to compute *a multicast tree* such that the end-to-end delay time of the tree is minimized for each value of r, where the end-to-end delay time of a tree is the maximum end-to-end delay time among all end-to-end delay time from s to $d \in MC$. The network under consideration in this paper can be modified to have advantage of the existing algorithms for multicasting by assigning

a weight of $r/B(e) + D(e)$ to each link in the network. But as we will show in the section 5 of this paper, such algorithms perform very poorly for various message sizes r and networks with bandwidth and delay-time associated with each link.

In this chapter, we have investigated for the first time the performance of various heuristic algorithms for multicast tree construction taking into account several multicast switch architectures. These multicasting switch architectures include the store-and-forward switch - wherein the entire packet is stored in the switch before it is sent on the outgoing link, the pipeline switch - wherein the incoming bit stream is relayed to an outgoing link by using pipelining techniques, and the hybrid switch - which behaves like a store-and-forward switch if the incoming link bandwidth is greater than the outgoing link bandwidth, otherwise it behaves like a pipeline switch. In section 2, we will discuss these switch architectures in detail. Indeed, the pipeline switch is more powerful in comparison with the other switches. Guo and Chang [20] present an excellent survey on multicast switches.

Our contributions in this research are summarized in the following:

- We have evaluated existing heuristics and new proposed heuristics to compute a multicast tree that takes into account message size, and delay time and bandwidth of each link.

- We have investigated various multicast tree construction heuristics that takes into account different path finding algorithms that includes quickest path, widest-shortest path, and shortest path.

- We have evaluated our proposed heuristics to take into account three multicast switch architectures that span the breadth of all multicast switch architectures.

- We have performed extensive simulation studies taking into account various network generation models to evaluate the proposed multicast heuristics. The network models that we have considered include Waxman I [45], Waxman II [45], Locality [49], and Transit-Stub networks [49]. Note that as indicated in [49], the Transit-Stub network closely reflect the current Internet topology. We believe that our simulation for evaluating multicasting heuristics is the first of a kind that involves several network generation models.

- Our simulation also takes into account the distributions of message sizes and bandwidths on links.

This chapter is organized as follows. In section 5.2, we present details of various multicast switch architectures we have considered. Several heuristic algorithms for computing the multicast tree and its minimum end-to-end delay for the different switch architectures are also presented in section 5.3. In section 5.4, we present a novel algorithm to allocate bandwidth on links in such a way that the minimum end-to-end delay can be achieved for the case of hybrid switches. In section 5.5, we discuss the various networks models we have used in our simulations and interpret the results of our extensive simulation. Summary and further research are presented in section 5.6.

5.2 Paths and Multicast Switches

We have considered three path finding algorithms that are used to construct the multicast trees. The first of these is the quickest path algorithm which is executed on the original network for a given message size r. The second algorithm is the

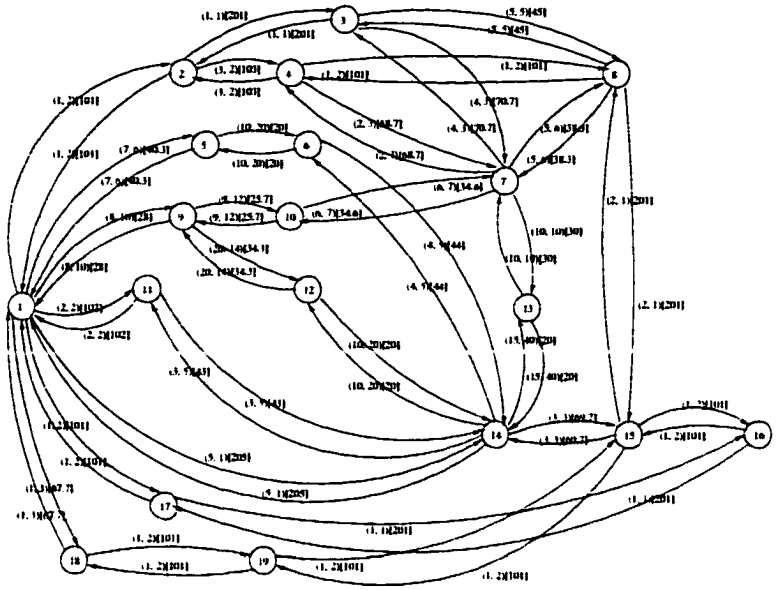

Figure 5.3: The original network and the number in the square brackets on each link e is $200/B(e) + D(e)$. The label (d, b) on each link represent the delay d and bandwidth b of each link.

shortest path algorithm that constructs the path on the network with the weight $w(e)$ of every link e set to $r/B(e) + D(e)$. The third path finding algorithm is the widest-shortest path algorithm that works on the original network and is independent of the message size r. The shortest path algorithm can be easily modified to obtain the widest path (the path with the largest bandwidth) among all shortest paths between a pair of nodes. Consider the network in Figure 5.3 the numbers in the square brackets are obtained by assigning a weight $w(e) = r/B(e) + D(e)$ with $r = 200$ for every link in the network.

Once a path is constructed using any of the path finding algorithms the end-to-end delay is dependent not only on the bandwidths and link delays on the links of the path, but also on the switch architecture at the nodes in the path. For this purpose, we consider three different multicast switch architectures. A *pipeline* switch sends

the bits from the incoming link to the appropriate outgoing links without buffering. The *end-to-end delay* of a path P from source s to a destination d in transmitting a message of size r is $r/B(P) + D(P)$ using the pipeline switches at nodes of the path. The *store-and-forward* switch is the most simplest of all switches. Every incoming packet is stored in the buffer before it is sent out on all the appropriate outgoing links. The end-to-end delay for a path P consisting of store-and-forward switches is $\sum_e (r/B(e) + D(e))$, where e is a link on the path P. In the case of the *hybrid* switch, if the incoming link bandwidth is larger than the bandwidths of outgoing links then the switch behave like a store-and-forward switch, otherwise it works like a pipeline switch. The end-to-end delay for a path P which contains hybrid switches is $r/B(P) + D(P) + T$, where $T = \sum_e r/B(e(i,j))$, where e is a link in the path P connecting node i and node j and the incoming bandwidth to node i is greater than the outgoing bandwidth to node j from node i.

The following table gives the end-to-end delay from node 1 to node 13 in the network in Figure 5.3 using three path algorithms.

Algorithm and Path	Multicast Switch Architecture		
	Pipeline	Store-and-Forward	Hybrid
Quickest Path 1–9–10–7-13	61.6	118.9	81.6
Shortest Path 1–9–12–14–13	73	102.3	73
Widest-Shortest Path 1–2–4–7–13	116	302.7	116

Table 5.1: This table displays the end-to-end delay from node 1 to node 13 and the path in

the network given in Figure 5.3 for a message size of $r = 200$ for various switch architectures and path algorithms.

From the above table it is clear that the given pipeline switches the quickest path algorithm outperforms all the others in terms of end-to-end delay. The shortest path on the link modified network (using the weight given in square brackets in Figure 5.3) is clearly the best choice for the case of store-and-forward switches. For a network with hybrid switches the quickest path algorithm is the best choice for the path finding algorithm. It will be established in Section 5.4 that a weaker switch such as the hybrid switch can match the performance of the pipeline switch if bandwidths of the links on the quickest paths can be lowered suitably.

5.3 Construction of Multicast Trees

Algorithms for the construction of multicast trees follow two types: shortest path and steiner tree based. The shortest path based algorithms try to minimize the end-to-end delay. If the objective is to build a multicast tree for a given source to a set of destinations taking into account only link delays then a pruned single source shortest path tree would suffice. Such a tree can be constructed in polynomial time [15]. The steiner tree construction problem is intractable since it tries to construct a tree for a given set of nodes (source and destinations) such that the sum of the costs associated with the links are a minimum [19]. Heuristics for steiner tree based algorithms use minimum spanning tree algorithms due to Kruskal's [15] and Prim's [15]. Salama [38] presents an excellent survey of various multicasting problems and algorithms.

In this section, we present two sets of algorithms; one for shortest path based and the other one based on minimum spanning tree construction (steiner heuristics). Our construction algorithms will be independent of the switching architecture but will be dependent on the message size r.

5.3.1 Shortest Path Based Algorithms

Shortest path based heuristics work by merging optimal paths from each of the destinations to source. The network induced by merging these paths will form a tree if the criteria for optimal path selection is based on delay or cost. In this subsection, we present three algorithms that choose path based on different criteria and mechanisms to handle the subnetwork formed by paths that are merged.

The *merging quickest path* algorithm determines the quickest path from the source to each of the destinations and the quickest paths induce a subnetwork. This subnetwork need not be a tree as illustrated in Figure 5.4. A depth first search tree starting from the source on this subnetwork is performed to obtain a multicast tree. Such a tree need not be optimal for pipeline switches and finding the optimal tree appears to be computationally intractable.

The second algorithm called *widest-shortest path* determines the widest-shortest path from the source to each of the destinations and merges these paths to form the multicasting tree. From Table 5.1 we can conclude that the widest-shortest path algorithm is not optimal for store-and-forward switches.

The third algorithm is based on shortest path with link weights (see Figure 5.3) and we can use Dijkstra's algorithm [15] to construct the tree. The tree that results after merging shortest paths is an optimal tree assuming the switches are store-and-

forward switches.

Merging Quickest Path

We now present the algorithm Min-Path of [37] originally proposed to compute a minimum end-to-end delay paths from s to all $d \in V$, for any given value of r. Let $B_1, B_2, \ldots, B_c$ denote the distinct values of the bandwidths $B(e)$, $e \in E$. Let $G(b) = (V, E(b))$ denote the subnetwork where $e \in E(b)$ if and only if $B(e) \geq b$. Let a $s - d$ *path* in $G(b)$ denote the shortest delay path based *only* on the link-delays. The algorithm of [37] is as follows:

---

algorithm Min-Path(r)

1. for $j = 1, 2, \ldots, c$, and $d \in MC$ compute $s - d$ path P_j^d in $G(B_j)$;
2. for each $d \in MC$
3. compute index d_k which minimizes $\{r/B(P_j^d) + D(P_j^d) | j = 1, 2, \ldots, c\}$;
4. return $P_{d_k}^d$;

Step 1 of this algorithm is executed by c invocations of the Dijkstra's shortest path algorithm with a total time complexity $O(cm + cn \log n)$ using Fibonacci heaps [15]. The cost of steps 2-4 is $O(bc)$, where only pointers to the paths are returned in line 4. Then, the multicast-tree composed of the *constituent paths* $P_{d_1}^1, P_{d_2}^2, \ldots, P_{d_c}^c$ can be obtained in $O(m + bn)$ time as follows: construct a network $G^1 = (V, EP)$ such that $e \in EP$ if and only if e is contained in one of the constituent paths, and obtain a depth-first tree rooted at s. Let us define the *bandwidth* of a subtree of

multicast-tree to be that of a constituent path with the largest bandwidth from the root of the subtree to one its nodes from MC. At each node v of the multicast-tree the link-disjoint subtrees rooted at v and their bandwidths can be computed in $O(n+m)$ time.

For messages of different sizes, this algorithm may have to be executed repeatedly since the multicast tree might be different. By precomputing the multicast-table as described in Section 5.4, the multicast tree can be obtained with significantly lower on-line computational cost.

The algorithm for the computation of multicast-table for the algorithm called Merging-Quickest-Path is obtained by adapting the algorithm of [32], which was designed to compute a path-table, i.e. for the special case $|MC| = 1$. The outline of our algorithm is as follows:

---

algorithm Compute-Multicast-Table

1. for each $d \in MC$ do
2. Compute-Path-Table($d, P_L^0, r_L^0, P_R^0, r_R^0$);
3. combine path-tables to form multicast-table;

For each destination $d \in MC$, we compute the path-table using the algorithm Compute-Path-Table in line 2, and merge the path-tables to obtain the multicast-table in line 3. The merging of the path-tables is performed by first sorting and merging the lists of values of r that denote the end points of the intervals of the individual path-tables. Then, for each interval in the merged list, corresponding

paths from individual path-tables are retrieved. Each path-table has no more than m entries [32] and thus the multicast-table has no more than bm entries. Once the constituent paths of each interval of r in the merged list are computed, the corresponding multicast-trees can be computed using the technique described in Section 5.2.

For completeness, we now briefly explain the algorithm Compute-Path-Table of [32] used in step 2 above. At any invocation of the algorithm Compute-Path-Table, paths P_L and P_R are known to achieve the minimum end-to-end delays for the message sizes r_L and r_R, respectively. The algorithm tests if P_L and P_R achieve the minimum end-to-end delay for the entire interval $[r_L, r_R]$ by first computing the message size r_I corresponding to their intersection (line 1). Then a path P_I with the minimum end-to-end delay for the message size r_I is computed (line 2). If P_I can be replaced by one of P_L or P_R under the condition $[D(P_I) = D(P_L)$ and $B(P_I) = B(P_L)]$ or $[B(P_I) = B(P_R)$ and $B(P_I) = B(P_R)]$, then P_L and P_R are entered into the path-table for this interval (line 4). If not, the algorithm is recursively called for each of new intervals $[r_L, r_I]$ and $[r_I, r_R]$ in lines 6 and 7, respectively. Initially, $P_L = P_L^0$ is a shortest delay path, $P_R = P_R^0$ is a shortest-widest path, $r_L^0 = 0$, and $r_R^0 = \frac{nD_{\max}B_{\max}^1 B_{\max}^2}{B_{\max}^1 - B_{\max}^2}$, where $D_{\max} = \max_{e \in E} D(e)$, $B_{\max}^1$ is the bandwidth of the shortest-widest path, and $B_{\max}^2$ is the bandwidth of the shortest-widest path in the network G^2 obtained by removing all links with bandwidth at least B_{max}^1.

---

algorithm Compute-Paths(d, P_L, r_L, P_R, r_R)

1. compute intersection size r_I;

2. $P_I \leftarrow$ minimum end-to-end delay $s-d$ path for message size r_I;

3. **if** $[D(P_I) = D(P_L)$ and $B(P_I) = B(P_L)]$ or $[B(P_I) = B(P_R)$ and $B(P_I) = B(P_R)]$ **then**

4. Path$[r_L, r_I] \leftarrow P_L$; Path$[r_I, r_R] \leftarrow P_R$;

5. **else**

6. Compute-Table(P_L, r_L, P_I, r_I);

7. Compute-Table(P_I, r_I, P_R, r_R);

We now estimate the complexity of the algorithm Compute-Multicast-Table. The complexity of computation of path-tables is $O(cqb + cmb + cbn \log n)$, where q is an upperbound on the number of entries of the path-table for any $d \in MC$ [32]. The merging of the path-tables can be achieved in $O(bm \log m)$ using available list merging techniques. Thus the total time complexity is $O(cqb + cmb + cbn \log n + bm \log m)$ which is upperbounded by $O(m^2 n + mn^2 \log n)$.

Merging Widest-Shortest Paths (WSP)

Widest-Shortest Path is the shortest delay path from s to d with the largest bandwidth among all shortest path from s to d. The path can be found using labeling Dijkstra's shortest path algorithm where weights of link are the delay time and bandwidth on the link.

The basic idea of WSP is first to compute one to all widest-shortest path tree rooted at s using labeling Dijkstra's shortest path algorithm. Once the widest-shortest path tree is constructed, it is pruned to remove leaf nodes of degree 1 until

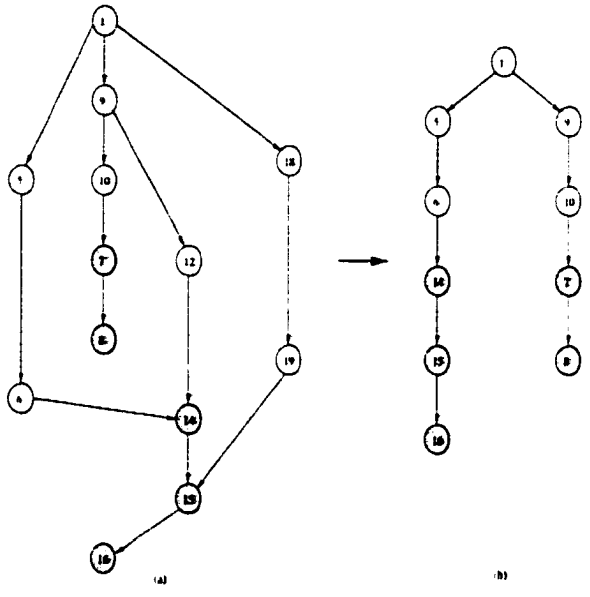

Figure 5.4: The subnetwork formed by merging quickest paths shown in (a) and the depth first search of the subnetwork is shown in (b). The source node is node 1 and the destination nodes are the labels of shaded circles. The end-to-end delay of the tree is shown in Table 5.2.

either a destination node or source node is reached. The widest-shortest path based algorithm to compute the multicast tree work is as follows:

Assume that Q is the priority queue with Fibonacci heap and $d[v]$ denotes the shortest distance from source s to any node v. The algorithm also maintains a set S that contains vertices whose final shortest path from s have already been determined. To determine the widest-shortest path, we define an array B to store the path bandwidth from the source to each node $v \in V$. $B[v]$ denotes the path bandwidth from s to d. Each element of tree$[v]$ represents the parent of node v in the widest-shortest path tree. The minimum key in the algorithm is the minimum of $d[v]$ for all $v \in Q$.

---

algorithm Merging-Widest-Shortest-Path

1. $d[v] = \infty$ and tree$[v] \leftarrow$ **NIL** for each node $v \in V$
2. $d[s] \leftarrow 0$
3. $S \leftarrow \emptyset$
4. $Q \leftarrow V$
5. **for** $Q \neq \emptyset$ **do**
6. $\quad u \leftarrow$ a node with minimum key from Q
7. $\quad S \leftarrow S \cup \{u\}$
8. $\quad$ **for** each vertex $v \in Adj[v]$ **do**
9. $\quad\quad$ **if** $d[v] > d[u] + D(u, v)$ **then**
10. $\quad\quad\quad d[v] \leftarrow d[u] + D(u, v)$
11. $\quad\quad\quad B[v] \leftarrow \min(B[u], B[u, v])$
12. $\quad\quad\quad$ tree$[v] \leftarrow u$

13. **if** $(d[v] = d[u] + D(u,v))$ and

14. $B[v] < \min(B[u], B[u,v])$ **then**

15. $\text{tree}[v] \leftarrow u$

16. prune the tree

Since we need to find the widest-shortest paths from s to each destination $d \in MC$, path bandwidth should be maintained to compare with that of new found shortest-path with the same cost. Step 11 of this algorithm is to maintain the path bandwidth of shortest path found so far. In step 14, if new shortest path with the same cost has a higher path bandwidth, then tree is updated so that the tree always keeps the widest-shortest path in the tree. In step 16, we perform a backtracking operation starting from each destination $d \in MC$ and remove nodes that are not in the paths from each destination $d \in MC$ to the source (root) of the tree.

The time complexity of step 16 is no more than $O(n)$ where $q = |MC|$, and step 1-15 runs in $O(m + n \log n)$ using Fibonacci heap. Thus, total time complexity of this algorithm runs in $O(qn+m+n \log n)$. In Figure 5.5-(a) the pruned single source shortest path tree based on just link delays is given and Figure 5.5-(b) presents the pruned widest-shortest multicasting tree.

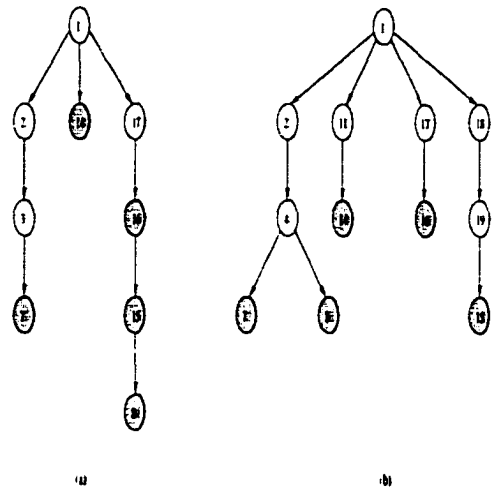

Figure 5.5: The multicast tree based on link delays is shown in (a) and widest-shortest multicast tree in shown in (b). The source node is node 1 and the destination nodes are the labels of shaded circles. The end-to-end delays are shown in Table 5.2.

Merging Shortest-Path With Modified LINK Weight (MLW)

Given a message with size of r and a network $G = (V, E)$ where each $e \in E$ is associated with $D(e)$ and $B(e)$, we define new link weight $w(e)$ as a weight of an link where $w(e) = r/B(e) + D(e)$. For the network in Figure 5.3, the $w(e)$ for every link e is given in square brackets. The algorithm MLW uses $w(e)$ to determine the single source shortest path tree and after pruning this tree will be the multicasting tree. The minimum key represents the minimum of $w(v)$ for all $v \in Q$.

The basic step of the algorithm can be represented as follows:

---

algorithm Merging-Modified-Link-Weight

1. $w[v] = \infty$ and tree$[v] \leftarrow$ **NIL** for each node $v \in V$
2. $w[s] \leftarrow 0$
3. $S \leftarrow \emptyset$

4. $Q \leftarrow V$

5. **for** $Q \neq \emptyset$ **do**

6. $u \leftarrow$ a node with minimum key from Q

7. $S \leftarrow S \cup \{u\}$

8. **for** each vertex $v \in Adj[v]$ **do**

9. **if** $w(v) > w(u) + r/B(u,v) + D(u,v)$ **then**

10. $w(v) \leftarrow w(u) + r/B(u,v) + D(u,v)$

12. tree$[v] \leftarrow u$

13. prune the tree

This algorithm has the same time-complexity as the Merging-Widest-Shortest Path algorithm.

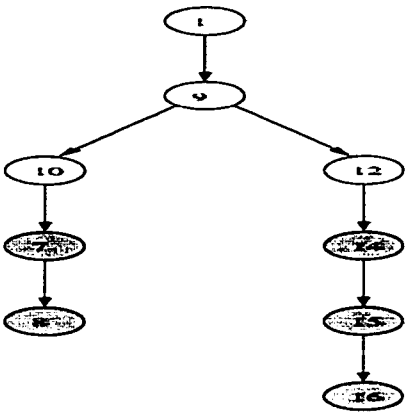

Figure 5.6: The multicast tree based on modified link weights is shown. The source node is node 1 and the destination nodes are the labels of shaded circles. The end-to-end delays are shown in Table 5.2.

Comment on Shortest Path Tree Based Algorithms

Our results for the end-to-end delay of the multicast tree is consistent with the results for one-to-one path end-to-end delay presented in Table 5.1.

Algorithm and Tree	Multicast Switch Architecture		
	Pipeline	Store-and-Forward	Hybrid
Merging Quickest Path	125	275	265
Merging Modified Link Weight	142	253	228.7
Merging Widest-Shortest Path	202	305	302

Table 5.2: This table displays the end-to-end delay of each of the multicast tree that results from the algorithms based on shortest path computation. The message size chosen is 200, source node is 1 and destination nodes are $\{7, 8, 14, 15, 16\}$.

5.3.2 Minimum Spanning Tree Based Algorithms

In this subsection, we present two minimum steiner tree based algorithms. Heuristics based on minimum steiner tree have been developed for minimizing the sum of the cost of links that form the tree [38]. The algorithms that we present use the steiner based heuristics to minimize the end-to-end delay of multicast trees.

The *Best-First Algorithm* which is based on Prim's minimum spanning tree algorithm starts with a multicast tree that contains the source node only. Then it adds each destination $d \in MC$, one at a time, to the existing multicast tree via the least cost path to any node which is already in the tree. The *TM-Heuristic* [41] is a Best-First algorithm where the least cost path is constructed based on a cost associated with the links. We propose three least cost path selection criteria which includes quickest path, widest-shortest path, and shortest path based on modified link weight as part of our Best-First algorithm. As observed earlier, if the path selection is based on quickest paths, then the resulting subnetwork need not be a tree and we need perform a depth first search to obtain a multicast tree.

The *Grow-Tree Algorithm* is based on both Kruskal's and Prim's minimum spanning tree algorithm. Note that Rayward-Smith [35] presents a Kruskal's algorithm based heuristic to construct the minimum spanning tree that takes into account costs associated with links. Grow-Tree algorithm also starts with the multicast tree with a given source node only. However, unlike the Best-First algorithm, it adds an link (u, v) but not a path to the existing multicast tree with $u \in V'$ and $v \notin V'$, where V' is the set of nodes in a current multicast tree. Let $P(x \sim y)$ be a path from node x to node y in a multicast tree. To add an link (u, v) to the existing tree with $u \in V'$ and $v \notin V'$, the end-to-end delay time of path $P(s \sim u, v)$ is computed for all

neighbors v of node u. If the end-to-end delay time of a path $P(s \sim u, v)$ including the link (u, v) is the minimum among all such paths, then the link (u, v) is added to the multicast tree. In the following subsections we will present the Best-First and Grow-Tree algorithms in detail.

Best-First Algorithm

In the case of quickest path based algorithm, Best-First algorithm may construct the subnetwork which is not the tree as in the case of merging quickest path approach. First step is to implement the Best-First algorithm, and the second step is to construct the depth-first tree from the subnetwork obtained by the first step.

Let Best-First-Subnet be a subnetwork of the given network G with $G = (V, E)$, and initially, it contains the source node only. Let Time(v, d) denote a end-to-end delay time from node v to d. We will assume that the variable min-Time in the algorithm Best-First below is initially set to infinity.

The algorithm Best-First is presented below:

---

algorithm Best-First

1. **while** $MC \neq \emptyset$ **do**
2. min-Time $= \infty$
3. **for** each node $v \in$ Best-First-Subnet **do**
4. **for** each destination $d \in MC$ **do**
5. **if** Time(v, d) < min-Time **then**
6. min-Time $\leftarrow$ Time(v, d)
7. selected-Node $\leftarrow v$

8. selected-Destination $\leftarrow d$

9. add the path from v to d to Best-First-Subnet

10. $MC = MC - \{d\}$

11. Compute Depth-First tree with Best-First-subnet if based on the quickest path

12. Otherwise, skip step 10.

13. return(tree)

To execute the step 4, we need to compute the end-to-end delay time Time(v, d). If Best-First algorithm is based on the quickest path, Time(v, d) is the end-to-end delay time of the quickest path from v to d. In the case of the widest-shortest path based multicast tree, Time(v, d) is the end-to-end delay time of the widest-shortest path from v to d. Similarly, Time(v, d) is the shortest path based on modified link weight for Link Weight based multicast tree construction.

The cost of While loop in step 1 is $O(|MC|)$, and step 2 is executed for current number of nodes in Best-First-subnet. To compute the Time(v, d) in step 4, it takes $O(cm + cn \log n)$ for the quickest path, and $O(m + n \log n)$ for both the widest-shortest and the shortest path with modified link weight. Since the computation of Time(v, d) is expensive, we need the preprocessing step to compute quickest paths, widest-shortest path, and shortest path with modified link weight for $P(v, d)$ for all pairs of nodes, v and d, where $v \in V$ and $d \in MC$. Then any Time(v, d) can be computed in $O(1)$ time. Thus, with the preprocessing step, total time complexity to build multicast tree is $O(|MC|n'|MC|')$ where n' is the current number of nodes in Best-First-Subnet and $|MC|'$ is the current number of destinations in MC.

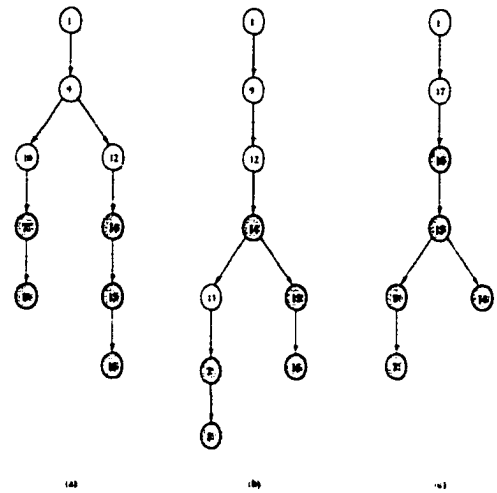

Figure 5.7: The Best-First multicast tree based on quickest path is shown in (a) and tree based on modified link weight is shown in (b), widest-shortest based tree is shown in (c). The source node is node 1 and the destination nodes are the labels of shaded circles. The end-to-end delay of the tree are shown in Table 5.3.

Grow-Tree Algorithm With The Quickest Paths

Suppose Grow-Tree is based on the widest-shortest paths or the shortest path with a weight $w(e)$ of an link $e \in E$ with $w(e) = r/B(e) + D(e)$. Since any subpath of a shortest path is also the shortest path, the path $P(s \sim u, v)$ with minimum end-to-end delay time by adding an link (u, v) to the existing multicast tree results in nothing but the shortest path from s to v where $P(s \sim u)$ is the shortest path from s to u. This implies that Grow-Tree based on the widest-shortest paths and the shortest path with $w(e) = r/B(e) + D(e)$ results in the same multicast tree based on WSP and MLW respectively. Thus, we consider Grow-Tree based on the quickest path only.

Let Grow-Tree denote the multicast tree obtained by algorithm Grow-Tree. In the algorithm, the end-to-end delay time of path $P(s \sim u, v)$ obtained by adding an

link (u, v) to the path $P(s, u)$ is defined as Time$(s \sim u, v)$. Then, algorithm works as follows:

---

algorithm Grow-Tree

1. **while** there exist unMarked $d \in MC$ **do**
2. min-Time $= \infty$
3. **for** each node $u \in$ Grow-Tree **do**
4. **for** each unMarked $v \in adj[u]$ **do**
5. **if** Time$(s \sim u, v) <$ min-Time **then**
6. min-Time $\leftarrow$ Time$(s \sim u, v)$
7. selected-Link $\leftarrow e(u, v)$
8. add the selected-Link (u, v) to the Grow-Tree
9. mark node v
10. return(Grow-Tree)

To develop this heuristic algorithm, we considered the following property such that any subpath of the quickest path may not be the quickest path. Suppose that an link (u, v) has already been added to the existing multicast tree. Later, an link $e(w, v)$ is tested, and Time$(s \sim w, v)$ is less than Time$(s \sim u, v)$ where a node v is not a destination. Although a path $P(s \sim w, v)$ is quicker than a path $P(s \sim u, v)$ to send the message from s to v, this does not implies that the path from s to d via $P(s \sim w, v)$ is quicker than the path via $P(s \sim u, v)$. For this reason, the end-to-end delay time from s to d via path $P(s \sim w, v)$ may be inefficient. Thus,

in this heuristic algorithm, once an link (u, v), called selected-link, is added to the existing multicast tree, a node v is marked and not tested anymore and return a tree correctly.

Step 2 of Grow-Tree algorithm runs for the number of nodes in the current Grow-Tree, and neighbors which are not marked and adjacent to each node from step 2 is tested in step 3. Thus, these two steps take $O(m')$ with $m' \leq m$ where m is the number of links in network. In the case of step 1, it is terminated when all $d \in MC$ are marked. Since one node is marked for each step 2, it takes at most $O(n)$ time for all $d \in MC$ to be marked. For the step 4, if we keep track of the delay and bandwidth of the path, whenever tree is updated, it takes only $O(1)$ time to compute the Time$(s \sim u, v)$. To keep track of the delay and bandwidth of the path updated, $O(1)$ time is needed so that it is required only $O(1)$ time to execute the step 4. Therefore, the total time complexity of this algorithm is (nm') without any preprocessing step.

Algorithm and Tree	Multicast Switch Architecture		
	Pipeline	Store-and-Forward	Hybrid
Grow-Tree with quickest paths	104	370.7	170.7
Best-First with quickest paths	142	253	228.7
Best-First with shortest paths	142	253	228.7
Best-First with widest shortest paths	210	643.3	310

Table 5.3: This table displays the end-to-end delay of each of the multicast tree that results from the algorithms based on spanning tree computation. The message size chosen is 200, source node is 1 and destination nodes are $\{7, 8, 14, 15, 16\}$.

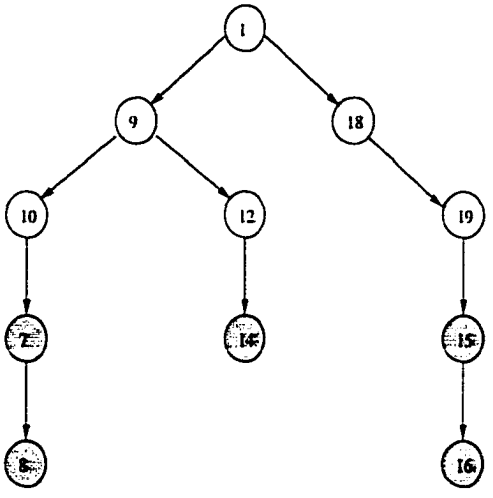

Figure 5.8: The multicast tree constructed by Grow-Tree algorithm is shown. The source node is node 1 and the destination nodes are the labels of shaded circles. The end-to-end delay of the above tree are shown in Table 5.3.

5.4 Bandwidth Allocation for Hybrid Switches

As indicated earlier, an hybrid switch buffers the complete packet from a link l_i before transmitting to the appropriate outgoing link l_j if $B(l_i) > B(l_j)$. This buffers causes an extra delay. In the following example, we will indicate that by appropriately reducing the bandwidth on the links the end-to-end delays can be improved in the case of hybrid switches.

Consider the multicast tree shown by Figure 5.9. Let s be the source node for the multicast tree with three destinations d_1, d_2, and d_3. Let P_i be the path from s to d_i for $1 \leq i \leq 3$. Let the delay D (the propagation delay) for the paths be such that $D(P_1) = 5$, $D(P_2) = 10$, and $D(P_3) = 50$. The label associated with each link in Figure 5.9 represents the bandwidth of the link. Let $r = 60$ be the size of the message to be multicasted. In Figure 5.9-(a) the multicast tree with the

initial bandwidth is shown and Figure 5.9-(b) shows the multicast tree with reduced bandwidths on some of the links. The minimum end-to-end delays for the trees in Figure 5.9-(a) and Figure 5.9-(b) are 70.5 and 52, respectively.

Our algorithm to determine the bandwidth on each link that minimizes the end-to-end delay works as follows. For each outgoing link $e(s, v)$ from s choose as candidate bandwidths the set of all bandwidths on links in the subtree rooted at v that are greater than or equal to $B(s, v)$. For the link $e(s, a)$ in Figure 5.9 (a) the candidates are $2, 3, 4, 5, 30$, and 35. Now, the link $e(s, a)$ is assigned a candidate bandwidth and the end-to-end delay for the tree is calculated using the calculations of hybrid switches described previously. The candidate bandwidth that results in minimum end-to-end delay is assigned to the link $e(s, a)$. The operation is performed for each of the outgoing links from s. The algorithm is executed now at node a, the child of s and the process is repeated until all links in the tree have been assigned the new bandwidth.

5.5 Simulation

We compared the performances of various algorithms under different switching architectures using extensive simulations. Our simulations were constructed using C programs written for the LINUX environment. We used four network generations models to generate our initial network and these include the Waxman I [45], Waxman II [45], Locality [49], and Transit-Stub models [49]. In our simulation, plane with size 1000 *km* x 1000 *km* are used for Waxman I, Waxman II, and Locality. For transit-stub mode, size of plane is 1000 *km* x 1000 *km*, size of each transit do-

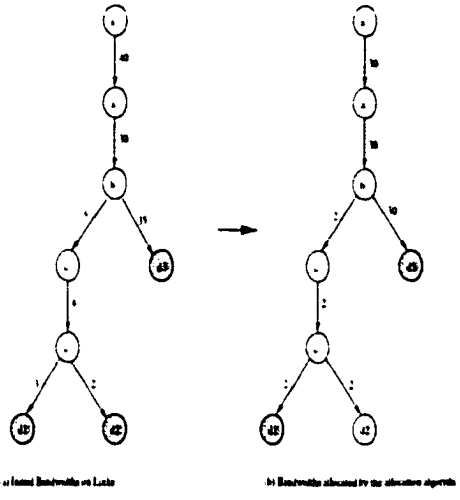

Figure 5.9: The given multicast tree before bandwidth allocation is shown in (a), the multicast tree after bandwidth allocation is shown in (b). The label on each link represents the bandwidth of the link. The source node is node 1 and the destination nodes are $\{d_1, d_2, d_3\}$ with shaded circles. The end-to-end delay of the above tree taking into account bandwidth on links and message size of 200 is 70.5 units of time for the hybrid switch before bandwidth allocation. The end-to-end delay of the above tree after bandwidth allocation is 52 for hybrid switch.

main is 150 *km* x 150 *km*, and size of each sub domain is 50 *km* x 50 *km*. The random networks generated using the models are connected with each node having an average degree in the range $[4, 7)$. We choose networks with number of nodes equal to 20, 40, 60, 80, and 100. Without loss generality, a source and a set of destinations are uniformly selected for each multicast session. The number of destinations that were selected were 10%, 20%, 30%, 40%, 50%, and 60% of the total number of nodes in a network. For each network that was generated, there are two different weights on each link were assigned: link delay and bandwidth. The link bandwidths are partitioned into two classes, low-bandwidth and high-bandwidth. Links with bandwidths in the range $[32Kbit/sec, 1,544Mbit/sec]$ and in the range $[6.312Mbit/sec, 155Mbit/sec]$ are classified as low and high, respectively. The link bandwidths are assigned in such way that 10/high bandwidth and 20/90/In the case of delay-time, distance is first uniformly distributed on each link with unit *km*, this distance is converted to delay-time using the formula

$$T_p = \frac{distance}{velocity\ of\ propagation\ of\ an\ electrical\ signal}$$

, where T_p represents the delay(propagation)-time of an link. For example, if an assigned distance is 10 *km*, then $T_p = \frac{10 \times 10^3}{2 \times 10^8} = 5 \times 10^{-5} s$.

The overall simulation experiment is organized as in the following algorithm for each model of network generation.

---

algorithm Simulation

1. **for** each size N (the number of nodes) in the set {20, 40, 60, 80, 100} **do**
2. **for** $i = 1$ **to** 30 **do**

3. Generate a network G with N number of nodes

4. Randomly assign link delays to the network G

5. **for** $j = 1$ **to** 9 **do** !Choose % of Low bandwidth

6. Assign $j \times 10$/of the links with high bandwidth

7. **for** $k = 1$ **to** 6 **do** !Choose % of destinations

9. **for** $l = 1$ **to** 30 **do** !Choose destinations

10. Randomly choose source s and $k \times 10\%$ of destinations

11. Run all the heuristics for various message sizes (in bytes) r={32, 256, 1024, 2048, 5120, 10240, 20480, 51200, 61400, 65535} and accumulate statistics.

5.5.1 Simulation Results

In this subsection, we will present the results of our simulation. Our extensive simulations indicate that the relative performances of various heuristics remained the same for all different sizes of networks and destinations. We will present the results for a network with 100 nodes and 20 destinations. The message sizes that were chosen for this presentation were 256 bytes, 1024 bytes and 65535 bytes (the maximum size of an IP packet). We will present our results only for Locality and transit-stub models of network generation since locality, WaxmanI and WaxmanII models showed very similar performance characteristics.

Store-and-Forward Switches

Since the Dijkstra's shortest path is the optimal algorithm, the Modified Link Weight (MLW) heuristic should perform the best and the simulation results as indicated in Figure 5.10 and Figure 5.11 attests to this fact. Grow Tree heuristic based on Quickest Paths (GTQP) performs admirably well even in the case of store-and-forward switches. The heuristics based on widest shortest path (WSP and BFWS) performs poorly compared with quickest path based algorithms. In the presence of a network containing many links with low bandwidths and taking into consideration large message size the performance of all the heuristics except based on widest shortest paths is very similar. In the transit-stub model of network generation the relative performances of various heuristics remain the same, but the differences are very small.

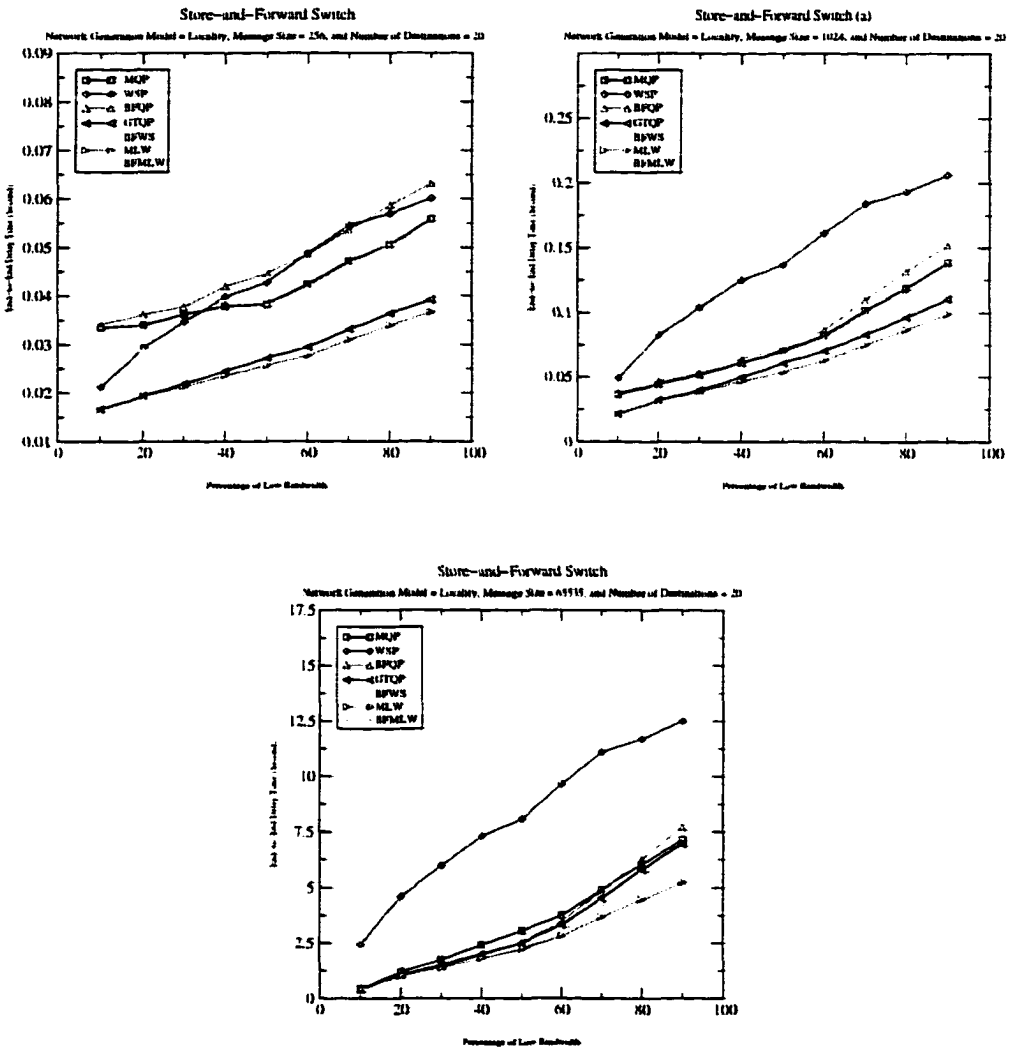

Figure 5.10: Store-and-Forward Switch with Network Based on Locality model

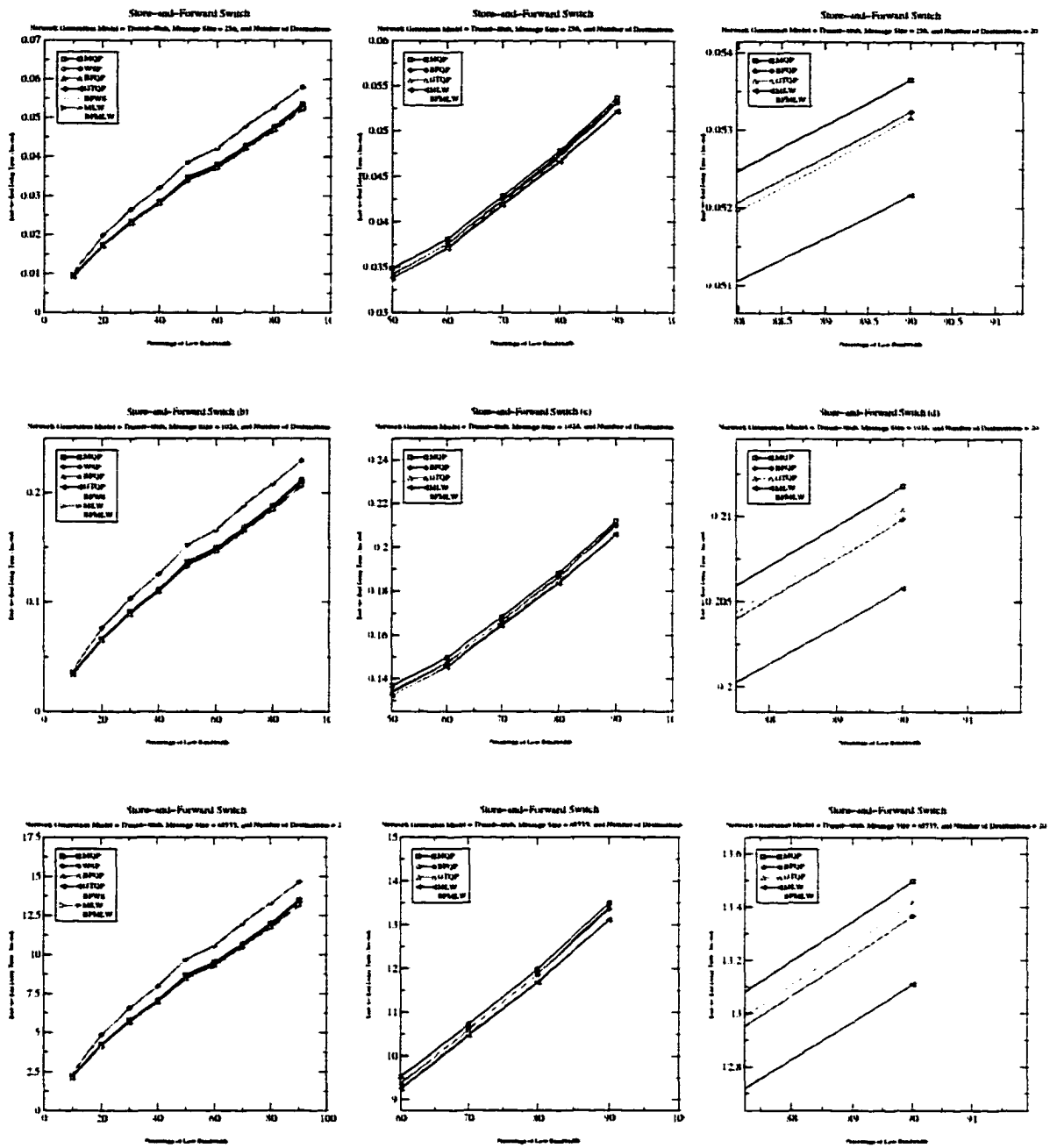

Figure 5.11: Store-and-Forward Switch with Network based on Transit-Stub model. The graphs in the first column contain all the heuristics, the second column shows the differences in performances for low bandwidths greater than 50% and the third column shows the relative performances for low bandwidths greater than 85%

Pipeline Switches

The Grow Tree heuristic based on quickest path is the best heuristic for all networks and messages sizes except for very large message size in which case the merging quickest path heuristics performs the best. The differences in performances can be clearly seen in the case of small message sized for the locality model (Figure 5.12). These differences fade away for all messages sizes in the case of transit-stub model. This is because of the bottleneck on the links in the transit domain. In Figure 5.13 we show the graphs for the transit-stub model. The graphs in the second and third column are presented to indicate the relative performance of the algorithms in case of network with many links containing low bandwidths.

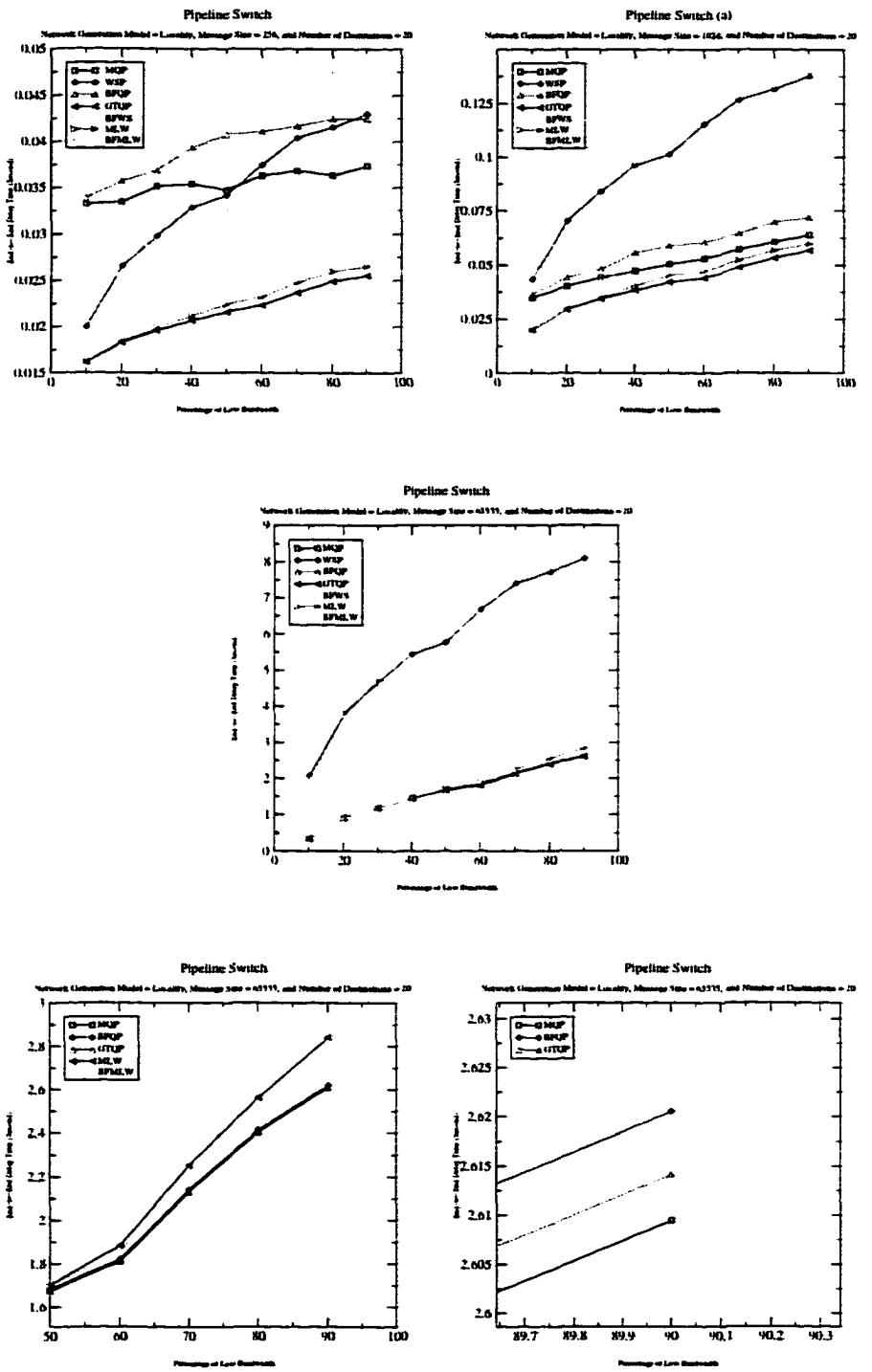

Figure 5.12: Pipeline Switch with Network Based on Locality model

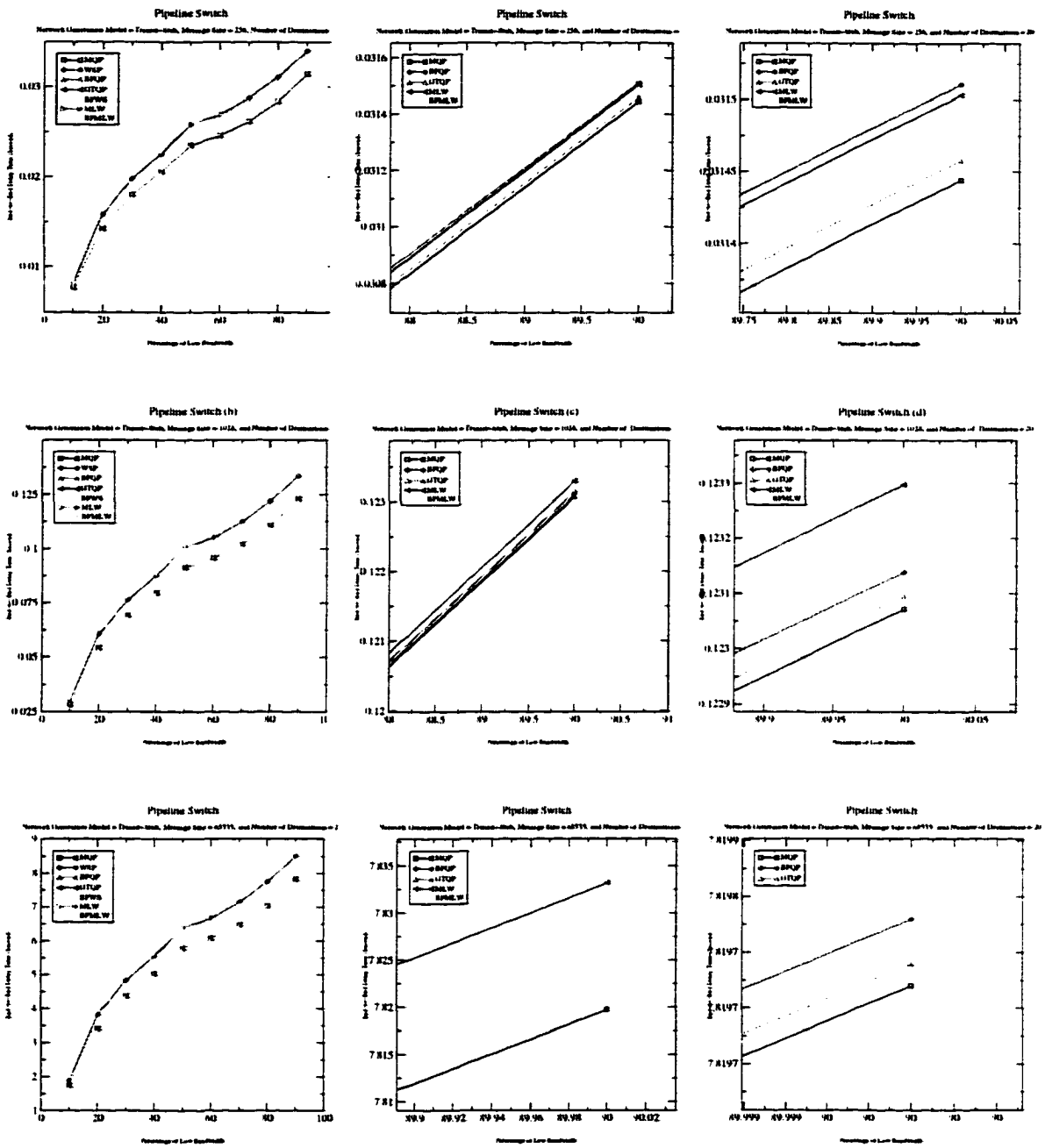

Figure 5.13: Pipeline Switch with Network Based on Transit-Stub Model The graphs in the first column contain all the heuristics, the second column shows the differences in performances for low bandwidths greater than 50% and the third column shows the relative performances for low bandwidths greater than 80%

Hybrid Switches

The hybrid switches behave like store and forward switches in some cases and on the average heuristics based on modified link weight perform admirably well in the case of hybrid switches. The grow tree heuristic's performance matches very closely with that of Modified link Weight. In comparison with the store-and-forward switches, the hybrid switches have obvious lower end-to-end delays but the relative performances of various heuristics appear to be the same. The graphs based on locality models are shown in Figure 5.14. The grow tree heuristic based on quickest path (GTQP) is slightly better than the one based on modified link weight for the case of transit-stub models (see Figure 5.15). Interestingly, the Merging Quickest Path (MQP) is the worst heuristic among all the heuristics that takes into account bandwidth and message size into account in the transit-stub model.

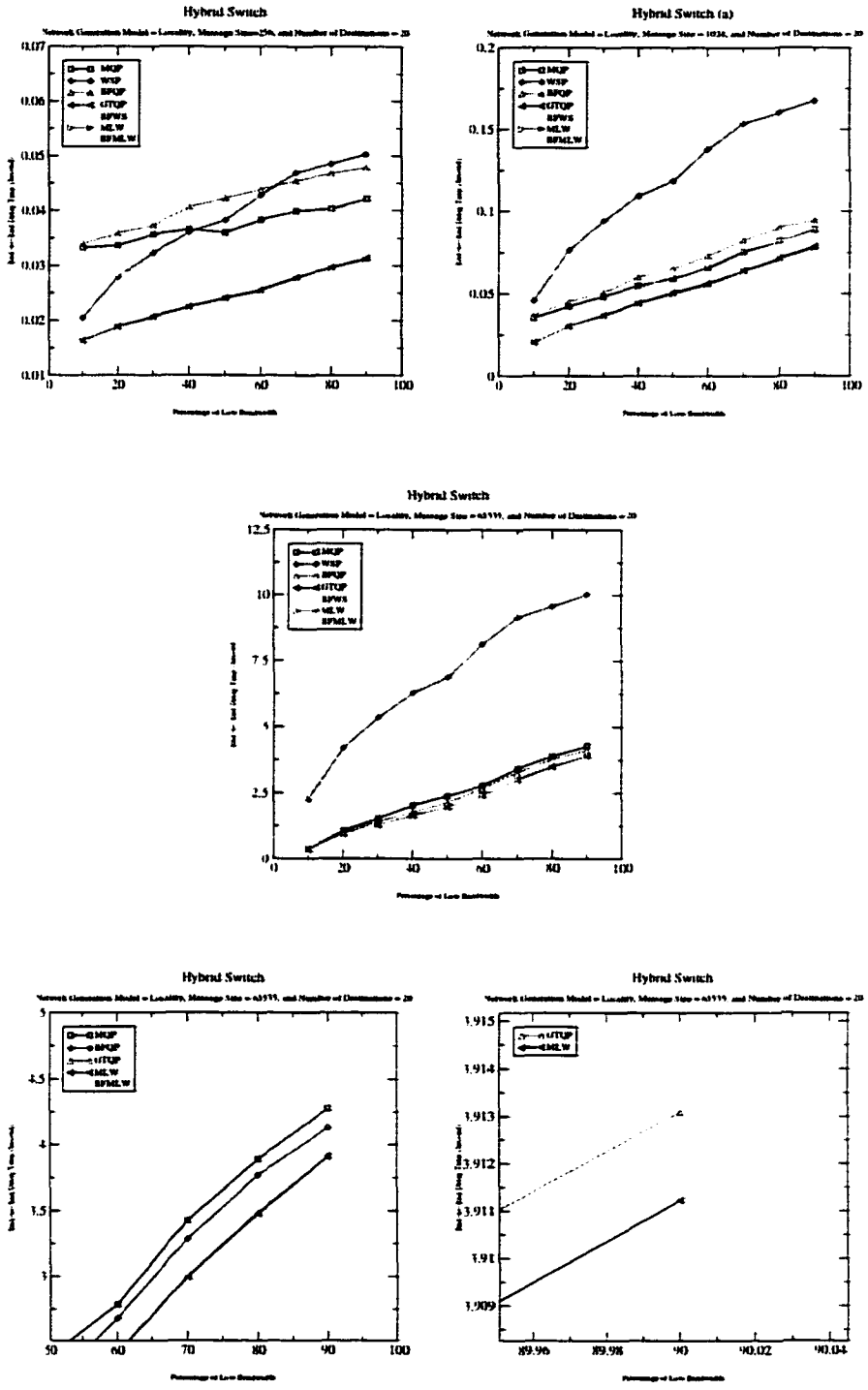

Figure 5.14: Hybrid Switch with Network Based on Locality Model

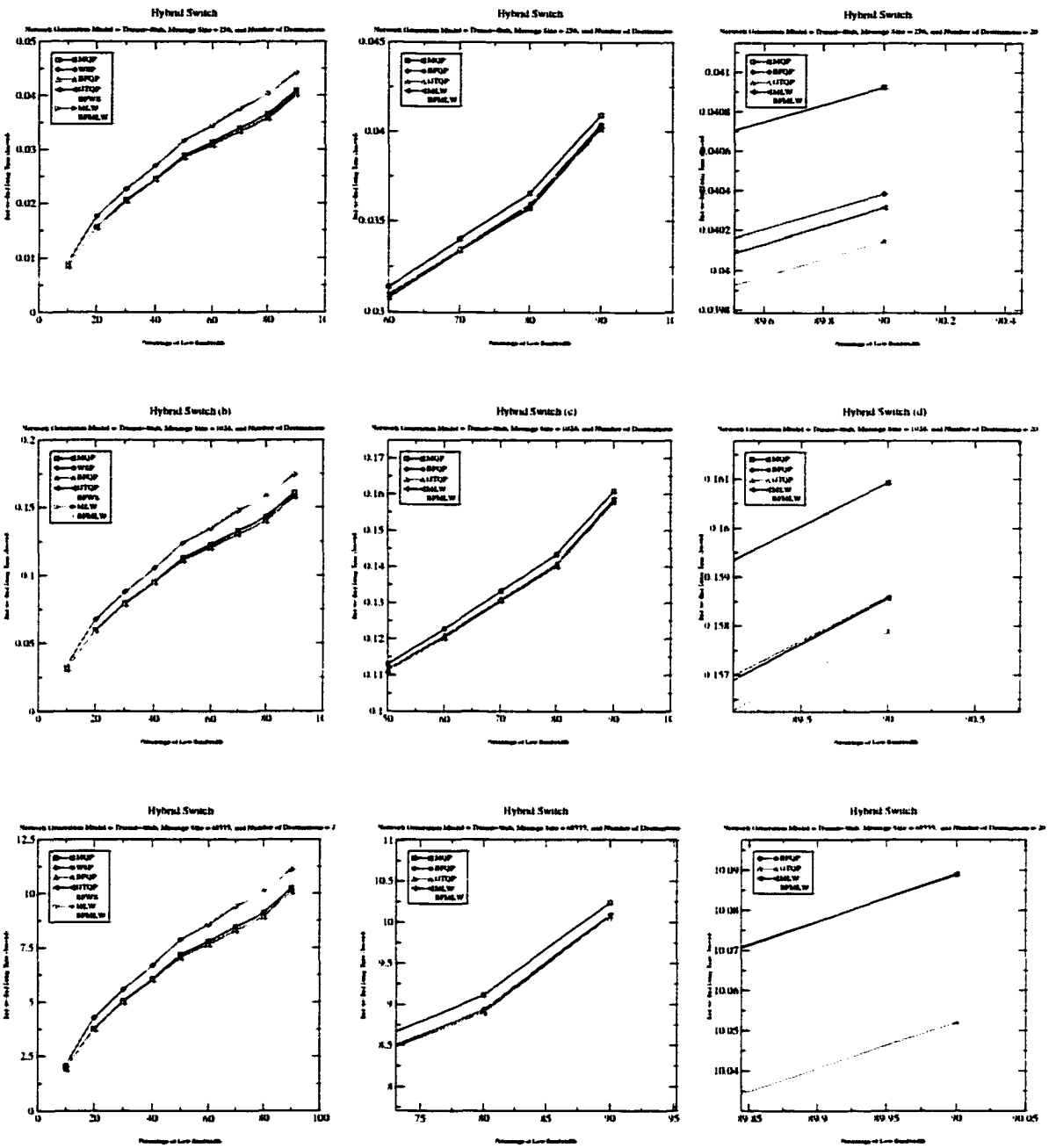

Figure 5.15: Hybrid Switch with Network Based on Transit-Stub Model The graphs in the first column contain all the heuristics, the second column shows the differences in performances for low bandwidths greater than 50% and the third column shows the relative performances for low bandwidths greater than 80%

Bandwidth Allocation for Hybrid Switches

As indicated earlier, the end-to-end delays can be improved in the case of hybrid switches if the bandwidths on the links can be appropriately reduced. The graphs in Figures 5.14, 5.15, 5.16, and 5.17 indicate this. The grow-tree heuristic performs the best in these cases and the merging quickest path heuristic outperforms all others when message size is large and the network contains many links with low bandwidths. These results are the same for both the locality and transit-stub model of network generation.

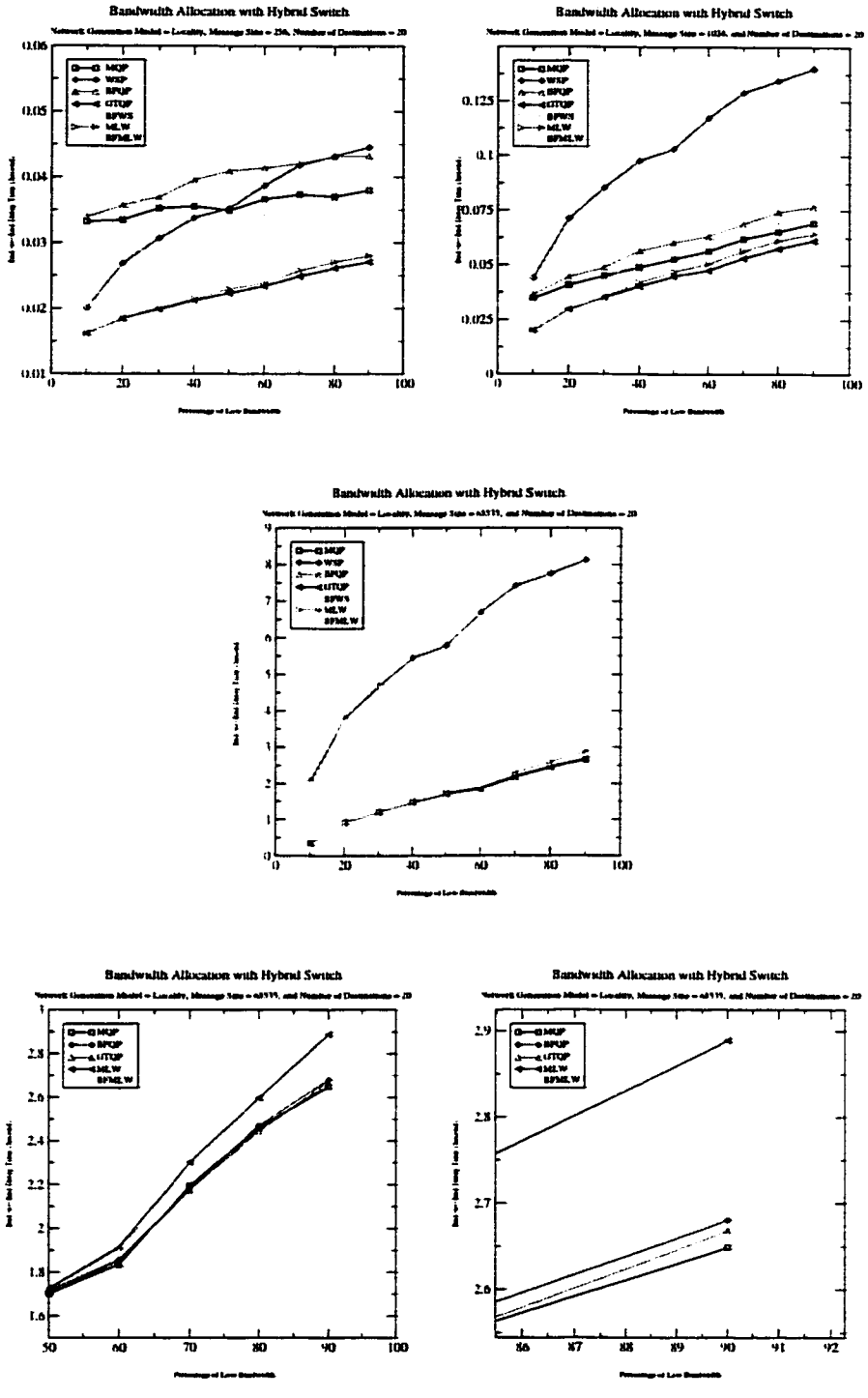

Figure 5.16: Bandwidth Allocation with Hybrid Switch with Network Based on Locality Model

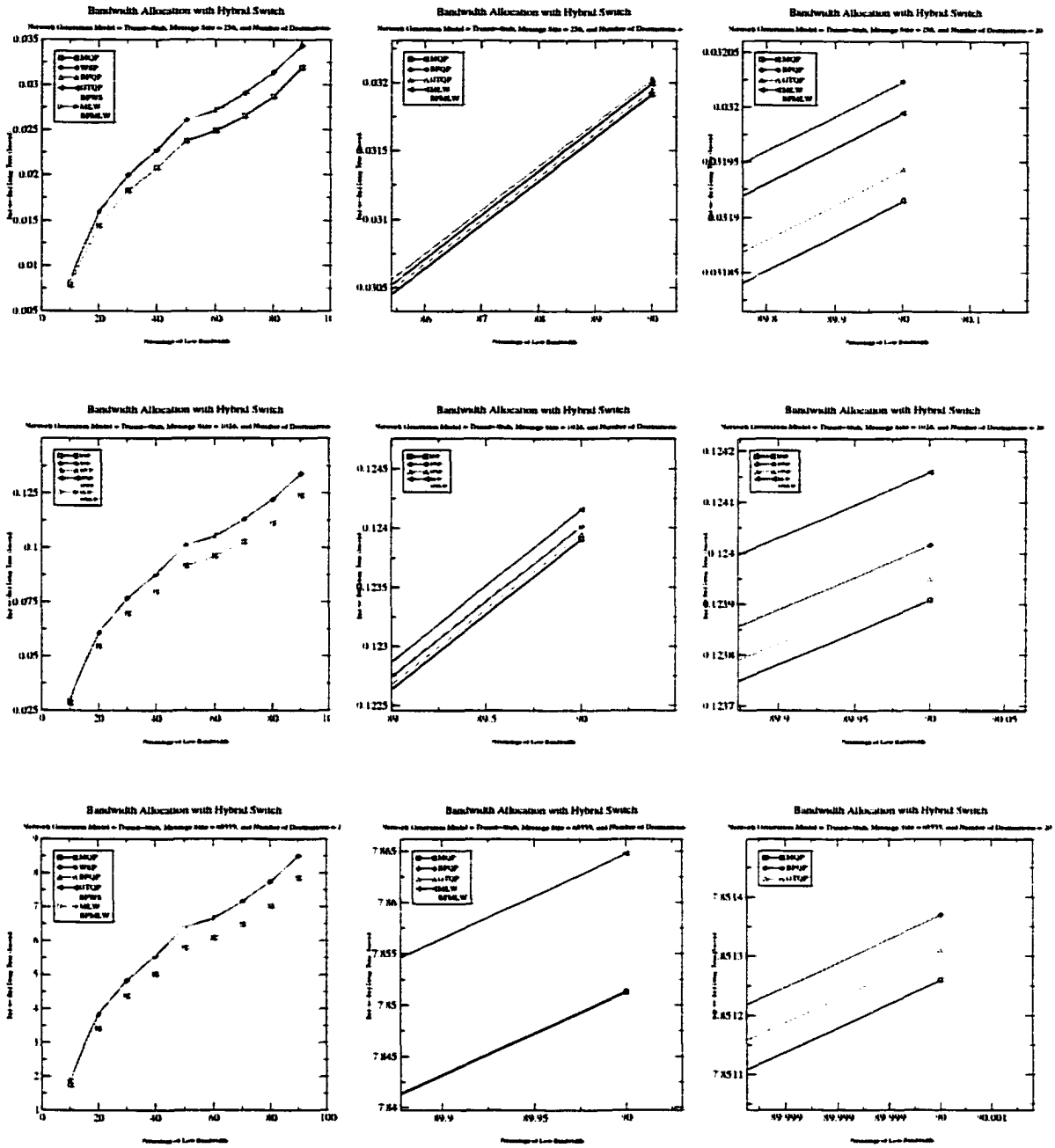

Figure 5.17: Bandwidth Allocation with Hybrid Switch with Network Based on Transit-Stub Graph The graphs in the first column contain all the heuristics, the second column shows the differences in performances for low bandwidths greater than 50% and the third column shows the relative performances for low bandwidths greater than 89%

5.6 Concluding Remarks

We considered the transmission of a message of size σ from a source to a set of destinations with minimum end-to-end delays over a computer network where bandwidth can be reserved and guaranteed on the links. Various heuristics were proposed and evaluated extensively on networks generated by different graph generation models. Impact of bandwidth and message sizes were studied for several network sizes. It is indicated that the Grow-Tree heuristic performs admirably well in all different switching architectures. Clearly, the heuristic of choice depends on the switching architecture, message sizes, and percentage of low bandwidths.

Chapter 6

On Multicasting with Minimum Costs for the Internet Topology

6.1 Introduction

We consider a computer network represented by a graph $G = (V, E)$ with n nodes and m arcs or links, where V and E are a set of nodes and a set of arcs (links), respectively. Each link $e(i, j) \in E$ is associated with *cost* $C(e) \geq 0$. Consider a *simple path* P form i_0 to i_k given by $(i_0, i_1), (i_1, i_2), \ldots, (i_{k-1}, i_k)$, where $(i_j, i_{j+1}) \in E$, for $j = 0, 1, \ldots, (k-1)$, and all $i_0, i_1, \ldots, i_k$ are distinct. Subsequently, a simple path is referred to simply as a path. The *path-cost* of P is given by $C(P) = \sum_{j=0}^{k-1} C(e_j)$, where $e_j = (i_j, i_{j+1})$. The *tree-cost* of tree T is given by $C(T) = \sum_{e} C(e)$, for all $e \in T$. The objective is to minimize the $C(T)$ under networks with asymmetric links.

In previous research, most of the multicast problems to minimize the cost of mul-

ticast tree only considered **undirected graph** as the *underlying* network. However, it is revealed that link utilization of link $e(u, v)$ is different from that of link $e(v, u)$ in real network environment, especially in **WANs** [14], thus we cannot consider those algorithms as efficient methods when applied to asymmetric networks.

On the other hand, to evaluate the performance of the multicast tree, network models constructed by random graph generators are used and the choice of random network model is very important. Since different network topology may produce different results (sometimes even significantly different), after simulating the same algorithm, the random network model which almost reflects the current Internet topology should be used to get accurate result. Currently, network model generated by the random graph generator designed by Waxman [45] is being used most frequently for simulation, but unfortunately this model, called WaxmanI, does not reflect the real current Internet. Several new random network models have been introduced in [49], and new models called Locality, N-Level Hierarchy, and Transit-Stub are more similar to the current Internet topology rather th an WaxmanI. Among these models, Transit-Stub almost reflects the current real Internet topology [49].

As indicated in [49], the routing characteristics of the Transit-Stub model follows that a shortest path will traverse Transit domain(s) if and only if the two endpoints are in different domains. This means that costs of links passing through Transit domain(s) to go to destinations are cheap which means those links have relatively large bandwidths. Also, in real Internet, when a message is multicasted, it passes through backbones if destinations are in different domains. Thus, paths to different destinations are likely to share a lot of same links. Using these characteristics, we may think that it is likely efficient to construct a multicast tree with minimum costs

using shortest (minimum cost) paths.

The algorithm (TM) due to Takahashi and Matsuyama [41] is a shortest-path-based algorithm and was further studied and generalized by Ramanathan [31]. The algorithm (KMB) by Kou, Markowsky, and Berman [27] is a minimum spanning tree based algorithm. Ramanathan [31], in his comparison between parameterized TM and KMB, has shown that TM outperforms KMB in terms of the cost of the tree constructed. Moreover, unlike the KMB algorithm, TM works on asymmetric directed networks. Our proposed algorithm like the TM is a shortest path based multicast tree construction algorithm. We show in this chapter that our algorithm produces multicast trees with lower tree costs in comparison with the TM algorithm for Internet like networks.

Our algorithm consists of the following two summarized steps which we will elaborate in the next section.

1. Using the cost $C(e)$ associated with each link e, determine all the minimum cost paths (will be referred as also the shortest path) from s to each $d_i \in D$ in the given network. Each minimum cost path from s to $d_i \in D$ forms a directed path. Merge all the minimum cost paths from s to each of the d_i's to form a directed acyclic graph G'. Note that each link in G' is a link in the original network and has the same associated cost.

2. Find a multicast tree given s and $MC = \{d_1, d_2, ..., d_k\}$ on the directed acyclic graph G'.

We will show in the next section that using a simple modification to the Dijkstra's shortest path algorithm we can indeed construct G'. In order to find the multicast tree on the graph G' we propose a new heuristic called the *most shared link* (MSL)

and show by empirical evaluation that our algorithm is superior to both TM and KMB for networks that follow the Internet topology. Given a directed acyclic graph, no polynomial algorithm for finding the multicast tree with minimum cost is known to exist [46].

Our contribution in this research is three-fold. First, we propose a new heuristic for multicast tree construction in directed asymmetric networks. Second, the sizes of input random networks are large and they reflect the Internet topology. We use the Transit-Stub network generation model proposed by [49]. Third, we have compared our heuristic with the two well-known heuristics, the TM and KMB and determined that the cost of the tree constructed by our algorithm is smaller than those constructed by TM and KMB. The time-complexity of our algorithm is the same as that of the TM algorithm.

The rest of this chapter is organized as follows. In section 6.2, we present details of our algorithm. We discuss the network model and the results of our simulation in section 6.3. Conclusions are presented in section 6.4.

6.2 Algorithm

We will first present the modified Dijkstra's shortest path algorithm to compute the network G' discussed in the introduction.

Let us define the predecessor data structure as an array $Pred[j]$ for $j \in V$, where each element of the array points to a linked list. If $Pred[j] \rightarrow p \rightarrow q$, then there are two paths from s to j with one of them passing through the link (p, j) and the other through the link (q, j). Thus the data structure $Pred$ can store all shortest

paths for every pair of nodes (s, d), where s is the source and d is a member of the multicast group (MC). The following algorithm is a modification of the Dijkstra's algorithm that is used to determine *Pred*.

Algorithm Compute_Pred ($s \in V$)

Comment1 : Assume that the minimum costs for all (s, i) with $i \in V$ is precomputed.

Comment2 : Let minimum cost for any (s, i) be $min(i)$.

Begin

1. cost[v] = ∞ for each node $v \in V$
2. cost[s] = 0
3. $Pred[s][i]$ = NULL for $i \in V$
4. $S \leftarrow \phi$
5. $Q \leftarrow V[G]$
6. **while** $Q \neq \phi$ **do**
7. $u \leftarrow$ a node with minimum of cost[u] from Q
8. $S = S + \{u\}$
9. **for** each vertex $v \in Adj[u]$ **do**
10. **if** cost[v] > $cost[u] + C(u, v)$ **then**
11. cost[v] = $cost[u] + C(u, v)$
12. **if** cost[v] = $min[v]$ **then**
13. $Pred[v] \rightarrow$ node = u
14. **endif**
15. **endif**
16. **endfor**

17.**endwhile**

End Compute_Pred

The time-complexity of the above algorithm is $O(m + n \log n)$ using Fibonacci heaps implementation of the Dijkstra's algorithm [1]. Using the *Pred* information we can construct the network G' in time $O(m)$. Note that G' is a directed acyclic network.

In order to determine the multicast tree on G' using our most shared links approach, we have to first label the arcs in G'. An arc (u, v) in G' has a set of labels. L_{uv}. A node label (or identifier) $d \in MC$ is in L_{uv} if and only if there is path from node u to d using the arc (u, v). That is, the label on an arc indicates the set of destinations that can be reached by using that arc. The label for each arc in G' can be determined by backtracking from each d to s after temporarily reversing the arcs in G' in $O(m)$ time. Next we need the following definition.

Definition 6.2.1 *If $L_{uv} \subseteq L_{uw}$ then an arc (u, v) is redundant, otherwise non-redundant.*

The heuristic to determine the multicast tree T from G' consists of the following steps. Note that we use a clever implementation of the following steps to reduce the time complexity of the algorithm.

1. Perform a breadth first search starting from the node s (the root of tree T).

2. For each node u encountered during the breadth first search perform the following two steps.

a. Remove arcs (u, v) that are redundant.

b. Modify the labels L_{uv} of non-redundant arcs as follows: Let $L' = L_{uv} \cap L_{uw}$ with $|L_{uv}| \leq |L_{uw}|$ and $v < w$. Assume that $L' \neq L_{uv}$ and $L' \neq L_{uw}$. Now label $L_{uv} = L_{uv} \setminus L'$.

3. Prune the breadth first search tree to remove arcs that do not reach any $d \in MC$. This can be accomplished by performing a depth first search.

Steps 1 and 2 are implemented using clever data structuring in the following algorithm.

Algorithm RRL(G', MC)

Input : G' the acyclic directed network and MC the multicast group with $D = |MC|$.

Output : The multicast tree T

comment 1: Let CV be a current Boolean vector of size D.

comment 2: Let FP be a Boolean vector of size D for each node in G'.

comment 3: Let A be a Boolean vector of size D for each arc in G'.

comment 4: Let $N(u, v)$ be the number of destinations that can be reached by using the arc (u, v). Note that $N(u, v) \leq D$.

comment 5: Let L_{uv} be the label on arc (u, v).

comment 6: $Q = \emptyset$ initially.

Begin

1. Assign L_{uv} for each (u, v) in G' by backtracking from each $d \in MC$.

2. **for** $i = 1$ to D **do**

3. $CV[i] \leftarrow$ false

4. $FP_s[i] \leftarrow$ true

5. **endfor**

6. $Q \leftarrow Q \cup \{s\}$

7. **while** Q not empty **do**

8. remove p from Q

9. **for** each arc (p, v) **do**

10. **for** $i = 1$ to D **do**

11. $A_{pv}[i] \leftarrow$ false

12. **endfor**

13. **for** each $x \in L_{pv}$ **do**

14. $A_{pv}[x] \leftarrow$ true

15. **end for**

16. $A_{pv} \leftarrow FP_p \wedge A_{pv}$

17. **end for**

18. compute $N(p, v)$ based on true value in A_{pv}

19. sort the arcs (p, v) based on $N(p, v)$ in descending order
using the bucket sort. Let $(p, v_1), (p, v_2), \ldots, (p, v_k)$ be the sorted arcs.

20. $CV \leftarrow A_{pv_1}$

21. **for** $i = 2$ to k **do**

22. **for** $j = 1$ to D **do**

23. **if** $A_{pv_i}[j] ==$ true **then**

24. **if** $CV[j] ==$ false **then**

25. $CV[j] \leftarrow$ true

26. **else**

27. $A_{pv_i}[j] \leftarrow$ false

28. $N(p, v_i) \leftarrow N(p, v_i)$ - 1

29. **endelse**

30. **endif**

31. **endfor**

32. **endfor**

33. remove all arcs (p, v_i) with $N(p, v_i) = 0$

34. place all neighbor v_i of p with $N(p, v_i) > 0$ in Q

35. $FP_{v_i} \leftarrow A_{pv_i}$

36. **endwhile**

End RRL

The time-complexity of the algorithm RRL is evaluated as follows. Step 1 can be completed in $O(n + m)$ time using depth first search. Steps 2-5 take $O(D)$ time which is at most $O(n)$. The while statement in step 7 is executed at most $O(n)$ time. Steps 9-17 is executed at most $O(m \times D)$ during the entire execution of the algorithm. Steps 18-19 can be completed in $O(m \times D)$ during the breadth first search as we visit nodes level by level. Steps 21-32 can be executed in $O(m \times D)$ during the entire execution of the algorithm. Hence the total time complexity of the above algorithm is $O(m \times D)$. The total time complexity of our entire algorithm including the construction of G' is $O(m \times D + n \log n)$. The time complexity of KMB is $O(n^2 \times D)$ [27] and TM has a time-complexity of $O(m \times D + n \log n)$ using Fibonacci implementation of the Dijkstra's shortest path algorithm [41].

Figure 6.1-(a) shows a directed asymmetric network. Assuming that the source

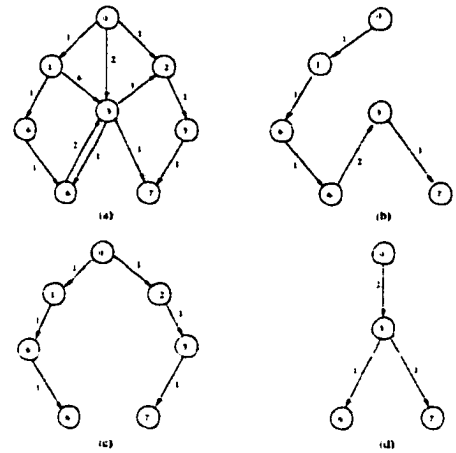

Figure 6.1: Given a network (a), a multicast tree based on KMB is shown in (b), a multicast tree based on TM is shown in (c), and a multicast tree based on MSL is shown in (d), where a source is 0, and $MC = \{6, 7\}$.

node is labeled 0 and the $MC = \{6, 7\}$, the trees constructed by algorithms KMB, TM, and our algorithm are shown in Figures 6.1-(b),(c), and (d), respectively. The cost of the trees generated by KMB, TM and our algorithm are 6, 6, and 4, respectively. The subnetwork G' for the network in Figure 6.1-(a) with $s = 0$ and $MC = \{6, 7\}$ is shown in Figure 6.2-(a). The text on each arc is the arc label. Figure 6.2-(b) is the multicast tree T for G'.

6.3 Simulation

To compare performances of algorithms introduced, we used extensive simulations. We implemented the algorithms in language C under the Linux environment. As mentioned earlier we used the Transit-Stub model proposed by Zegura et. al [49] to generate our random networks. We set the size of plane, each transit domain, and each stub domain to 1000 *km* x 1000 *km*, 150 *km* x 150 *km*, and 50 *km* x 50 *km*,

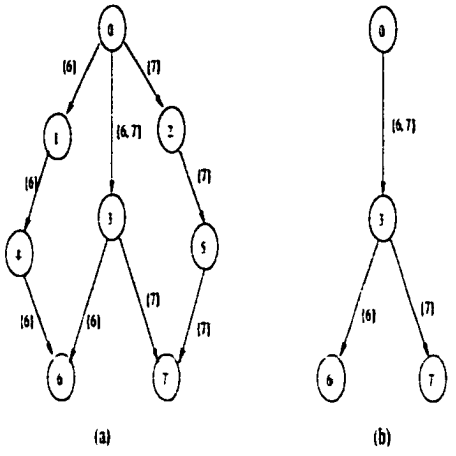

Figure 6.2: Subnetwork by merging all minimum cost paths from source to each destination is shown in (a) and tree derived from (a) is shown in (b) .

respectively.

Note that the Waxman model [45] is a popular graph generation model, but generates graphs that do not reflect the Internet topology [49].

The weights on the arc represent the cost of using the arc. In the Transit-Stub model, the cost of the arcs in the backbone network is less compared with other arcs in the network. Semantically, it is advisable to use the backbone to route traffic between inter-domain nodes since the backbone has a higher bandwidth in comparison with other arcs.

We considered networks with number of nodes equal to 117, 204, 315, 420, and 500. We generated 30 different networks for each size given above. The random networks used in our experiments are directed and connected, where each node in graphs has the average degree 4. Without loses of generality, a source and a set of destinations are uniformly selected for each multicast session. The number of destinations chosen by our simulation was in range from 10–300 depending upon the

size of the graph.

The overall simulation experiment is organized as in the following algorithm.

Algorithm Simulation

1. **for** $i = 1$ **to** 30 **do**
2. Generate a network G with N number of nodes using the Transit-Stub model.
3. Randomly assign cost to arcs to the network G
4. **for** $k = 1$ **to** 6 **do** !Choose number of destinations
5. **for** $l = 1$ **to** 30 **do** !Choose destinations
6. Randomly choose a source s and x number of destinations based on size N
7. Run TM, KMB, and MSL heuristics and evaluate the cost of the tree constructed
8. **endfor**
9. **endfor**
10. **endfor**

6.3.1 Simulation Results

In this subsection, we will present the results of our simulation. To evaluate performances of MSL, we compare with previous heuristics KMB [27] and TM [41]. Since algorithm proposed in [31] has the best performance when it is equivalent to TM, we omit the algorithm proposed in [31] in our evaluation. Our simulation indicate that

the relative performances of two heuristics KMB [27] and TM [41] remained almost same for all different sizes of the multicast group. However, the simulation revealed that the performance of our algorithm MSL is relatively better than those of both KMB and TM. We will present results for networks with 117 nodes, 204 nodes, 315 nodes, 420 nodes, and 500 nodes.

Since it is impractical to find the optimal solution for large graphs, we used the normalized surcharge δ_H of algorithm [31] with respect to MSL defined as follows:

$$\delta_H = \frac{C(T_H) - C(T_{MSL})}{C(T_{MSL})} \tag{6.1}$$

In the above equation $C(T_H)$ is the cost of tree based on algorithm H and $C(T_{MSL})$ is the cost of tree based on algorithm MSL.

To depict relative performances by plots, δ_H is multiplied by 100 to express as a percentage.

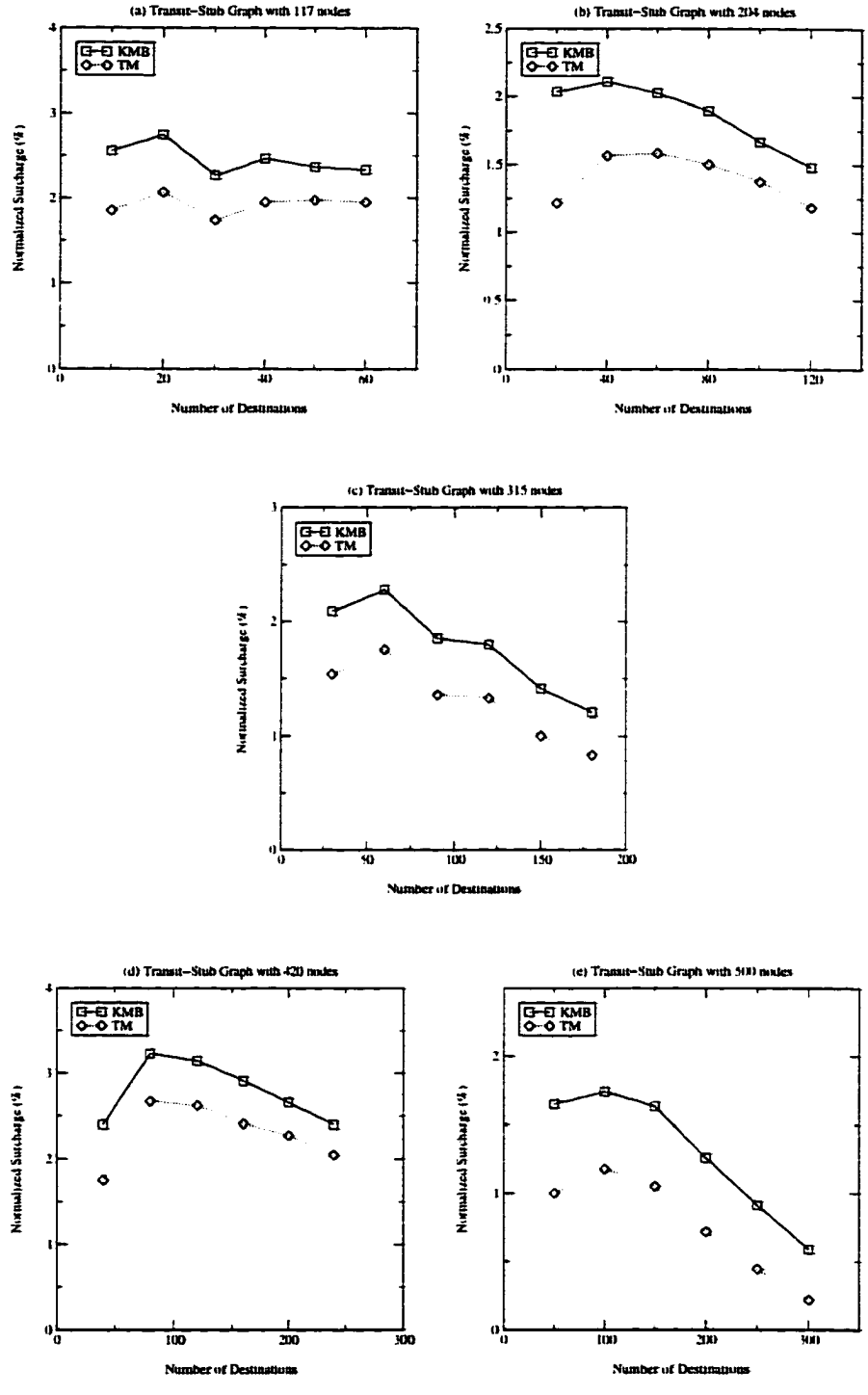

Figure 6.3: Normalized Surcharges versus the number of destinations for network with 117 nodes, 204 nodes, 315 nodes, 420 nodes, and 500 nodes, with respect to MSL .

As indicated in Figure 6.3, it can be easily noticed the MSL always outperforms KMB. Notice that the relative performance of MSL is highest when the number of destinations are about 20% of number of nodes, and it becomes decreasing after then.

6.4 Concluding Remarks

We considered the transmission of a message from a source to a set of destinations with minimum cost over a computer network. We presented a simple algorithm that specifies a multicast tree with minimum cost. We also presented simulation results to illustrate the relative performances of algorithms. One interesting result from simulation is that if adequate global information is known at the source and the network topology which is very close to the real Internet topology, the algorithm MSL we proposed outperforms KMB and TM which are most straightforward and efficient among algorithms known so far.

Chapter 7

Conclusion

In this dissertation, we have investigated various unicast and multicast routing algorithms in wide are networks.

In chapter 2, we presented five variations of the quickest path problem reflecting mechanisms such as circuit switching,Internet protocol and their combinations. We presented an unifying algorithm to compute the quickest paths in the first four modes, and for the last mode, Dijkstra's algorithm is adapted for computing the quickest path. It would be interesting to study for the computation of multiple paths for various modes.

In chapter 3, we developed efficient algorithms to construct and update the path-table that maps all intervals for σ to the corresponding quickest paths. We showed that the path-table can be built and updated in linear time. Future research direction is the computation of path-tables for all other modes.

In chapter 4, we studied the reliability problem of the quickest paths. For any value of message size σ, we showed that all-to-all the quickest most reliable paths and the most reliable quickest paths can be computed in $O(n^2m)$ time where n and

m are the number of nodes and the number of edges or links in the network.

In chapter 5, various heuristics for constructing multicast trees with the minimum end-to-end delay were proposed. Using various random network models, these heuristics were evaluated extensively. Impact of bandwidth and message sizes were studied for various network sizes. Our simulation showed that the Grow-Tree heuristic we proposed outperforms in all different switching architectures and the heuristic of choice depends on the switching architecture, message sizes, and percentage of low bandwidths

Finally, in chapter 6, we have introduced a simple algorithm that specifies a multicast tree with the minimum-cost over a computer network. Our simulation results showed that if adequate global information is known at the source and the network topology which is very close to the real Internet topology, the relative performance of the algorithm MSL we proposed is better than performances of KMB and TM which are most straightforward and efficient among algorithms known so far.

We studied intensively to develop new multicast tree heuristics for minimizing both end-to-end delay and cost, and introduced various already existing heuristics for each QoS. In addition to these heuristcs, many other heuristics were introduced for multicast routing with delay-bounded minimum cost which is the combination of minimum end-to-end delay and the cost [52, 26, 2, 40]. Future research direction for multicast routing will be on developing new novel heuristic for delay-bounded minimum cost.

Bibliography

[1] Ravindra K. Ajuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] Tawfig Alrabiah and Taieb F. Znati. Low-cost, bounded-delay multicast routing for qos-based networks. In *ICCCP1998*, 1998.

[3] P. Aukia. Quality of service based routing, 1996. Internet draft.

[4] S. Bahk and W. El-Zarki. Dynamic multi-path routing and how it compares with other dynamic routing algorithms for high speed wide area networks. In *Proc. of ACM SIGCOM*, 1992.

[5] Young-Cheol Bang, S. Radhakrishnan, Nageswara S. V. Rao, and Stephen G. Batsell. On update algorithms for quickest paths. Technical report, University of Oklahoma, 1999.

[6] F. Bauer and A. Varma. Distributed algorithms for multicast path setup in data networks. *IEEE/ACM Transations on Networking*, 4(2):181–191, 1996.

[7] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[8] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture, 1994. IETF RFC 1633.

[9] Gen-Huey Chen and Yung-Chen Hung. Algorithms for the constrained quickest path problem and the enumeration of quickest paths. *Computers Operations Research*, 21(2):113–118, 1992.

[10] Gen-Huey Chen and Yung-Chen Hung. On the quickest path problem. *Information Processing Letters*, 46(3):125–128, 1993.

[11] Y. L. Chen and Y. H. Chin. The quickest path problem. *Computers and Operations Research*, 17(2):153–161, 1990.

[12] H. Chu and K. Nahstedt. Dynamic multi-path communication for video traffic. In *Proc. of Hawian Int. Conf. on Systems Sci.*, 1997.

[13] Cisco. Routing basics. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito-doc/routing.htm.

[14] K. Claffy, P. Francis, and H. Braun. Traffic characteristics of the t1 nsfnet backbone. In *Proceedings of IEEE INFOCOM 93*, 1993.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill Book Co., New York, 1990.

[16] T.H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[17] Ernesto de Queiros Vieira Martins and Jose Luis Esteves dos Santos. An algorithm for the quickest path problem. *Operations Research Letters*, 20:195–198, 1997.

[18] J. J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Transactions on Networking*, 5(1):148–160, 1997.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Co., San Francisco, 1979.

[20] Ming-Huang Guo and Ruay-Shiung Chang. Multicast atm switches: Survey and performance evaluation. *ACM sigcomm*, 28:98–131, 1998.

[21] Fred Halsall. *Data Communications, Computer Networks and Open Systems.* Addison-Wesley, 1996.

[22] Yung-Chen Hung and Gen-Huey Chen. Distributed algorithms for the quickest path problem. *Parallel Computing*, 18:823–834, 1992.

[23] F. K. Hwang and D. Richards. Steiner tree probkems. *Networks*, 22:55–89, 1992.

[24] B. K. Kadaba and J. M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Transactions on Communications*, COM-31(3):343–351, 1983.

[25] H. Kanakia, P. P. Mishra, and A. R. Reibman. An adaptive congestion control scheme for real time packet video transport. *IEEE/ACM Transactions on Networking*, 3(6):671–682, 1995.

[26] V. P. Kompella, J. C. Pasquale, and G. C. Polyzoa. Multicast routing for multimedia communications. *IEEE/ACM Transations on Networking*, 1(3):286–292, 1993.

[27] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15:151–145, 1981.

[28] O. Kyas. *ATM Networks*. International Thomson Computer Press, 1997. Second Edition.

[29] D.T. Lee and E. Papadopoulou. The all-pairs quickest path problem. *Information Processing Letters*, 45(5):261–267, 1993.

[30] J. F. Mollenauer. On the fastest routes for convoy-type traffic in flowrate-constrained networks. *Transportation Science*, 10:113–124, 1976.

[31] S. Ramanathan. Multicast tree generation in networks with asymetric links. *IEEE/ACM Transations on Networking*, 4(4):558–568, 1996.

[32] N. S. V. Rao and S. G. Batsell. Algorithm for minimum end-to-end delay paths. *IEEE Communications Letters*, 1(5):152–154, 1997.

[33] N. S. V. Rao and S. G. Batsell. On routing algorithms with end-to-end delay guarantees. In *IC3N: International Conference on Computer Communications and Networks*, pages 162–167. 1998.

[34] N. S. V. Rao and S. G. Batsell. QoS routing via multiple paths using bandwidth reservation. In *IEEE INFOCOM98: The Conference on Computer Communications*, volume 1, pages 11–18. 1998.

[35] V. Rayward-Smith. The computation of nearly minimal steiner trees in graphs. *International Journal of Mathematical Education in Science and Technology*, 14(1):15–23, 1983.

[36] V.J. Rayward-Smith and A. Clare. On finding steiner vertices. *Networks*. 16:283–294, 1986.

[37] J. B. Rosen, S. Z. Sun, and G. L. Xue. Algorithms for the quickest path problem and the enumeration of quickest paths. *Computers and Operations Research*. 18(6):579–584, 1991.

[38] Hussein Farouk Salama. *Multicast Routing For Real-Time Communication On High-Speed Networks*. PhD thesis, North Carolina State University, 1996.

[39] M. L. Shore, L. R. Foulds, and P. B. Gibbons. An algorithm for the steiner problem in graphs. *Networks*, 12:323–333, 1982.

[40] R. Sriram, G. Manimaran, and C. Siva Ram Murthy. Algorithms for delay-constrained low-cost tree construction. *Computer Communications*, 21:1693–1706, 1998.

[41] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Mathematica Japonica*, 24(6):573–577, 1980.

[42] Jean Walrand. *COMMUNICATION NETWORKS.* McGraw-Hill, 1998.

[43] Jean Walrand and Pravin Varaiya. *HIGH-PERFORMANCE COMMUNICATION NETWORKS.* Morgan Kaufmann, 1996.

[44] Z. Wang and J. Crowcroft. QOS routing for supporting resource reservation. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, 1996.

[45] B. M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9), 1988.

[46] P. Winter. Steiner problem in networks. *Networks*, 17:129–167, 1987.

[47] G. Xue, S. Sun, and J. B. Rosen. Fast data transmission and maximal dynamic flow. *Information Processing Letters*, 66, 1998.

[48] Guoliang Xue. End-to-end data paths: Quickest or most reliable? *IEEE communication letters*, 2(6):156–158, 1998.

[49] Ellen W. Zegura, Kenneth L. Calvert, and Michael J. Donahoo. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Transactions on networking*, 5(6), 1997.

[50] A. Z. Zelikovsky. A 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9:463–470, 1993.

[51] L. Zhang, S. E. Deering, D. Estrin, S. Shankar, and D. Zappala. RSVP: A new resource reservation protocol communications network. Technical Report 95-607, ISI, Univeristy of Southern California, 1995.

[52] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for delay constrained minimum-cost multicasting. In *Proceedings of INFOCOM*, pages 377–385, 1995.