University of Nevada
Reno

# A Low Cost Algorithm for Dynamic Multicast Routing in Computer Networks

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
with a major in Computer Science.

by

Pingyan Tan

Dr. Frederick C. Harris, Jr., Thesis advisor

December 1995

The thesis of Pingyan Tan is approved:

---

Thesis Advisor

---

Department Chair

---

Dean, Graduate School

University of Nevada

Reno

December 1995

**Abstract**

This thesis presents a new algorithm named Minimum Average Distance (MAD) for multicast routing in computer networks. For performance evaluation, we will compare it with a different algorithm known as LS multicast routing which was presented in [6] in 1991 and has been introduced into Internet recently.

The new algorithm focuses on low-cost routing. It supports the *group* concept and the *unknown-destination* delivery. It uses a *minimum average distance* method to select the forwarding links to achieve the low-cost goal. The algorithm defines a distance radius and uses it to support the *unknown-destination* delivery without requiring the multicast packets to carry individual group member addresses. Another important feature of this algorithm is that no delivery tree has to be maintained, which saves memory and computation time. Moreover, its computation time complexity is only linear in the group size.

A simulation model is implemented. The simulation results strongly show that the new algorithm has a better performance than the compared algorithm, in both cost and delay.

**Acknowledgements**

I would like to thank Professor Harris, my advisor, for his generous help when it is most needed, for his guidance, support, and encouragement throughout the thesis process.

I am very grateful to Professor Wishart and Professor Johnson for serving on my committee and for their valuable time.

Finally, a special thank to my parents and my husband, for their support, their encouragement when things were not easy, and their belief in my abilities.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Routing is one of the most important aspects of computer networks. Unicast routing has been the dominant scheme used in network routing. However, in the past few years, multicast routing has seen widespread use in those networks that support it. There exist many cases where an application must send the same information to more than one destination, such as:

- updating all copies of a replicated file or database.
- sending data packets to all participants in computer mediated conference.
- emailing to more than one receiver.
- dissemination of intermediate results to a set of processors supporting a distributed computation.

Multicast routing can be defined as follows: Given a network $(N, L)$, where $N$ is a set of nodes and L a set of links. a source node $s \in N$, and a set of destination nodes $D \subseteq N$. If $1 < |D| < |N| - 1$, then the problem is a multicast problem. If $|D| = 1$, then the problem is a unicast problem and if $|D| = |N| - 1$, then the problem is a broadcast problem.

The subset $D$ is called a *group*, and there can be more than one group existing in a network at the same time. Multicasting should be more efficient than unicas-

ting separate copies to each destination, because it should reduce the transmission overhead.

Modern networks also require the *unknown-destination* delivery function for routing. If a set of destinations can be identified by a single group address, such a group address can be used to reach one or more destinations whose individual addresses are unknown to the sender, or whose address may change over time.

Algorithms for solving the multicast routing problem can be divided into two categories: low-delay algorithms and low-cost algorithms. Low-delay routing algorithms minimize the path length from the source to every destination node. This usually requires that the message be sent to every destination along the shortest path. Low-cost routing algorithms minimize the total traffic. The total traffic is equivalent to the total length of the path in a delivery tree. Finding a minimum cost routing tree is a Steiner Tree Problem in a graph, which is known to be NP-complete[8].

This thesis presents a new algorithm for multicast routing. This algorithm is aimed at low-cost routing, and supports the *unknown-destination* delivery function as well.

In Chapter 2 we review what is known about routing problems, describing different aspects of the problem and some known algorithms from the literature. Chapter 3 gives the formal description of our proposed algorithm and a simple network example to explain the algorithm. Chapter 4 develops a simulation model for our proposed algorithm and a compared algorithm which is presented in [6]. Chapter 5 concludes this thesis with a discussion of the advantages of our algorithm.

# Chapter 2

# Review of the Literature

The routing algorithms that we refer to will be the algorithms used in the network layer protocol that guides the packets through the communication subnet to their correct destination. The network is abstracted into a set of nodes and links, or a graph. A subnetwork is treated as a node at this level, so that the activities within a subnetwork are hidden. In other words, we consider point-to-point links.

## 2.1 Main Issues in Routing

The two main functions performed by a routing algorithm are the selection of routes for various source-destination pairs and the delivery of messages to their correct destination once the routes have been selected.

The two main performance measures affected by routing algorithms are packet delay and the throughput. The packet delay is defined to be the time between the generation of a packet and the arrival of that packet at its destination. The throughput is commonly defined to be the difference between the offered load and the rejected load, where the offered load is the messages issued by each node, and the rejected load is the messages rejected by the network because of traffic congestion[4].

## 2.2 Classification of Routing Schemes

There are a number of ways to classify routing algorithms. We briefly discuss some of the classifications [4, 6, 13], and then pay more attention to the ones that are related to our work.

The first classification we will look at is based on the decision time. The times at which routing decisions are made depend on whether the network uses datagrams or virtual circuits. In a datagram network, two successive packets of the same source-destination pair may travel along different routes, and a routing decision is necessary for each individual packet. In a virtual circuit network, a routing decision is made when a virtual circuit is set up. We are mainly concerned with datagram routing.

The second classification divides routing algorithms into centralized and distributed. In centralized algorithms, all routing choices are made at a central node, while in a distributed algorithm the computation of routes is shared among the network nodes with information exchanged between them as necessary.

The third classification of routing algorithm is based on the routing strategies. There are four major types of routing strategies: fixed routing, flooding routing, random routing, and adaptive routing.

One of the simplest routing strategies is fixed routing. In this case, a route is selected for each source-destination pair of nodes in the network. The routes are fixed, or at least only changed when there is a change in the topology of the network. Thus the costs used in designing routes cannot be based on any dynamic variable such as traffic. It can, however, be based on cost or expected traffic. The advantage of fixed routing is its simplicity. Its disadvantage is its lack of flexibility.

Another simple routing strategy is flooding routing. This technique requires no

network information, and works as follows. A packet is sent by a source node to every one of its neighbors. At each node, an incoming packet is retransmitted on all outgoing links except for the link that it arrived from. If the incoming packet is a duplicate it will not be forwarded. This ensure that the flooding process will eventually terminate. This technique is highly robust and can be used to send high-priority messages. The disadvantage of flooding is the total traffic load that it generates.

The next routing strategy is random routing. With random routing, a node selects only one outgoing path for retransmission of an incoming packet. The outgoing link is chosen at random, generally excluding the link on which the packet arrived. Random routing has the simplicity and robustness of flooding with less traffic load. To ensure that the routing process terminates, it may have to keep track of the duplicated packets. Also, a pure random routing scheme may fail to cover all nodes in the network. One way to solve this problem is to forward packets to the outlinks round-robinly. Random routing is not commonly used because of the unpredictable delay and high traffic caused by this scheme.

The final routing strategy is adaptive routing. Adaptive routing strategies adapt to measurable changing condition such as topology change and by other elements such as adaptive routing update time.

The fourth classification of routing algorithms is based on the transmission methods. There are two major classifications of transmission methods, store-and-forward and hot-potato forwarding.

Store-and-forward is probably the easiest to explain and the most widely implemented. When a packet or message arrives at a intermediate node on its path to the destination site, it waits in a queue for its turn to be transmitted on the next link in its path.

Hot-potato forwarding states that for all packets simultaneously arriving at a given node and not intended for reception at that node are immediately placed onto the outbound links leaving that node. If two or more packets contend for the same outgoing link to achieve minimum distance routing, then all but one will be misrouted to links which produce longer paths to the eventual destination.

The fifth classification of routing algorithms is based on the number of destinations a packet may be addressed to. We'll focus on this type of routing. Three types of routing schemes are Unicast Routing, Broadcast Routing, and Multicast Routing.

Unicast Routing is defined to be routing from a single source node to a single destination node in the network and will be discussed in Section 2.3. Broadcast Routing is routing from a single source node to all other nodes in the network and we will be described in Section 2.4. Multicast Routing is routing from a single source to a subset of the other nodes in the network. We will refer to this subset as a group. Multicast Routing also includes the case of routing from a single source to several groups which may overlap one another. We will discuss this type of routing in Section 2.5.

We must point out that the above three types of routing are not mutually exclusive; on the contrary, they often coexist in a network.

## 2.3 Unicast Routing

Unicast routing is the dominant activity in the network. It can be seen as a special case of multicast routing, but it is never treated that way because of its dominance. In this section we will describe the shortest-path based algorithms which are widely used in network routing[4].

In a network each link is assigned a cost, which can simply be one, the recent

traffic, or the queue length at the link. The cost for the same link from the two opposite directions can therefore be different. The goal of the shortest-path based routing is to choose a path which gives the minimum sum of link costs.

In unicast routing algorithms, each node usually has a routing table with entries of the form

| destination | distance | next hop | ... |
|---|---|---|---|

where next hop tells which link to take to reach the destination on the shortest path from the current node. When a packet arrives at a node, the shortest path is looked up from the table and the packet is sent to the "next hop" link. The routing table is updated when there is a topology change or a change in link cost. Link cost change is possible only when the link cost reflects the traffic. In this case, a cost change message is nothing but a traffic report, which is broadcast periodically.

Several shortest path algorithms are important to us. We describe them precisely since we will implement some of them for our study.

### 2.3.1 Bellman-Ford Algorithm

The first algorithm we wish to look at is the Bellman-Ford Algorithm [4, 13]. In this algorithm you first find the shortest paths from a given source node subject to the constraint that the paths contain at most one link; then find the shortest paths with a constraint that paths can be at most two links; and so on. This algorithms proceeds in stages and is formally described below. First we must define a few terms that will be used.

$s$ = source node

$$
\begin{aligned}
d_{ij} &= \begin{cases} 0 & \text{if } i = j; \\ \infty & \text{if the two nodes are not directly connected;} \\ < \infty & \text{if the two nodes are directly connected.} \end{cases} \\
h &= \text{max. no. of links in a path at the current stage} \\
D_n^{(h)} &= \text{cost of the shortest path from } s \text{ to } n \text{ with } \leq h \text{ links.}
\end{aligned}
$$

The algorithms has two steps; the first step is just initialization and the second step is repeated until none of the costs change:

1. Initialize:

   $D_n^{(0)} = \infty$, for all $n \neq s$

   $D_s^{(h)} = 0$, for all $h$

2. For each successive $h \geq 0$:

   $D_n^{(h+1)} = \min_j \{D_j^{(h)} + d_{jn}\}$

The path from $s$ to $i$ terminates with the link from $j$ to $i$.

For the iteration of step 2 with $h = k$, and for each destination node $n$, the algorithm compares potential paths from $s$ to $n$ of length $k+1$ with the path that existed at the end of the previous iteration. If the previous shortest path has less cost, then that path is retained. Otherwise, a new path with length $k+1$ is defined from $s$ to $n$; this path consists of a hop from node $j$ to node $n$. In this case, the path from $s$ to $j$ that is used is the $k$-hop path for $j$ defined in the previous iteration.

### 2.3.2 Dijkstra's Algorithm

The second algorithm is known as the Dijkstra's algorithm[4, 13]. You first find the shortest paths from a given source node to all other nodes by developing paths in order of increasing path length. The algorithm also proceeds in stages. By the $k$-th stage, the shortest paths to the $k$ nodes closest to (least cost away from) the source

node have been determined; these nodes are in a set $M$. At stage $k+1$ , the node not in $M$ that has the shortest path from the source node is added to $M$. As each node is added to $M$, its path from the source is defined. Before we can formally describe the algorithm we must define some notation.

$$
\begin{aligned}
N &= \text{set of nodes in the network} \\
s &= \text{source node} \\
M &= \text{set of nodes so far incorporated by the algorithm} \\
d_{ij} &= \begin{cases} 0 & \text{if } i = j; \\ \infty & \text{if the two nodes are not directly connected;} \\ < \infty & \text{if the two nodes are directly connected.} \end{cases} \\
D_n &= \text{cost of the shortest path from } s \text{ to } n, \text{ currently known to the algorithm}
\end{aligned}
$$

The algorithm has three steps. The first step again is initialization and the second and third steps are repeated until $M = N$. That is, steps 2 and 3 are repeated until final paths have been assigned to all nodes in the network.

1. Initialize:

   $M = \{s\}$

   $D_n = d_{sn}$ for $n \neq s$

2. Find the neighboring node not in M that has the least-cost path from node $s$ and incorporate that node into $M$:

   Find $w \notin M$ such that $D_w = \min D_j,\ j \notin M$

   Add $w$ to $M$

3. Update least-cost paths:

   $D_n = \min\{D_n, D_w + d_{wn}\}$ for all $n \notin M$

   If the latter term is the minimum, the path from $s$ to $n$ is now the path from $s$ to $w$ concatenated with the link from $w$ to $n$.

### 2.3.3 Floyd-Warshall Algorithm

The third main unicast algorithm is the Floyd-Warshall algorithm[4]. This algorithm finds the shortest paths between all pairs of nodes. It also assumes that there are no negative length cycles. For clarity, suppose we want to find the shortest paths from node 1 to all other nodes.

Let $D_{ij}^{(n)}$ be the shortest path length from node $i$ to $j$ when only nodes 1 to $n$ can be on the intermediate path. The algorithm goes as below.

1. $D_{ij}^{(0)} = d_{ij}$
2. $D_{ij}^{(n+1)} = \min\left\{D_{ij}^{(n)},\ D_{i,(n+1)}^{(n)} + D_{(n+1),j}^{(n)}\right\}$

Step 2 is repeated until $n$ reaches $N$, the number of nodes in the network.

### 2.3.4 Distributed Bellman-Ford Algorithm

The fourth main algorithm is the Distributed Bellman-Ford algorithm [4]. We will describe the basic form of the algorithm and then some improvements for solving problems that the basic algorithm cannot handle.

First let us look at the basic algorithm. Consider a routing algorithm that routes each packet along a shortest path from the packet's source to its destination. The link cost may change either due to link failures and repairs, or due to changing traffic conditions in the network. It is therefore necessary to update shortest paths in response to these changes. Algorithms which do this are called distributed algorithms. The main idea is to compute the shortest distances from every node to every other node by means of a distributed version of the Bellman-Ford algorithm.

Assume that each link $(i, j)$ has a positive length $d_{ij}$, the network is strongly connected (meaning that all nodes in the network are mutually reachable), and if

$(i, j)$ is a link, then $(j, i)$ is also a link. This algorithm only requires that each node know the length of its outgoing links and the identity of every node in the network.

The algorithm calculates the shortest distance $D_i$ from each node $i$ to, for example, node 1. Let $N(i)$ denote the neighbors of node $i$, the Bellman-Ford algorithm

$$D_i^{(h+1)} = \min_{j \in N(i)} [d_{ij} + D_j^{(h)}], \quad i \neq 1$$

suggests that only the information on the neighbors of $i$ is necessary for calculation to carry on. The distributed asynchronous Bellman-Ford algorithm is as follows:

At each time $t$, a node $i \neq 1$ knows $D_j^i(t)$, the latest distance estimate of node $j \in N(i)$, and $D_i(t)$, the latest distance estimate of node $i$. Here distance means the distance to node 1. Initially these distances are set to be arbitrary positive values (usually quite large.) Each node updates its information only at some times $t_0, t_1, \ldots t_n$. At any time $t_k$, one of the three events occurs:

(1) Node $i$ updates $D_i(t)$ by $D_i(t) = \min_{j \in N(i)}[d_{ij} + D_j^i(t)]$ and all $D_j^i(t)$ unchanged.

(2) Node $i$ receives from some $j \in N(i)$ new $D_j$ and update $D_j^i$.

(3) Node $i$ is idle, no estimates held by node $i$ are changed.

All nodes performs the same calculations in parallel and asynchronously, and all $D_i(t)$ converge to the correct shortest distances within finite time.

Now let us look at some improvements on the distributed Bellman-Ford algorithm, most of which are based upon versions of distributed shortest path algorithms.

First, Humblet[7] proposed an adaptive distributed shortest path algorithm which does not suffer from the "routing table looping" behavior associated with the Bellman-Ford distributed shortest path algorithm. For the basic Bellman-Ford distributed algorithm, link failures will cause the looping problem, i.e., some nodes will be updating

their routing table endlessly, incrementing the distance entry forever (this is known as *counting*), and messages will cycle back and forth among involved nodes. In other words, the algorithm will not converge. In [7], the problem was solved by setting an upper-bound on the maximum length that a link can achieve, and more importantly, by maintaining at each node a shortest path tree that is rooted at that node.

Next Awerbuch and Gopal[2] proposed an approximate distributed Bellman-Ford algorithm which has a polynomial message complexity. This algorithm is an improvement upon the basic algorithm which suffers from exponential message complexity in some scenarios. The only difference between the approximate algorithm and the basic algorithm is in the adopting rule. In the basic algorithm, the adopting rule is: Suppose node $x$, with a label $a(x)$, receives $a'(x)$ from $z$. If $a'(x) < a(x)$, then $x$ adopts $a'(x)$ and sets the value of parent($x$) to $z$. In the approximate algorithm, the adopting rule is: Suppose node $x$, with a label $b(x)$, receives $b'(x)$ from $z$. If $b'(x) < b(x)/k$, then $x$ adopts $b'(x)$ and sets the value of parent($x$) to $z$. Where the label $a(x)$ and $b(x)$ are the current known shortest distance from node $s$ to node $x$, and parent($x$) contains the identity of its previous node on the current known shortest path from $s$ to $x$. Note that the two algorithms become identical when $k = 1$. For the approximate algorithm $k$ is greater than or equal to 1.

## 2.4 Broadcast Routing

There are many cases in which a message needs to be broadcast to all nodes in the network. The most important use is to transfer control information when links are removed or the topology is changed. Flooding and Spanning tree are two major algorithms for broadcasting.

Flooding is a simple algorithm for broadcasting. A node broadcasts an update

message to all nodes by sending the message to its neighbors, which in turn send the message to their neighbors other than the one from which the message comes.

Given a connected graph $G = (V, E)$ with $V$ the vertex set and $E$ the edge set, and a weight $W_{ij}$ for each edge $(i, j)$, the problem is to construct a spanning tree with minimum sum of edge weights. Such a tree is called a Minimum Weight Spanning Tree (MST). An MST may be useful for disseminating messages destined for every node in a data network. An edge weight represents the communication cost of a message along the edge in either direction, and the total spanning tree weight represents the cost of broadcasting a message to all nodes along the spanning tree.

Broadcast algorithms often need a sequence number for each packet to avoid duplicates received at a node; and sometimes a periodical broadcast is used to guarantee that up-to-data information will become available within some fixed time. Spinelli and Gallager[12] proposed an algorithm for broadcasting network topology change, which does not require a sequence number and does not require a periodical broadcast. The main idea of the algorithm is to keep each network node informed of the entire network topology when the topology occasionally changes over time. This algorithm is an event driven algorithm, i.e., it sends messages only in response to receiving other messages, or in response to topological changes in adjacent links.

Gossiping[9] is another way to broadcast. In Gossiping, each node exchanges information with exactly one of its neighbors at prescribed time instants. At different times, a node may exchange information with different neighbors. In doing so, every node is eventually informed of the state of all other nodes after what is defined as the cyclic gossiping time. To minimize the gossiping time and the number of links required for information exchange, the gossiping procedure is first mapped into a graph coloring problem, in which adjacent edges are colored with different colors; for

a graph with $k$ distinct colors, at time $t$, $t = 0, 1, 2, \ldots$, each pair of vertices which are joined by an edge colored $i = t(\mathrm{mod}\ k)$ exchange all of their information at that time. Some limited results are given in [9].

## 2.5 Multicast Routing

There is a large potential demand for multicast transmission such as e-mail, where a message has to be sent to more than one address; distributed database management, where a query may be sent to several sites that may have the data to answer it; and tele-conferencing. Deploying multicast routing algorithms in a very large network is a complex task. In this section, we will first review several basic multicast routing algorithms, then we will introduce some other algorithms which describe different aspects of multicast routing.

### 2.5.1 Flooding

Flooding is the simplest algorithm and was described in Section 2.4. A node in the network will receive a packet that was sent to a multicast destination. It will first try to ascertain whether this is the first reception of this particular packet or whether this is a duplicate. If it is a first transmission, it will transmit a copy of the packet on all neighbors, ensuring that the packet eventually reaches all network destinations. The key problem to any flooding implementation is indeed the test for "first reception." One of the methods for solving this problem is by looking in the link state database, if the received link state update is newer than the local value, the database is updated and the packet is forwarded, else it is ignored.

### 2.5.2 Spanning Trees

A more efficient solution than flooding is the "spanning tree" technique. One will simply build up an "overlay" network by marking source links as "part of the tree" and other links as "unused." The set of selected links forms a loopless graph or a tree which spans to all nodes in the network. Once a spanning tree has been built, the propagation of multicast packets becomes very simple. When a multicast packet is received it is immediately forwarded to all outgoing links that are part of the tree except the incoming link. Since the tree is loopless, we guarantee that the packet will reach all destinations. But the basic spanning tree has two drawbacks: it does not take into account group membership and it concentrates all the traffic into a small subset of the network links. This results in an algorithm that is very easy to implement, but is very inefficient.

### 2.5.3 Reverse-Path Forwarding (RPF)

The RPF algorithm takes advantage of a routing table to "orient" the network and to compute an implicit spanning tree for each network source. In its simplest form, the RPF algorithm works as follows:

1. When a multicast packet is received, note source $S$, and the incoming link $I$ (which can be thought of as an adjacent node.)

2. If $I$ belongs to the shortest path to $S$, forward the packet to all links except $I$.

3. If the test in step 2 is false, refuse the packet.

The advantage of this simple form is that RPF does not require any more resources than the normal "unicast" routing table; if we know how to compute the route toward $S$, then we can safely process multicast packets received from $S$.

One obvious drawback of RPF is it only works when hop-distance is used, or when the distances in opposite directions are equal. The problem with the simple form of RPF is that group membership is not taken into account when building the tree and packets are simply flooded to the whole network.

### 2.5.4 RPF and Prunes

The "pruning" idea is to complete the basic RPF by memorizing the group membership and to forward a packet only if there is a group member down the tree. This will result in minimal trees but requires very dynamic updates. When a multicast transmission starts from source $S$, the first packet is propagated to all the network nodes. This is the flooding. The leaf nodes will all receive the first multicast packet. If there is a group member attached to the leaf node, everything is fine, but if there is no group member the node does not want to receive further packets. It will thus send back a "prune message" to the node that sent it this packet: don't send further packets from source $S$ to group $G$ on this interface $I$.

The intermediate nodes that receive the prunes memorize them. If they have received a prune message through all the interfaces on which they forwarded the first packet and if there is no local recipient, they have no need for receiving from this source further packets for that group. They will forward the prune command up the RPF tree. Eventually, the tree will include only the branches that lead to active recipients.

There are two obvious drawbacks in the flood and prune algorithm:

1. The first packet is flooded to the whole network.

2. The nodes must keep states per group and source.

### 2.5.5 Steiner Trees

Given a connected, weighted graph $G = (V, E)$, where $V$ is the vertex set and $E$ the edge set, also given a subset $M$ of $V$, we can find more than one subgraph $S = (V', E')$ of $G$ such that $M \subseteq V'$ and there is a path in $S$ between every pair of nodes in $M$. $S$ is defined to be a Steiner Graph. The Steiner Tree problem in a graph is to find a Steiner Graph $T$ such that the sum of edge weights in $T$ is the minimum.

If we forward a packet along a Steiner Tree that connects the multicast destination nodes and the source node, the communication cost will be the minimum. This will eventually help reduce the packet delay and increase the throughput.

Since the Steiner Tree problem is NP-complete [8], no polynomial time algorithms for this problem exist. Many heuristics have been developed, and Table 2.1 [8] gives a listing of existing heuristic solutions to the Steiner Tree problem in a graph.

### 2.5.6 Flow Model and Optimal Routing

The shortest path algorithm is a greedy algorithm because it puts all the traffic between a source-destination pair on the single shortest path. Optimal routing tries to build a precise model for overall optimality and solve a nonlinear optimization problem, i.e., the traffic is distributed among all possible paths between the pair. In this subsection, we will first describe the model for unicasting and then introduce the model for multicasting.

**Unicasting Flow Model.** Let $F_{ij}$ be the traffic arrival rate at link $(i, j)$. $F_{ij}$ is called the flow of link $(i, j)$.

| Method | Submethod | Acronym | Complexity |
|---|---|---|---|
| Path heuristics | shortest paths | SPH | $O(n(e + v \log v))$ |
| | repetitive | SPH-N | $O(n^2v^2)$ |
| | | SPH-V | $O(nv^3)$ |
| | | SPH-zN | $O(n^2v^2)$ |
| | | SPH-NN | $O(n^3v^2)$ |
| | with origin | SPOH | $O(e + v \log v)$ |
| | Kruskal | KBH | $O(nv^2)$ |
| | Y | YH | $O(n^2e \log v)$ |
| Tree heuristics | MST | MSTH | $O(e + v \log v)$ |
| | greedy | GTH | $O((v - n)^2v + n^2)$ |
| | distance network | DNH | $O(e + v \log v)$ |
| | repetitive | RDNH | $O((v - n)^q n^2 + v^3)$ |
| | multiple | MDNH | $O(ev + v^2 \log v)$ |
| | simulated annealing | SAH | |
| | hill climing | HCH | |
| Vertex heuristics | average distance | ADH | $O(v^3)$ |
| | fast | FADH | $O(v^3)$ |
| | restricted | RADH | $O(n^2e \log v)$ |
| | median | MH | $O((v - n)v^2)$ |
| | antimedian | AMH | $O((v - n)v^2)$ |
| | upgrading | UH | $O(v^3 + vn^3)$ |
| | modified upgrading | MDH | |
| Contraction | | CH | $O(v^3)$ |
| Dual Accent | | DAH | |
| Set Covering | | SCH | |

Table 2.1: Steiner Tree Heuristics [8]

Let the cost function for optimization be

$$D = \sum_{(i,j)} D_{ij}(F_{ij}) \qquad D_{ij}(F_{ij}) = \frac{F_{ij}}{C_{ij} - F_{ij}} + d_{ij}F_{ij}$$

where $C_{ij}$ is the link capacity of $(i, j)$.

Denote $\omega(k, l)$ to be a source-destination pair (SD pair) from node $k$ to node $l$, let $r_\omega$ be the input arrival rate entering $k$ and destined for $l$. Let $P_\omega$ be the set of paths connecting $k$ and $l$, and $x_p$ be the flow of a path $p$. Then for link $(i, j)$

$$F_{ij} = \sum_{p \text{ containing } (i,j)} x_p$$

The optimization problem is defined as

$$\text{minimize} \quad \sum_{i,j} D_{ij}(F_{ij})$$

$$\text{subject to} \quad \sum_{p \in P_\omega} x_p = r_\omega, \quad x_p \geq 0$$

The solution is the specification of traffic flow to each path. The major problem is that an analytical solution to the problem does not exist in most cases. Several iterative algorithms had been developed and are covered very thoroughly in [4].

**Multicasting Flow Model.** Stassinoponlos and Kazartzakis give a model for the multidestination optimal dynamic routing problem in [14] and [15]. Their algorithms are based on the maximal flow-minimal cut theorem[11] and solve the problem by an iterative link-by-link optimization method. The following is the model:

Let $N$ be the set of nodes, $M \subseteq N$ is the set of destination nodes, and $L$ is the set of directed links $(p, q)$, $p, q \in N$. Link $(p, q)$ has the capacity $C_{pq} > 0$. If $i = (j, k)$ is an source-destination pair with $j \in N$, $k \in M - \{j\}$, we let $x_i$ and $r_i$ denote the initial traffic and external arrival rate at node $j$ destined for node $k$. The set $I$ comprises all source-destination pairs $i$ for which $x_i$ or $r_i$ is nonzero. Each link

$(p,q) \in L$ can serve traffic belonging only to some source-destination pairs in general. We denote by $I_{pq}$ to be the set of all pairs in $I$ for which there is an path from the source to the destination node passing through link $(p,q)$.

For each $i \in I$, define a nonnegative capacity allocation function $\xi_i(\cdot,\cdot)$ where domain is the set of links $L$, i.e.,

$$\xi_i(\cdot,\cdot): \ L \to R$$

$$(p,q) \to \xi_i(p,q)$$

the capacity allocated for each source-destination pair over the whole network is given by the family $\{\xi_i(\cdot,\cdot)\}_{i\in I}$ which must satisfy

$$\sum_{i\in I_{pq}} \xi_i(p,q) = C_{pq}, \ \ \forall (p,q) \in L$$

$$\xi_i(p,q) = 0, \ \ i \notin I_{pq}$$

For each $i \in I$ we say that the subset $A_i \subset N$ is a cut if $A_i$ contains the source and $N - A_i$ contains the destination node.

We let $\mathbf{A}_i$ be the family of all possible cuts $A_i$. For each $i \in I$ the external arrival rate $r_i$ can be served if and only if

$$\sum_{(p,q)\in A_i} \xi_i(p,q) - r_i \geq 0, \forall A_i \in \mathbf{A}_i$$

Our problem is the minimization of

$$\max_{i\in I}\left\{\max_{A_i\in\mathbf{A}_i}\left\{x_i\left(\sum_{(p,q)\in A_i}\xi_i(p,q)-r_i\right)^{-1}\right\}\right\}$$

and to maximize

$$\min_{i\in I}\left\{\min_{A_i\in\mathbf{A}_i}\left\{x_i^{-1}\sum_{(p,q)\in A_i}\xi_i(p,q)-r_i\right\}\right\}$$

The determination of the flow on a single link requires an $O(|n|^6)$ time complexity per iteration for a network of $n$ nodes, based on this model and the solution given in [15].

### 2.5.7 Related Algorithms for Multicast

In this subsection, we briefly introduce several earlier algorithms and then pay more attention to the link state algorithms by Deering[6] which are relatively new and have been experimented with on the Internet. First we will give some detailed description of Deering's algorithms. This will help understand the assumptions made in our proposed algorithm presented in Chapter 3. The details are also necessary because we will simulate one of Deering's algorithms and compare it with our algorithm in Chapter 4.

Aguilar[1] proposed a algorithm for internet multicast routing which makes use of the IP (Internet Protocol) header for the destination address. The maximum size of an IP header is 60 bytes, of which 20 are used for identification and control information, including 4 for an internet destination address. The other 40 bytes can be used for options. These 40 bytes can be used for the internet addresses of multiple destinations. Since each address uses 4 bytes, with 8 bytes for control information, eight destination addresses can be included; this together with the already existing IP destination gives us 9 possible addresses. The routing follows "shortest" paths and the algorithm places all its destination addresses in an Internet packet. During propagation, the packet is replicated at gateways and the address list is partitioned among the newly created packets, according to their next Internet stop. This scheme's advantage is that it can be done without additional protocol information and no changes to the subnetwork routing. It can be deployed in an internet regardless of the subnetworks transmission

mode and regardless of whether the subnets have or do not have multicast capacity. The drawback for this algorithm is that it limits the size of the multicast destinations to only 9.

Belkeir and Ahamad[3] proposed two heuristic algorithms in dynamic multicast which update the multicast tree incrementally as the membership changes and reduce the total bandwidth required for sending data and control messages. These two algorithms are Restricted Broadcast Algorithm (RBA) and Nearest-Insertion Algorithm (NIA).

Restricted Broadcast Algorithm (RB) is a broadcast-tree based algorithm. The idea is to construct a multicast tree for each multicast group as the sub-tree of a minimal cost broadcast spanning tree. This type of multicast tree can be computed in polynomial time. The broadcast tree is constructed once and used as the basic structure to support message delivery for all multicast groups. Furthermore, when the membership of the group changes, the algorithm is executed by a node to update the multicast tree incrementally.

In Nearest-Insertion Algorithm (NI), when a multicast tree needs to be extended (when a new member joins the group), all links in the broadcast tree should be considered as potential candidates to be added to the multicast tree. The algorithm considers all nodes and links, and chooses the ones that are on the path to a participant node in the group which is nearest to the new member.

The advantage of the RB and NI algorithm is that they have no disruption of the multicast tree since the trees are incrementally updated without reconstruction. The problem is the broadcast tree is fixed, not dynamic, which does not respond to the change of traffic in the network, and the maintenance of a separate multicast tree is still required for each source-group pair.

Early in 1983, Bharath-Kumar and Jaffe[5] studied several heuristic algorithms for multicast routing and evaluated their performance. These algorithms are Minimum Spanning Tree (MST), Minimum Destination Cost (MINDC), Nearby Heuristic 1 (NH1), Nearby Heuristic 2 (NH2), and Nearest Neighbor Heuristic (NN).

A minimum spanning tree for the "induced graph" (also called distance graph[8]), $I(G)$, of the given graph $G$ is constructed and used for multicast. The algorithm is as follows:

Given network graph $G$, source node $s$, and destination node set $D$, with $|D| = m$, we define $I(G)$ to be the complete graph on $m + 1$ nodes (one node s and one for each $d \in D$) with cost $sd(i, j)$ associated with edge $(i, j)$ where $sd(i, j)$ is the shortest distance from $i$ to $j$ in $G$. The algorithm executes following steps:

1) Determine $sd(i, j)$ for every $i, j \in \{s\} \cup D$.

2) Construct $I(G)$.

3) Determine the Minimal Spanning Tree $T$ for $I(G)$, and route according to $T$ as follows:

a) For each neighbor of $s$ in $T$, $s$ sends a copy of the message. Node $s$ includes in each message the portion of $T$ rooted at that neighbor.

b) Each node receiving such a message sends out copies if it is not a leaf of the tree, using the same distribution algorithm as the source.

MINDC uses local information, which is defined to be the shortest path information from the source to all destinations. This algorithm is to choose the routing that minimizes DC, i.e., the message is sent from the source to each destination node on its shortest path routes.

Both NH1 and NH2 algorithms require nearby information defined as the shortest path tables of the neighbors of the source node as well. In NH1, the source node uses all information available to its neighbors as follows. Assume that $s$ were to bifurcate to neighbors $n_1$, $n_2$, and $n_3$. Assume further that $s$ routes to a set of destinations $D_i$ through $n_i$ with $D_i \cap D_j = \emptyset$ for $i \neq j$ and $\cup_i D_i = D$. Then the local cost of the routing is the sum of the costs of the three links, connecting $s$ to $n_1$, $n_2$, and $n_3$. The estimated global cost is $\sum_{(i=1)}^{3}$(sum of costs from $n_i$ to each node in $D_i$).

Algorithm NH1 routes from $s$ to its neighbors by choosing the routing that has the smallest total estimated cost. Once this is done, intermediate nodes choose their next nodes in the same way. One of major problem with NH1 is that it has to consider all $2^m$ partitions of $D$, giving an exponential time complexity.

In NH2, the algorithm investigates the possibility of using the neighbors one by one. The destination $d$ closest to $s$ is chosen, and the neighbor $n_1$ which lies on the shortest path from $s$ is investigated. For each $d' \in D$, if the least cost path from $n_1$ to $d'$ is smaller than that from $s$ to $d'$, $d'$ is assigned to $n_1$. After every $d' \in D$ is examined, if any are left unassigned, the one that is closest is chosen next. Let $n_2$ be the neighbor on the shortest path from $s$ to it. We now attempt to assign the remaining destinations to $n_2$ as above. The procedure continues until all destinations are assigned to neighbors of $s$. A copy of the message to be routed is sent to these neighbors. Then, the analogous procedure is used at each of these nodes.

In NN algorithm, each node $I$ that has responsibility to route to a certain destination sends the message on the shortest path to the destination nearest to $I$. Also, responsibility for all other destinations are transferred to that destination.

Deering[6] implemented, among others, two algorithms based on link-state (LS) routing: LS Truncated-Broadcast Routing and LS Multicast Routing. As mentioned

previously, we will give more details for these two algorithms. We first introduce a protocol called "Host Membership Protocol" which is the basis of the multicast in [6], then describe basic LS routing and the two algorithms introduced by Deering.

**The Host Membership Protocol.** This protocol provides information for routers so that routers can learn which host groups are present on their directly-attached subnetworks. The protocol enables every router attached to a subnetwork to learn which host groups are present on the subnetwork, when the router starts up; to detect when a new host group appears on the subnetwork; and to detect when a host group disappears from the subnetwork.

The Host Membership Protocol is based on a simple query-response scheme. On each subnetwork, one of the attached routers is elected as the interrogator. The interrogator periodically multicasts a host membership query packet to the subnetwork, the query packet is sent to the all-hosts group, whose scope is local to the subnetwork, and to which all hosts belong.

For each group that is present on the subnetwork, one member host responds to the query by multicasting a host membership report packet. The report packet is addressed to the group that is being reported, and is sent with a maximum hop-limit of one. This results in delivery to all other members of that group on the same subnetwork, plus all attached routers. The report serves two purposes: it tells the routers that the group is present, and it tells the other members not to report the group. The routers conclude that a group is no longer present when no report is received for the group after a small number of queries. This protocol comprises two algorithms: the Host algorithm and the Router algorithm.

The host algorithm operates on abstract objects called "membership"; a mem-

bership $m$ has the following attributes:

| | |
|---|---|
| $m$.group | host group address |
| $m$.subnet | attached subnetwork identifier |
| $m$.timer | retransmission/delay timer |
| $m$.count | retransmission counter |

Whenever an upper-layer protocol invokes the JoinGroup operation, a new membership is created. This informs the routers of the group's presence, in case there were no previous members present on the subnetwork—the sooner the routers learn of the group, the sooner they are able to forward multicast packets destined to the group onto the subnetwork, thus minimizing join latency. When an upper-layer protocol invokes the LeaveGroup operation, the corresponding membership is deleted. When a host receives a host membership query, it does not respond immediately. Instead, for each group to which it belongs on the subnetwork and whose scope is wider than the subnetwork, it sets its membership's timer to a random delay value between zero and some maximum delay $T_d$; a separate random delay is chosen for each group. When a membership's timer expires, a corresponding host membership report is transmitted.

Whenever a host membership report is received, if the host belongs to the reported group, it stops its timer for that group and abstains from sending its own report.

The router algorithm operates on two types of object. A "group presence" object $p$ has the following attributes:

| | |
|---|---|
| $p$.group | host group address |
| $p$.subnet | attached subnetwork identifier |
| $p$.time-left | time left until group is assumed absent |

and an "attached subnetwork" object $k$ has the following attributes:

| | |
|---|---|
| $k$.interrogator | true if this router is interrogator for $k$ |
| $k$.timer | retransmission/query timer |
| $k$.count | retransmission counter |

The $k$.interrogator Boolean is set by an interrogator election algorithm.

When a router starts up, it immediately sends $N$ queries at intervals of $T_n$, so that it may quickly learn what groups are present without waiting for the interrogator to issue the next periodic query. After the initial $N$ transmissions, if the router has been elected interrogator, it proceeds to send periodic queries at intervals of $T_q$, until it fails or resigns as interrogator.

Whenever a router receives a host membership report from a particular group on a subnetwork, it resets the time-left for that group to a value $T_p$. If a group's time-left expires, the router concludes that the group is no longer present on the subnetwork and deletes its record.

**Link-State Routing.** Under the link-state routing algorithm, every router monitors the state of each of its incident links or subnetworks. The state of a subnetwork includes whether or not the subnetwork is operational, the set of neighboring routers reachable across the subnetwork, and in some cases, a measure of the traffic load on the subnetwork. Whenever the state of a subnetwork changes, the routers attached to that subnetwork broadcast the new state to every other router in the internetwork. The broadcast may be accomplished by a special-purpose, high-priority flooding protocol that ensures that every router quickly and reliably learns the new state. All routers know the state of all subnetworks and all routers, from which they can each determine the complete topology of the internetwork. Each router independently computes the shortest paths from itself to every possible destination, using Dijkstra's algorithm.

```
1. on change of designated router on subnetwork k:
     k.interrogator = k.designated
2. on change of topology:
     for each route r, r.parent = undefined
3. on receipt of multicast packet from s to g via subnetwork k:
     if ∃ route r such that r.destination = s
          if r.parent = undefined, <compute r.parent, r.child, r.leaf >
          if r.parent = k
               for each subnetwork j
                    if r.child[j] and (( not r.leaf[j]) or (∃ group presence p
                         such that p.group= g and p.subnet= j))
                              send copy of packet on subnetwork j
```

Table 2.2: LS Truncated Broadcast

**LS Truncated-Broadcast Routing.** Given the complete topological knowledge available in every LS router, any router can directly compute the shortest-path broadcast tree rooted at any particular source $s$, using Dijkstra's algorithm. The router can also identify from the tree computation, which of its child subnetworks are leaf subnetworks in the broadcast tree for $s$.

The LS truncated-broadcast algorithm is shown in table 2.2. This algorithm operates on "route" objects, which are the entries in the LS routing table; a route object $r$ has the following attributes of relevance to this algorithm:

| | |
|---|---|
| $r$.destination | internetwork destination address |
| $r$.parent | parent subnetwork, for broadcasts from destination |
| $r$.child[$k$] | child flags, per attached subnetwork $k$ |
| $r$.leaf[$k$] | leaf flags, per attached subnetwork $k$ |

The algorithm also operates on the "group presence" and "attached subnetwork" objects described earlier except that each attached subnetwork $k$ has one additional attribute:

$k$.designated: true if this router is LS designated router for subnetwork $k$.

In Table 2.2, each numbered item is called an event. There are three events in this algorithm, as described below.

**Event 1.** The election of a designated router to serve as a membership interrogator is required by the Host Membership Protocol.

**Event 2.** On any change of topology, the LS routing algorithm recomputes its entire routing table.

**Event 3.** Delivery of multicast packets by truncated-broadcast. The source of an incoming multicast packet is looked up in the routing table. If the route entry contains an undefined parent value, that indicates that this is the first multicast packet from that particular source since the last topology change; in that case, the parent, child, and leaf fields are computed. The router then checks that the packet arrived on the correct parent subnetwork for its source and, if so, forwards it on all child networks that are not leaf subnetworks or that have members of the destination group, which is the normal truncated-broadcast forwarding procedure. The determination of the parent, child, and leaf values for a route entry entails the execution of Dijkstra's algorithm to compute the shortest-path broadcast tree rooted at the source.

The algorithm works by starting with a node for the root, and iteratively adding all other nodes to the tree, nearest nodes first. When the router adds a descendent subnetwork node to its own node, it sets the child and leaf flags for that subnetwork, when the router adds a descendent router node to one of those child subnetwork nodes, it clears the leaf flag for that subnetwork. When all the nodes have been added to the tree, the child and leaf information is complete.

**LS Multicasting Routing.** This algorithm is an extension of the LS truncated-broadcast algorithm. The main idea behind this algorithm is to consider the set of

```
1. on change of designated router on subnetwork k:
       k.interrogator = k.designated
2. on appearance of disappearance of group g on subnetwork k:
       if k.designated
             flood link-state update, reporting change of g on k
3. on generation or receipt of link state update, reporting change of g:
       for each cache record c such that c.group = g
             discard c
4. on change of topology:
       for each cache record c
             discard c
5. on receipt of multicast packet from source s to group g via subnetwork k,
       with remaining hop limit hl:
       if ∃ no cache record c such that c.source = s and c.group = g
             create cache record c; c.source = s; c.group = g
             <compute c.parent and c.min hops attributes>
             if c.parent = k
                   hl = hl - 1
                   for each subnetwork j
                         if hl ≥ min-hops[j]
                               send copy of packet on subnetwork j
```

Table 2.3: LS Multicast

groups present on a subnetwork to be part of the state of that subnetwork. Whenever the set of groups changes, as with any other state change, the designated router for that subnetwork floods a link-state update message to all other routers, reporting the change. All routers end up with complete and consistent knowledge of which groups are present on which subnetworks.

The LS multicast algorithm is shown in Table 2.3. The algorithm operates on "cache record" objects that hold the information required to forward multicast packets. A cache record $c$ has the following attributes:

| | |
|---|---|
| $c$.source | source of multicast packet |
| $c$.group | destination of multicast packets |
| $c$.parent | parent subnetwork, for multicast from $c$.source |
| $c$.min-hops[$k$] | min hops to $c$.group, per attached subnetwork $k$ |

The algorithm also operates on "attached subnetwork" objects as defined for the router part of the Host Membership Protocol, except that each attached subnetwork $k$ has one additional attribute:

$k$.designated: true if this router is LS designated router for subnetwork $k$

Similar to LS Truncated algorithm, this algorithm defines five events as follows.

**Event 1.** The election of a designated router serves as an membership interrogator, as required by the Host Membership Protocol.

**Event 2.** Whenever the procedure for the Host Membership Protocol detects the appearance of a new group on an attached subnetwork or the disappearance of a previously present group, if the router is the LS designated router for that subnetwork, it invokes the LS protocol procedures to flood a link-state update, reporting the change.

**Event 3.** When a link-state update for a group is generated by the procedure for event 2 or received from another router, all cache records referring to that group are discarded.

**Event 4.** Similar to event 3, whenever there is a topology change, all cache records are discarded.

**Event 5.** Multicast packet from source $s$ to group $g$. Look up the cache record for that (source, group) pair; if there is no such record, create one and compute its parent and min-hops attributes. Then if the cache records shows that the packet arrived via the correct parent subnetwork, decrement the hop limit and forward a packet to each child subnetwork which leads to a group member within the remaining hop limit. A child subnetwork $k$ is indicated by a finite value of min-hops[$k$] which is the distance to the nearest group member via $k$ in the delivery tree.

# Chapter 3

# The Proposed Algorithm

In this chapter we propose an algorithm for datagram multicast routing which we call *Minimum Average Distance* algorithm. We first give a brief description of the main issues which we need to address. Then we give a formal description of our algorithm and a simple example to explain the algorithm.

## 3.1 Main Issues in Internetwork Multicast

For clarity, we will restate here some of the concepts previously introduced. In internetworks, multicast packets are delivered to each member of a multicast group. A multicast group is a set of destination nodes to which a message needs to be delivered. Each destination node is a *member* of the group. For each source there can be many different destination groups. For up-layer users, they need only know the group address, not the group member addresses. This is the "unknown-destination" delivery feature. It is the router that takes the burden of addressing the message to group members.

We assume that the subnetwork supports multicast (most local-area networks have the ability to address and deliver a packet to a set of destinations.) Based on this assumption, we omit the detail of a subnetwork and treat each router as a node.

In this context, we use node and router interchangeably.

A multicast group may be of arbitrary size, the membership of a multicast group may change dynamically, and the members of a group may be located anywhere in the internetwork.

We use the datagram model (or connectionless model) for our multicasting. Each packet doesn't carry the multiple destination addresses, each router maintains a routing table for all nodes, and each routing table entry records the shortest path and the first-hop node for each destination. Each router can determine the complete topology of the internetwork and can independently compute the shortest paths from itself to every possible destination. We also assume that routers maintain the group-member information which ensure that every router knows the membership of a group. When a packet arrives at a router, the router can independently decide whether there is any member in its attached subnetwork to which the packet will be forwarded.

We assign each link a state which includes the length of the link, the membership of a group, and whether the link has been disconnected or is reconnected. Whenever a link state changes, we assume that there is a designated router which broadcasts the new state to every other router in the internetwork and that all routers will quickly and reliably update their state database corresponding to the new state.

For a dynamic multicast routing algorithm, some basic requirements should be considered:

1) The algorithm should be computationally efficient: an algorithm is hardly useful if it requires more than a linear time to determine to which link a packet is to be sent. A constant time complexity is preferred.

2) The algorithm should be scalable: it should not heavily dependent on network size, group size, etc.

3) The algorithm should be compatible with unicast routing: it should not require excessive information compared with unicast routing, and should not change dramatically from unicast environment.

4) The algorithm should seek a minimization of delay and traffic: these two goals may or may not conflict with each other. A minimum cost tree may require packets to reach some member host along a non-shortest path. But when the minimum cost tree is utilized, the entire traffic is reduced, which may help reduce the delay.

Another important issue is that of the low-delay and low-cost routing. Low-delay routing is to minimize the path length from the source to every destination node. It usually requires the message to be sent to every destination along the shortest path or its equivalent as in unicast routing. Each link is assigned a length which can simply be 1 or the recent traffic rate or queue length on the link. The link length from two opposite directions can be different. Low-cost routing is to minimize the total traffic, which is equivalent to the total length of the paths in a delivery tree. In graph-theoretic terms, a minimum delay routing corresponds to a shortest-path tree, while a minimum-cost routing corresponds to a Minimum Steiner Tree in a graph. Finding a Minimum Steiner Tree is an NP-complete problem. There exist some heuristic algorithms for computing low-cost multicast tree, as listed in Chapter 2, but they either suffer from high computational cost or are not suitable for the datagram environment.

Our algorithm focuses on low-cost multicasting. This algorithm's computational complexity is only linear in the group size. In chapter 4, the simulation result shows that the performance of this algorithm is sufficiently satisfactory.

The main idea behind this algorithm is that we use the *minimum average dis-*

*tance* method to select the forwarding links. When a packet arrives at a node, the algorithm computes the average distance for each neighbor of the current node, where the *average distance* is the average distance from the neighbor node to all nodes in its *responsible destination set*. Then we select the neighbor node which has the minimum average distance to forward the packet. This procedure continues until all destination members are processed. A simple example presented in Section 3.3 will demonstrate the operation of the algorithm. Another important feature of this algorithm is that we do not construct the delivery tree at all. It uses the routing tables of neighbor nodes to compute the next node to which the packet will be forwarded, this saves memory and computation time.

## 3.2 Algorithm Description

First, we give the definitions for the following variables which will be used in the algorithm description.

$G$: The destination set, that is, a group.

$G_i$: Node $i$'s responsible destination set.

$N_i$: Node $i$'s open neighborhood set, i.e., all of node $i$'s neighbor nodes except for node $i$ itself.

$D_j$: Neighbor $j$'s responsible destination set.

$L_j$: Average length from the neighbor node $j$ to its responsible destination set.

$A_i$: The assigned neighbor set.

$d_{ij}$: The shortest path distance from node $i$ to node $j$.

We also define the routing table at each node with entries as:

| destination | shortest distance | next hop |
|---|---|---|

where "next hop" is used in the shortest path algorithm for both unicasting and

multicasting. Next hop tells which link to take to reach the destination on the shortest path from the current node. We do not use next hop directly, but compute the next hop based on our algorithm. Following is the description of the algorithm:

1. A packet $p$ addressed to group $G$ is now at node $i$:

   if $i$ is the source node then $p$.radius $= \infty$; $A_i = \emptyset$;

   else $p$.radius $= R$, where $R$ is a radius which is carried with $p$.

   $A_i = \{k\}$, where $p$ is from neighbor $k$ of node $i$.

   if $p$.radius $= 0$ then stop.

2. Compute node $i$'s responsible set

$$G_i = \{g | g \in G, \text{ and } d_{ig} \leq p.radius, \text{ and } g \neq i\} \tag{3.1}$$

3. For each $j \in N_i$ and $j \notin A_i$, compute $j$'s destination set

$$D_j = \{g | g \in G_i, \text{ and } d_{jg} < d_{ig}\} \tag{3.2}$$

   and the average length

$$L_j = (d_{ij} + \sum_{g \in D_j} d_{jg} - \hat{L}_j)/|D_j| \tag{3.3}$$

   where $\hat{L}_j$ is the sum of the overlapped next-hop link, defined as

$$\hat{L}_j = \sum_{n_{jk} > 1} (n_{jk} - 1) l_{jk}$$

   where $n_{jk}$ is the number of members in $D_j$ that will be sent via link $k$ from node $j$, and $l_{jk}$ is the length of the link.

   if (all $D_j = \emptyset$) or ($N_i = A_i$), stop.

4. Let $J$ be the subscript such that

$$L_J = \min_{j \in N_i, j \notin A_i} L_j$$

and

$$R = \max_{g \in D_J} d_{Jg} \tag{3.4}$$

send a packet with radius $R$ to node $J$.

5. $G_i = G_i - D_J \quad A_i = A_i \cup \{J\}$

if $G_i = \emptyset$, stop

else loop back to step 3.

When a packet arrives at a node $i$ and is addressed to group $G$, we give a condition to limit the size of the destination set; that is, to decide to which members of group $G$ should be sent the packet from the current node $i$. We use a variable $p$.radius as the condition. The value of $p$.radius is the path distance between the current node and the remotest destination member. That is, the current node $i$ only sends packets to those members (in $G_i$) whose shortest path distance from node $i$ to them is less than or equal to $p$.radius (step 2). If node $i$ is the source node, we set $p$.radius $= \infty$; this means that at the beginning, we consider all members of the group $G$ as the destination set. If node $i$ is not the source node, the packet must have come from its parent node; the packet then carries a radius with it which was computed at the parent node. $A_i$ at the source node is $\emptyset$. When a packet arrives at node $i$ from its parent, $A_i$ initially contains the parent node. $A_i$ will be increased by adding node $i$'s neighbors one by one until it contains all of the neighbors or other terminating conditions are satisfied (step 3).

For each neighbor $j \in N_i$ and $j \notin A_i$, we compute $j$'s responsible destination set $D_j$ by (3.2) and each $j$'s average length $L_j$ by (3.3). We then send the packet together with a radius to the selected neighbor node which has the minimum average path length. The radius is computed by (3.4); it is the path length between the

selected neighbor node and the remotest member from $G_J$. $G_i$ is updated by taking out the destination set of the selected neighbor node. $A_i$ is updated by adding in the selected neighbor node.

The algorithm will terminate at a node in three cases:

1) if all $D_j = \emptyset$

2) if $N_i = A_i$

3) if $G_i = \emptyset$

If none of the above condition are satisfied, it will go back to step 3 to process other neighbor nodes until one of the three conditions is satisfied.

Each node executes the same algorithm as its ancestors whenever a packet arrives. The whole algorithm execution will finally stop when the radius for every packet becomes zero.

## 3.3 Example

We use the network shown in Figure 3.1 to illustrate our algorithm. The result shows that the cost generated by our algorithm is 11; but with shortest path algorithm, the total cost is 13.

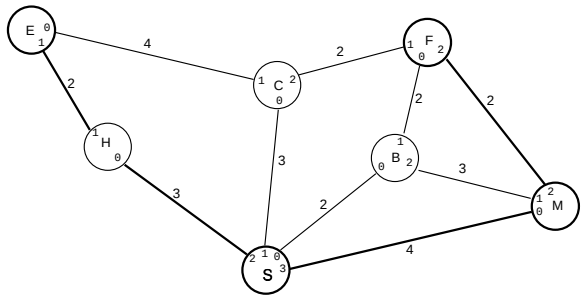


Figure 3.1: Example Network

Assume that the source node is $S$, and the destination set $G = \{E, F, M\}$.

**At node** $S$

1) The packet is generated at $S$, $p$.radius $= \infty$, $A_S = \emptyset$

2) $G_S = \{E, F, M\}$

3) $N_S = \{H, C, B, M\}$, $A_S = \emptyset$

$D_H = \{E\}$ $L_H = 3 + 2 = 5$

$D_C = \{E, F\}$ $L_C = (3 + 4 + 2)/2 = 4.5$

$D_B = \{F, M\}$ $L_B = (2 + 2 + 3)/2 = 3.5$

$D_M = \{M, F\}$ $L_M = (4 + 0 + 2)/2 = 3$

4) $J = M$ $R = 2$

**send packet to node** $M$ **with** $R = 2$

5) $G_S = \{E\}$ $A_S = \{M\}$ *goto step* 3

3) $N_S - A_S = \{H, C, B\}$

$D_H = \{E\}$ $L_H = 5$

$D_C = \{E\}$ $L_C = 7$

$D_B = \emptyset$ $L_B = \infty$

4) $J = H$ $R = 2$

**send packet to node** $H$ **with** $R = 2$

5) $G_S = \emptyset$ *stop*

**At node** $M$**:**

1) $G = \{E, F, M\}$, $p$.radius $= 2$, $A_M = \{S\}$

2) $G_M = \{M, F\}$

3) $N_M = \{B, S, F\}$ $A_M = \{S\}$ $N_M - A_M = \{B, F\}$

$D_B = \emptyset$ $L_B = \infty$

$D_F = \{F\}$ $L_F = 2/1 = 2$

4) $J = F$ $R = 0$

**send packet to node** $F$ **with** $R = 0$

5) $G_M = \{M\}$ $A_M = \{S, F\}$ *goto step* 3

3) $N_M - A_M = \{B\}$

$D_B = \emptyset$ *stop*

**At node** $H$**:**

1) $G = \{E, F, M\}$, $p$.radius $= 2$, $A_H = \{S\}$

2) $G_H = \{E\}$

3) $N_H = \{E, S\}$ $N_H - A_H = \{E\}$

$D_E = \{E\}$ $L_E = 2$

4) $J = E$ $R = 0$

**send packet to node E with** $R = 0$

5) $G_H = \emptyset$ *stop*

In the end, all packets at all involved nodes have $R = 0$; the entire algorithm execution is terminated.

## 3.4 Practical Considerations

For more efficiency, we can allow the packet to carry a limited number of member addresses. Suppose we let the packet carry the addresses of the four remotest members, the radius $R$ will be smaller and the algorithm will converge faster. As we saw in Chapter 2, the IP header usually has 40 bytes for optional use[1]; therefore, such consideration is highly feasible.

# Chapter 4

# Simulation Model and Results

## 4.1 Performance Measurements

The simulation study will measure two different relationships: one is delay versus offered load, the other is utilization versus offered load. Delay is defined to be the interval between the time a packet is generated and the time the packet arrives at the destination. In a multicast environment, the delay is the average delay for all destinations. If a packet is sent to a destination more than once, we only count the delay for the first one. Offered load is defined to be the offered rate of packet generation. Utilization is defined to be the average utilization of all links. We also define the throughput as:

$$throughput = offered\ load$$

(In networks, throughput is commonly defined as offered load − rejected load[4]. In our simulation, we always have rejected load = 0.)

## 4.2 Network Parameters

The parameters of interest in our algorithm are defined in Table 4.1. Note that the *ms* in the table should be interpreted as a relative time unit. The initial group population is also the average number of active groups, with the group birth and

| | |
|---|---|
| $t_b$ | transmission rate, *bits/ms* |
| $t_p$ | packet transmission time, *ms* |
| $\lambda_u$ | unicast packet generation rate, *packet/ms* |
| $\lambda_m$ | multicast packet generation rate, *packet/ms* |
| $\lambda_g$ | birth and death rate of groups, *group/ms* |
| $G$ | initial group population |
| $G_{\max}$ | maximum size of a group |
| $G_{\min}$ | minimum size of a group |

Table 4.1: Network Parameters

death rates being equal.

## 4.3 Data Structures Used with Simulation

### 4.3.1 Network

A network is a set of nodes and a set of directional links, where nodes can be numbered from 0 to $N-1$ for a network with $N$ nodes. Connectivity information is stored in a list of neighbors which is attached to every node. For the simulation purpose, We only need to maintain one copy of the network topology description (connectivity and link length), although in reality the description is duplicated at every node.

The information stored at a node includes:

1. Routing table. For our purpose, the following entry is enough. For an $N$ nodes network, there should be $N-1$ entries for each routing table.

| destination | distance | next hop |
|---|---|---|

2. Group table. This is a global group table. Each entry comprises a group number and a bit map indicating which node is in the group. If there are $k$ groups existing in network, there should be $k$ entries for the group table.

| group | node 1 | node 2 | ... |
|---|---|---|---|

3. Group presence table. This is a local group table. It is a hash table since groups join and leave and insert/delete operations are expected. This table indicates that the current node is participating in which group.

| group 1 | group 2 | group 3 | ... |
|---|---|---|---|

4. For the proposed algorithm, a node also needs the routing tables from its neighbors. For simulation, we don't need to store duplicated tables at different nodes. We may ignore the impact of transferring routing table information between neighbors. (In fact, if data transfer is not desirable, a separate calculation will also be able to generate the neighbors' routing tables at a small cost.)

5. For the algorithm we are comparing ours with we need, as described earlier, a set of cache records at each node.

| source | group | parent | min-hops[ ] |
|---|---|---|---|

where 'min-hops' is an array of $K$ elements for $K$ neighbors.

### 4.3.2 Packets

We define three types of packets: DATAS, DATAG, and GRPJL. All packets have the common fields:

| | |
|---|---|
| type | the packet type |
| src | where the packet is originated |
| at-node | currently the packet is at which node |
| from-link | from which link the packet comes to this node |
| to-link | to which link the packet will be forwarded |

Each type of packet has its particular fields:

1) DATAS: This type is unicast data packet with a single destination address.

| | |
|---|---|
| dst-addr | destination address |
| seq | sequence number for this (src, dst) pair |
| birth-time | when the packet is generated |

2) DATAG: This type is multicast data packet with a group destination address.

| | |
|---|---|
| gid | destination group address |
| seq | sequence number for this (src, grp) pair |
| radius | proposed algorithm only |
| radius-time | when the radius is computed |
| hop-limit | compared algorithm only |
| birth-time | when the packet is generated |

3) GRPJL: This type is group Join/Leave packet.

| | |
|---|---|
| gid | group that src is joining/leaving |
| op | JOIN/LEAVE |

## 4.4 The Compared Algorithm

We simulate and compare the performance of our proposed algorithm with the LS multicast routing algorithm by Deering [6]. Both algorithms require roughly the same amount of information, but Deering's algorithm, a low-delay algorithm, needs to maintain a delivery tree for each active source-group pair. For simplicity, we omit the detail of a subnetwork, and treat each router as a node, which can also serve as a host. A simplified version of Deering's algorithm is given in table 4.2.

## 4.5 The Simulation Model

We use a software called *smpl*, which is described in [10], as our tool for the simulations. The *smpl* package is an event-oriented simulation language which is designed for the simulation of computer and communication systems.

We use a *smpl* facility, which in essence is a server in the queuing model, to represent the links between nodes. Link $< i, j >$ is considered different from link $< j, i >$ as physically they are two channels and can have unequal length. Hence they are defined as two facilities. The utilization of this facility will be equal to the utilization of the links. A *smpl* token is used to represent a pointer to a packet.

```
1. On appearance/disappearance of group g on node k:
        flood link-state update, reporting change of g on k
2. On generation/receipt of link-state update, reporting change of g
        for each cache record c with c.group = g
                discard c
3. On receipt of multicast packet from source s to group g via node k
   with remaining hop-limit hl:
        if no cache recode c such that c.source = s and c.group = g
                create cache record c; c.source = s; c.group = g
                <compute c.parent and c.min-hops>
        if c.parent = k
                hl = hl −1
                for each adjacent node j
                        if hl ≥ min-hops[j]
                                send copy of packet to node j
```

Table 4.2: *Tree* Algorithm

We assume that the packet generation interval for both unicast and multicast is exponentially distributed with mean $1/\lambda_u$ and $1/\lambda_m$ respectively. We also assume that the group's birth and death interval is exponentially distributed with mean $1/\lambda_g$.

**Event routines.** The simulation model comprises six event routines as shown in Figure 4.1. We assume a broadcast protocol for link state report which includes group membership change report (Join, Leave). To ensure fast delivery of such information and not go into the details of flooding protocols, we pre-calculate a broadcast tree. The broadcast packet will have the highest priority. When a broadcast packet arrives at a node, the packet will be forwarded to all neighbors in the broadcast tree except where it is from. The events are described as follows.

**DATAGEN** The simulation enters this event when a data packet is generated at a node. The packet can be either a unicast data packet or a multicast data packet. For unicast, a destination node is randomly picked up; for multicast, a

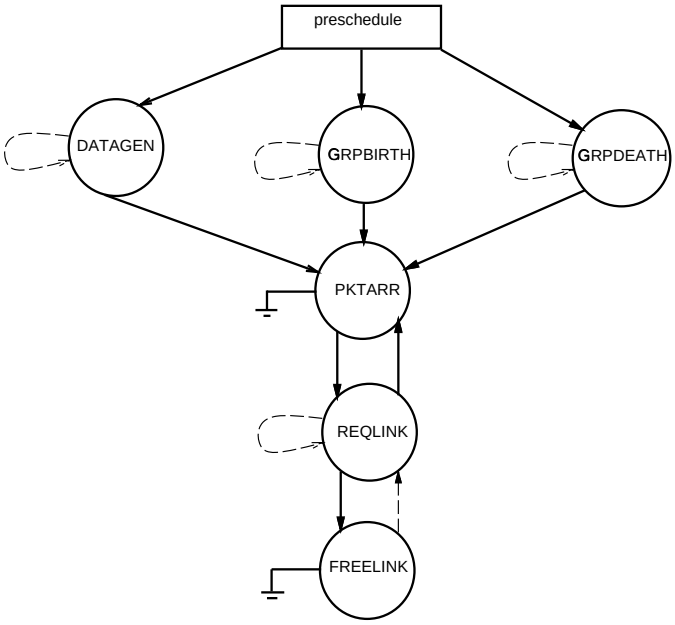


Figure 4.1: Events

destination group is randomly picked up; and for each packet being generated, a PKTARR event is scheduled. Finally an event for the same data type is scheduled, whose time exponentially distributed with mean $1/\lambda_u$ or $1/\lambda_m$, respectively.

**GRPJL** This event means a node joins/leaves a group. The join/leave period is exponentially distributed.

On a BIRTH event, a group size is randomly chosen between min-size and max-size, and a set of nodes are randomly picked up as group members. A packet of type GRPJL and operation JOIN is generated for each group member, with a random delay, then a PKTARR event is scheduled for each packet. Another BIRTH event is also scheduled, in time exponentially distributed with mean $1/\lambda_g$.

On a DEATH event, a group is randomly chosen. Then for each group member, a packet of type GRPJL and operation LEAVE is generated, also with a random delay. Similarly a PKTARR event is scheduled for each packet. Another DEATH event is scheduled too, similar to a BIRTH event.

**PKTARR** This event is entered when a packet arrives at a node, or when a packet is just generated at a node. This is the major event where activities for the two algorithms differ. Generally, if this node is a destination node, do statistics. If the packet needs to be forwarded, calculate the link(s) the packet should be forwarded onto and schedule a REQLINK event for each forwarding link. The three different types of packets are treated differently as described below:

DATAS: Unicast packet. This packet is simply to be sent to the next hop which can be looked up from the routing table.

DATAG: Multicast data packet. One of the algorithm procedures *radius* (for the proposed algorithm) or *tree* (Deering's algorithm) is to be called to compute the next hops, then the replicated packets are sent to these next hop nodes.

GRPJL: Group join/leave packet. If the node is the one that originates the packet, both the local group table and the global group table in this node are updated. If it is not the originating node, only the global-group table is updated. The packet is also forwarded to the neighbors in the broadcast tree. For the compared algorithm, the receiving node of this packet will delete all cache records $c$ such that $c$.group equals the involved group.

**REQLINK** The packet tries to reserve a link for transmission. If the link is available, schedule a PKTARR event in time $t_p$ and a FREELINK event immediately. If the link is busy, this event will be queued at the link, and later when the link is free, the queued event will be scheduled again.

**FREELINK** The link previously reserved is released after the transmission is completed. The head of the queued packets along with the REQLINK event at that link will be scheduled by the system to occur immediately.

## 4.6 Simulation Results

For simplicity, we call our algorithm "*Radius* algorithm" and Deering's algorithm "*Tree* algorithm." We randomly generated four networks with 40, 75, 97, and 123 nodes respectively (these sizes are sufficiently representative.) In each network, the group population fluctuates along an average, by a birth/death procedure, and the group size fluctuates as a random number uniformly distributed in a certain range. In the simulation, the multicast packet generation rate $\lambda_m$ is set to be the same as he unicast rate $\lambda_u$. These rates are changing for different passes of simulation. The rates are assigned the values with which no severe congestion is observed and reasonable utilizations are obtained. The following table gives other parameter ranges for the four networks:

| network | $G$ | $G_{\min}$ | $G_{\max}$ | av. # links | hit ratio | $\lambda_g$ | $t_p$ | link len |
|---|---|---|---|---|---|---|---|---|
| 40 | 3 | 2 | 39 | 2.95 | 91% | 1/1000 | 0.1 | 1 |
| 75 | 5 | 2 | 37 | 2.59 | 91% | 1/1000 | 0.1 | 1 |
| 97 | 4 | 2 | 48 | 3.22 | 92% | 1/1000 | 0.1 | 1 |
| 123 | 3 | 2 | 40 | 3.25 | 86-95% | 1/1000 | 0.1 | 1 |

See Section 4.2 for the meaning of the variables. In the table, the value 1/1000 for $\lambda_g$ means that the group birth and death rate is 1/1000 of the unicast packet generation rate.

The hit ratio is the ratio for not reconstructing the delivery tree for *Tree* algorithm. This is not a free variable, rather it is the function of other parameters. We choose the parameters so that the ratio is approximately slightly over 90%, which should be a reasonable assumption. This ratio can be decreased by increasing group appearance/disappearance or group membership change frequency, or by increasing group size or the number of groups.

We execute both our algorithm and Deering's algorithm with exactly the same

network parameters.

We assume that the time for the reconstruction of the delivery tree for *Tree* is $(E+N)\log N$ units[11], where $N$ is the number of nodes, and $E$ is the total number of links in the network. This is equivalent to transmitting $((E+N)\log N)/L$ packets from one node, where $L$ is the packet length (in bits). We also assume that the time for selecting the next hop for *Radius* algorithm is $G_s \times N_l$ units for each node, where $G_s$ is the destination group size and $N_l$ is the number of links at that node. This time is equivalent to transmitting $G_s \times N_l/L$ packets.

The first 3,000 packets received at each node (on average) are discarded to ensure that the network reaches a steady state; then we will do the statistics for the subsequent 10,000 packets received at each node (on average).

The simulation results are shown in Figures 4.2 to 4.11. In these figures, the horizontal axes represent the message rate which is the offered load. The vertical axes represent the delay (Figures 4.4, 4.6, and 4.8), utilization (Figures 4.5, 4.7, and 4.9), and reduction percentage (Figures 4.10 and 4.11), respectively. Figures 4.12,4.13, 4.14, and 4.15 are the enlarged lower portion of Figures 4.2, 4.4, 4.6, and 4.8, respectively. Figures 4.10 and 4.11 are the average reduction percentage for delay and utilization on the three larger networks, by using the following formula:

$$A = \frac{\sum_{i=1}^{3} \frac{T_i - R_i}{T_i}}{3} \times 100\%$$

where

$A$: average percentage delay or utilization reduction.

$T_i$ : *Tree* delay or utilization for the $i$th network.

$R_i$ : *Radius* delay or utilization for the $i$th network.

The results for the smaller, 40 node network, are based on a different range of

message rates (100 to 800) from that used in the three larger networks (50 to 400), because the smaller network can tolerate higher message rates for both algorithms. Therefore, the reduction cannot be averaged with the lager network results. We give similar calculations in the following table for the 40 node network.

| Message rate | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |
|---|---|---|---|---|---|---|---|---|
| Delay reduction | -2.6 | 2.5 | 17.0 | 10.3 | 29.8 | 23.2 | 29.6 | 2.6 |
| Util. reduction | 0.0 | 5.0 | 7.4 | 5.6 | 4.3 | 6.0 | 3.4 | 3.0 |

In the table, the numbers in the last two rows are reduction percentages.

## 4.7 Simulation Results Analysis

The simulation results show that our algorithm has a better performance than the compared algorithm since both the delay and the utilization are lower on the four networks. At certain crucial message rates, the delay can be lowered by 40–50%. The delay reduction at these rates means *Radius* can support higher offered load than can *Tree*.

Since the results for all four networks are similar, we only give a detailed analysis of the 75 node network (Figures 4.4, 4.5, and 4.13). For this 75 node network, when the message rate is from 50 to 100 packets/unit time, the delay goes from 0.55 to 0.58 for *Radius* and from 0.57 to 0.60 for *Tree*. In such a low message rate range, the main delay for both algorithms is caused by the transmission delay. Since *Tree* needs to reconstruct the delivery tree occasionally, this causes the higher delay than that of *Radius*. When the message rate is higher than 100 packets/unit time, the delay and utilization for both algorithm increase, but the delay and utilization for *Radius* are always lower. The reason is given as follows:

When the message rate increases, the corresponding utilization of links increases, as seen in figure 4.5. Utilization reflects the traffic load. The increased traffic causes

some packets to be queued for the available links. At this time, the queuing delay becomes dominant, while the transmission delay becomes insignificant. A few calculations can also show that the delivery tree computation time will have little impact on the total delay at higher message rates. The 75 node network has $E = 97$ links, therefore the tree computation time is $(N + E) \log N = 1071$, which is roughly the time to transmit one packet from a node. Suppose on average a packet travels half the "diameter" of the network, or $\sqrt{75}/2 = 4.33$ hops to reach its destination, with a 90% cache hit ratio, the probability of having one tree computation on its way is $4.33 \times 0.1 = 0.433$. In other words, a packet will on average need 0.433 tree computations, or equivalently, in its total delay a portion of 0.433 packet transmission times is caused by tree computations. Given that a packet transmission time is 0.1 in the time scale shown in the delay figures, from Figure 4.4, we can see the delay at the higher range is in the order of 100 to more than 1000 packet transmission times, compared with which 0.433 is certainly a tiny portion. In the case when the queuing delay plays the major role, the *Radius* algorithm shows more advantages, since it is a low-cost algorithm. Reduced total cost means reduced total traffic load, while the reduction in traffic load will decrease the waiting time for queued packets. Therefore, the total delay is eventually decreased. This is why *Radius* has lower delay than *Tree*, despite the fact that *Tree* always uses the shortest paths and *Radius* may use longer paths from time to time. Note that for the 123 nodes network in Figure 4.8, there is a crossover. From the figure, we can see that the curve for *Radius* is smoothly increasing when the message rate increases, while the curve for *Tree* suddenly slows down when the message rate increases from 300 to 350, and goes up again for the message rate 400. This might be caused by some special combination of network topology and message distribution, since both the network and the messages are randomly generated.

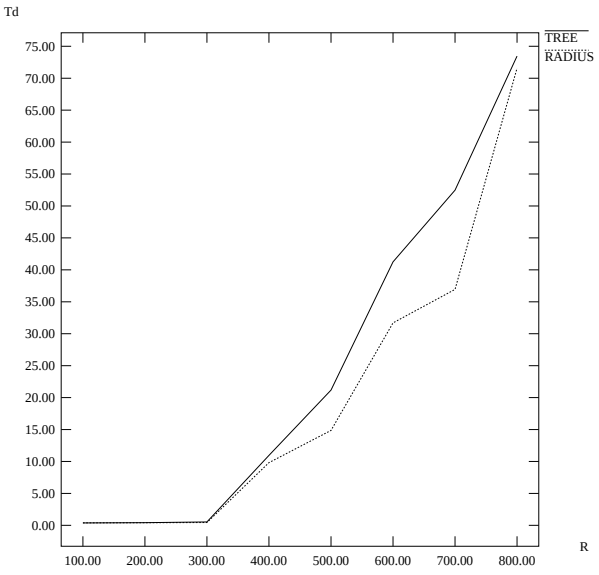


Figure 4.2: Delay ($N = 40$)

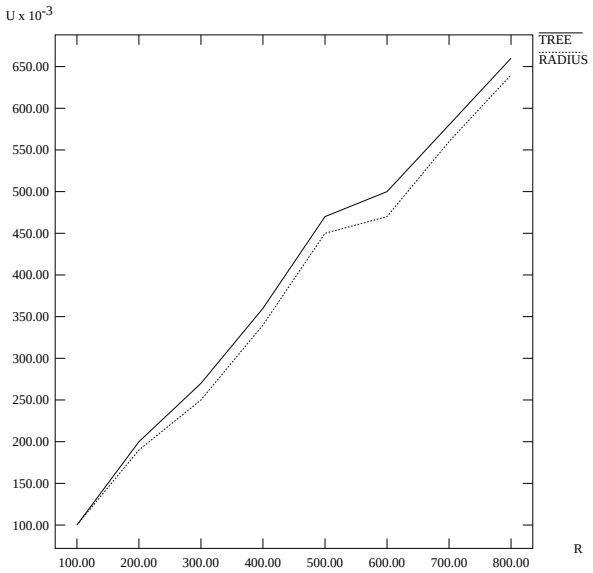


Figure 4.3: Util. ($N = 40$)

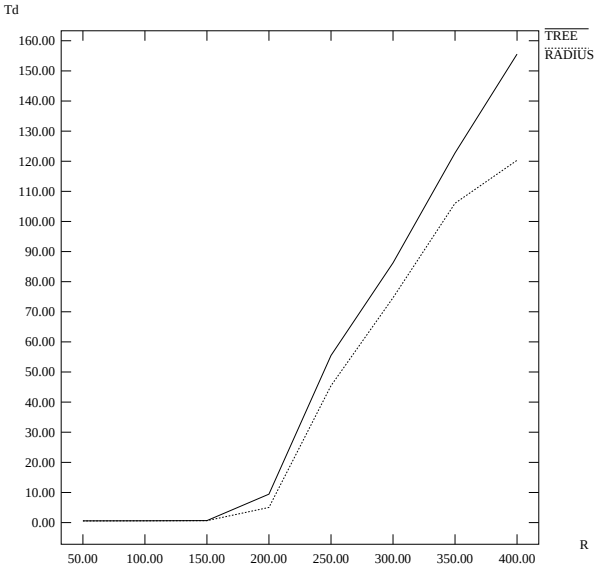


Figure 4.4: Delay ($N = 75$)

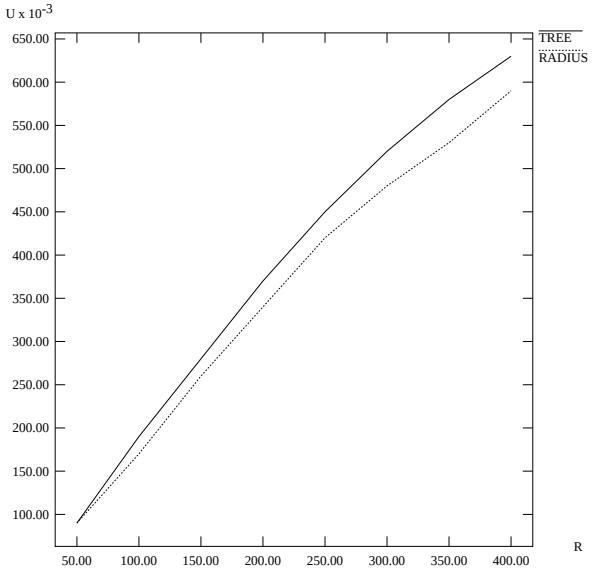


Figure 4.5: Util. ($N = 75$)

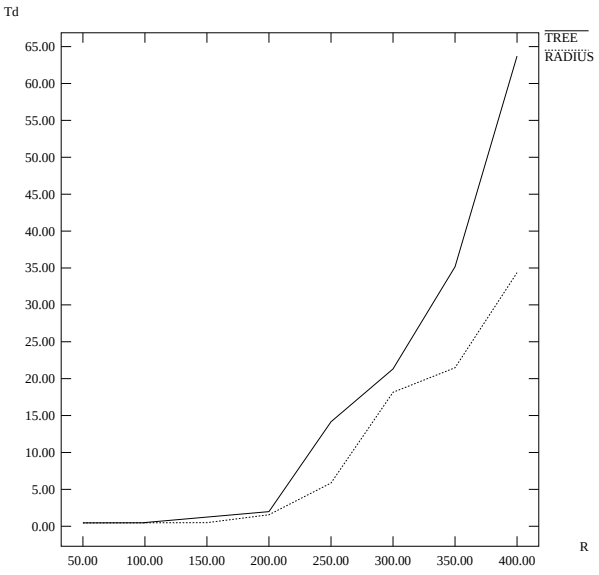


Figure 4.6: Delay ($N = 97$)

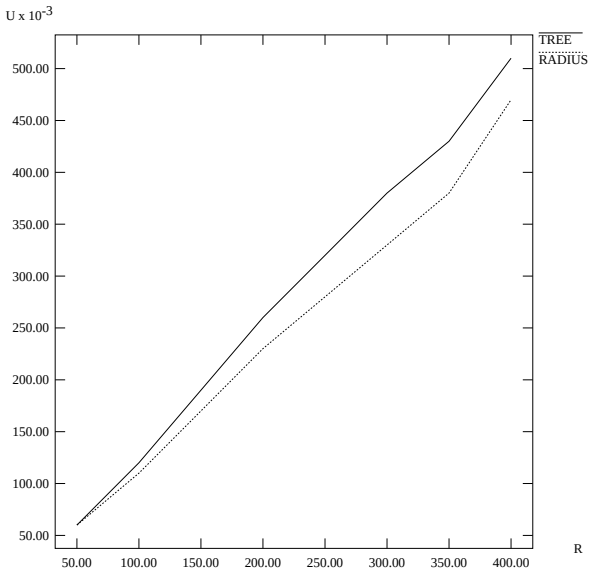


Figure 4.7: Util. ($N = 97$)

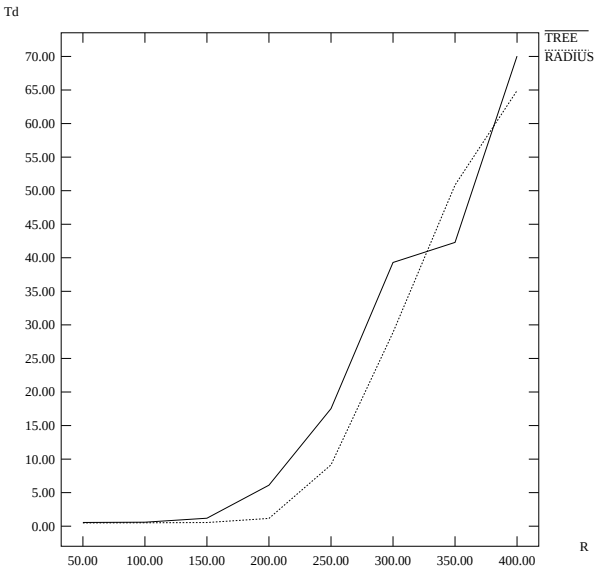


Figure 4.8: Delay ($N = 123$)

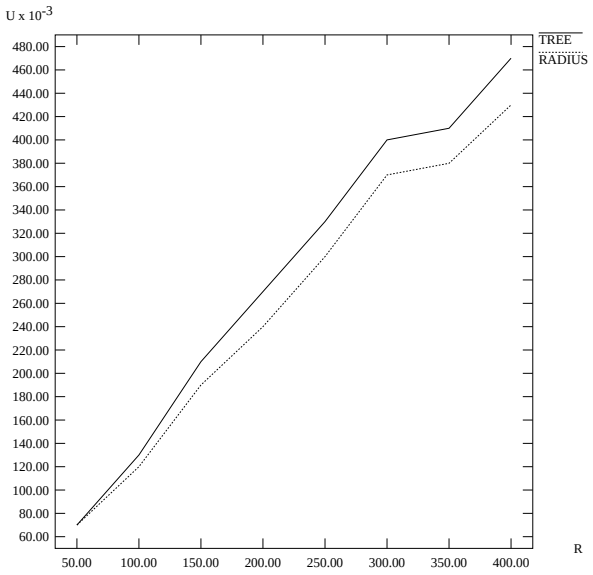


Figure 4.9: Util. ($N = 123$)

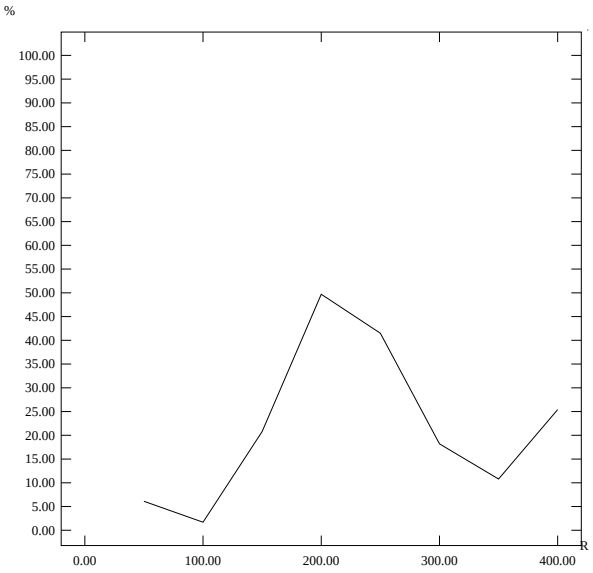


Figure 4.10: Delay Reduction

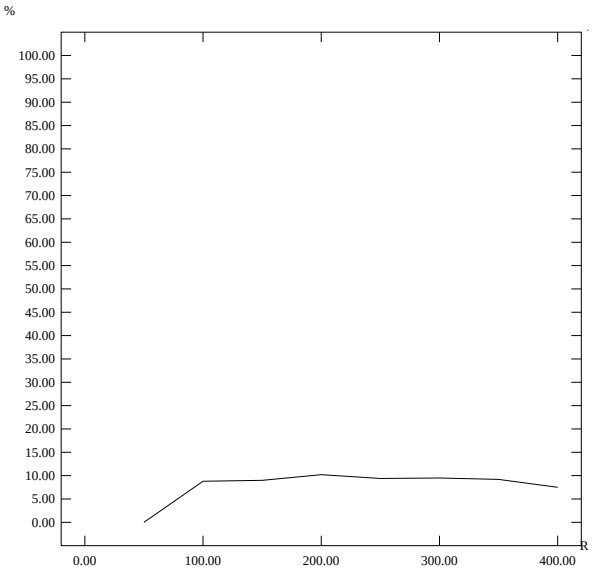


Figure 4.11: Util. Reduction

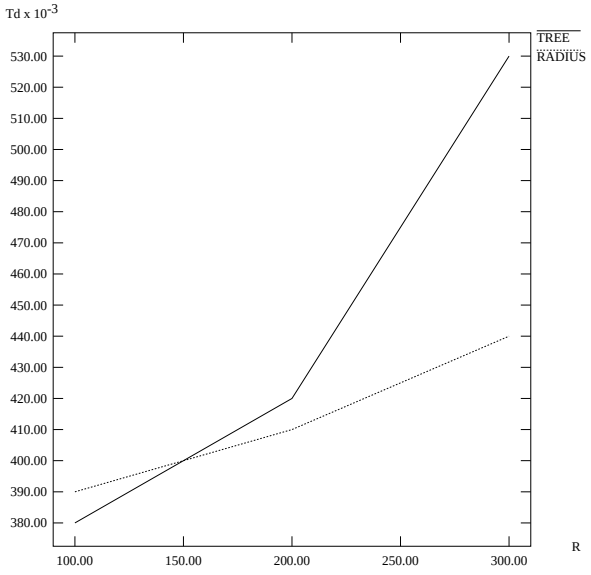


Figure 4.12: Delay (Enlarged, $N = 40$)

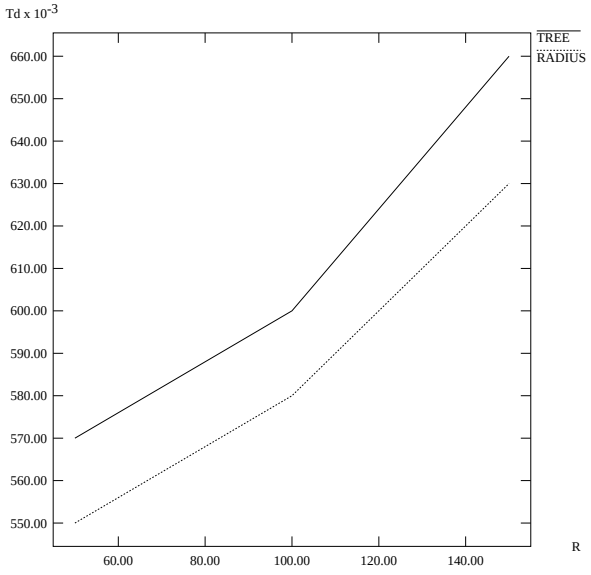


Figure 4.13: Delay (Enlarged, $N = 75$)

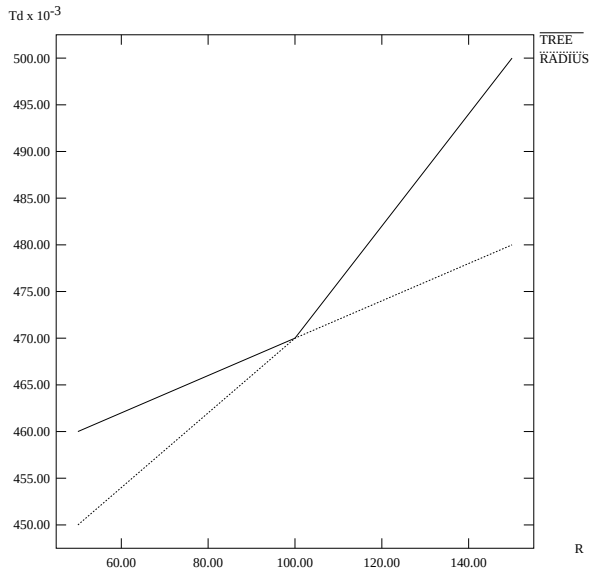


Figure 4.14: Delay (Enlarged, $N = 97$)

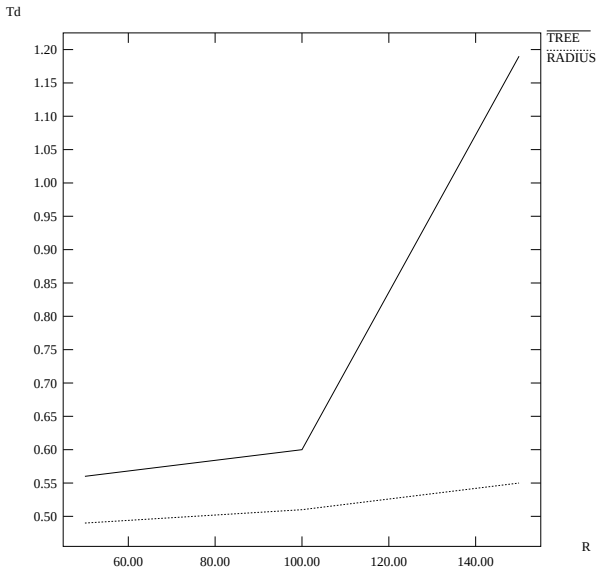


Figure 4.15: Delay (Enlarged, $N = 123$)

# Chapter 5

# Conclusions and Future Work

We conclude this thesis with a discussion of the relative advantages of our algorithm and some possible future work.

**Performance.** The simulation results show that our algorithm has a better performance. Both the delay and utilization are lower than that for the compared algorithm. When the message rate is very low and the groups are static, the compared algorithm might offer a slightly lower delay. But in that case, the delay is very low for both algorithms.

**Functionality.** This algorithm supports transmission from any node to any group, where the member of the group may be distributed across any number of groups, and may join and leave an group independently at any time.

**Scalability.** This algorithm is insensitive to the network topology, group size, and the number of groups. Whenever topology changes or a group birth or a group death occurs, or group membership changes, this algorithm executes exactly the same way.

**Compatibility.** Our algorithm is compatible with unicast environment. It does not require excessive information compared with that used in unicast environment. The routing tables of the neighboring nodes are usually available for many unicast

algorithms such as the distributed Bellman-Ford algorithm mentioned in Chapter 2.

**Reliability.** This algorithm has the same probability of loss, or damage of packets as the unicast routing. Whenever the routing table changes, the radius $R$ is set to be $\infty$; this guarantees that no loss is caused by the routing table change.

**Simplicity.** This algorithm is conceptually simple and easy to implement; computation time complexity is only linear in group size, therefore it is very efficient. In addition, no delivery tree is to be maintained, so that no cost is incurred by maintaining such a tree. It handles the transient groups as efficiently as the permanent groups and will not cause CPU or memory congestions for the routers when there are a large number of source-group pairs.

One problem our algorithm faces is that there might be certain conditions under which a group member receives a message with prolonged delay. Because non-shortest paths are taken, when a node receives a message through a much longer path than the shortest path, this will be undesirable for time constraint applications. All low-cost algorithms have the same problem, and there may be times at which some control must be exercized to prevent it from happening.

This algorithm has been designed under the "link state" environment, that is, every node knows the network topology and the global group information. Another situation is that a node only knows the state of its neighbors, which is where the distributed Bellman-Ford algorithm finds its application. It will be useful to develop a low-cost routing algorithm for such environments.

# Bibliography


[1] L. Aguilar. Datagram routing for internet multicasting. In *Proc. ACM SIGCOMM '84 Communications Architectures and Protocols*, pages 58–63. ACM, June 1984.

[2] B. Awerbuch, A. Bar-Noy, and M. Gopal. Approximate distributed Bellman-Ford algorithms. *IEEE Tran. on Communications*, 42(8):2515–2517, August 1994.

[3] N. E. Belkeir and M. Ahamad. Low cost algorithms for message delivery in dynamic multicast groups. In *Proc. 9th Intl. Conf. Distributed Computing Systems*, pages 110–117. IEEE, June 1989.

[4] Dimitri Bertsekas and Robert Gallager. *Data networks.* Printice-Hall, Inc., 2nd edition, 1992.

[5] K. Bharath-Kumar and J. M. Jaffe. Routing to multiple destination in computer networks. *IEEE Tran. on Communications*, COM-31(3):343–351, March 1983.

[6] S. E. Deering. *Multicast routing in a datagram internetwork.* PhD thesis, Dept. of Electrical Engeering, Stanford University, December 1991.

[7] P. A. Humblet. Another adaptive distributed shortest path algorithm. *IEEE Tran. on Communications*, 39(6):995–1003, June 1991.

[8] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* North-Holand, P.O. Box 211, 1000 AE Amsterdam, The Netherlands, 1992.

[9] J. Knisely and R. Laskar. Minimal cyclic broadcast graphs. *Congressus Numerantium*, 104:210–214, 1994.

[10] M. H. Macdougal. *Simulating computing systems.* M.I.T. Press, Cambridge, Massachusetts 02142, 1987.

[11] R. Sedgewick. *Algorithms.* Addison-Wesley Publishing Company, Inc, New York, 2nd edition, 1989.

[12] J. M. Spinelli and R. Gallager. Event driven broadcast without sequence numbers. *IEEE Tran. on Communications*, 37(5):468–474, May 1989.

[13] W. Stallings. *Data and computer communications.* MacMillan Publishing Company, New York, 4th edition, 1994.

[14] G. I. Stassinopoulos. Optimal dynamic routing in multidestination networks. *IEEE Tran. on Communications*, COM-35(4):472–475, April 1987.

[15] G. I. Stassinopoulos and M. G. Kazantzakis. A computationally efficient iterative solution of the multidestination optimal dynamic routing problem. *IEEE Tran. on Communications*, 39(9):1370–1378, 1991.