DMRP: A Distributed Multicast Routing Protocol for ATM Networks

R.Venkateswaran C.S.Raghavendra
School of EECS,
Washington State University
Pullman, WA 99164-2752.
{rvenkate,raghu}@eecs.wsu.edu
Tel: (908)–949–2015.
Fax: (908)–834–5379.

X.Chen V.P.Kumar
Bell Laboratories,
101 Crawfords Corner Road,
Holmdel, NJ 07733.
{xchen,vijay}@bell-labs.com

Abstract

In this paper, we present a Distributed Multicast Routing Protocol (DMRP) for dynamic computation of multicast trees. This protocol is a distributed version of a centralized heuristic for dynamic Steiner Tree computation. The protocol is deadlock-free and is a correct implementation of the centralized heuristic. It can be easily extended to the hierarchical framework supported by ATM Forum's PNNI protocols. This makes the protocol very scalable. Our motivation is to make this protocol a standard for multicasting under the PNNI framework. Our protocol draws a lot of inspiration from PIM, which has been proposed as the standard for multicasting in the internet domain. We would like to emphasize that unlike PIM which caters to maintaining only soft-states, DMRP maintains hard states. The hard states are required under the ATM framework and makes the DMRP protocol compatible with the philosophy of PNNI. The sharing of the multicast tree among multiple senders makes effective use of the network resources. This reduces the overall cost of the multicast tree. The protocol relies on Participant Initiated Join, where the participant who wants to establish connection with the multicast group computes the path to the already existing multicast tree. Using simulations, we show that the number of messages generated before a participant can join the multicast tree is small. Further, the join latency is also small. The simplicity of the protocol lends itself to easy implementation for the support of multipoint connections under the PNNI framework.

1 Introduction

Many multimedia applications like video-conferencing require multicasting support from the underlying network. Multicasting involves sending a particular information from a sender to more than one receiver. An efficient way of achieving this is to construct a multicast tree spanning the receivers and sending the information along the links of the tree. This approach is optimal if the multicast tree is optimal. Thus, it is important to find optimal multicast trees for a given set of participants.* We believe that the cost of the multicast tree used for multicasting is very important, especially for high-bandwidth applications like a video-conference application. Minimizing the bandwidth consumption makes efficient use of the available resources and increases the number of connections that can be handled simultaneously. The bandwidth factor encompasses other traffic parameters like Peak Cell Rate, Average Cell Rate and Maximum Burst Size, which can be combined to compute the effective bandwidth of a connection.

The problem of finding an optimal multicast tree can be reduced to the minimal *Steiner Tree* problem in graphs[3], which is a well-known NP-Complete problem. For a fixed set of participants, there are simple heuristics to compute near-optimal Steiner Trees. The cost of these near-optimal trees is at most 2 times the cost of the optimal tree. Distributed implementations of these heuristics have a good performance, with each node maintaining only the local information and the identities of all the participants[2].

But, for a dynamically changing set of participants, it is very difficult to compute good multicast trees, unless the tree is restructured whenever the set of participants changes. In [5], Waxman proposed a heuristic to compute good multicast trees without restructuring the tree after every change. In this heuristic, when a node wants to join an already existing multicast tree, it uses the shortest path to join the nearest node that is already on the multicast tree. The heuristic minimizes additive resources like bandwidth by sharing links among different participants. QoS-based routing can be used here to satisfy QoS constraints. It must be noted that the cost of the tree based on the dynamic heuristic cannot be bounded by a constant factor of the cost of the optimal tree.

*A participant is either a sender or a receiver or both.

Here, we propose a distributed protocol to implement this heuristic. No node is aware of the set of participants because this set is always changing. In our distributed protocol, we have primarily used the cost based on effective bandwidth to compute the cost of the multicast tree. We would like to emphasize that this does not preclude the protocol from using QoS-based delay-constrained routing. Moreover, our protocol is independent of any core-selection mechanisms because there is no concept of a core node as described by the Core-Based Tree approach[1]. Only the nodes which have receivers in the switching system or downstream receive the data for a particular connection. Therefore, there is no unnecessary information being sent to nodes that do not have any receivers downstream. We show that our distributed protocol is deadlock-free and generates the same tree as generated by the centralized heuristic proposed by Waxman.

Our protocols can be easily integrated with the Private Network Network Interface (PNNI) routing and signaling framework proposed by the ATM Forum[4]. The PNNI routing and signaling protocols are scalable because the framework provides a hierarchical view of the network. The network consisting of nodes and links is divided into small groups called peer-groups. The peer-groups are logically represented by a peer-group leader (PGL) to form a logical network at a higher level of hierarchy. The details about PGLs and their role in maintaining topological information can be found in [4].

Our protocol is scalable to large networks due to its compatibility with the PNNI protocols. It also does not need any additional information other than what is already specified for point-to-point routing and signaling protocols. Therefore, it is an easy extension to the existing PNNI protocols.

2 Distributed Protocol within Single Peergroup

The peergroup is a simple network consisting of nodes and undirected links connecting these nodes. There is a positive, non-zero weight associated with each link that reflects, in some sense, the cost of using the link. The cost of the tree is the sum of the weights of the links that belong to the tree. The dynamic heuristic requires that a new node that wants to join an already existing multicast tree joins the nearest node that is already on the multicast tree. The new branch that gets added to the multicast tree is the shortest path between the new node and the corresponding nearest node.

We would like to emphasize that the *nodes are not aware of the membership of the multicast group or the structure of the current multicast tree.* It could be argued that the membership information can also be incorporated within the PTSEs. But, we feel that flooding membership information pertaining to thousands of multicast groups will put an enormous overhead on the control traffic. Further, the multicast group addresses *cannot be easily aggregated*. This results in an additional complexity for flooding membership information at higher level peer-groups.

2.1 Basic Idea of the Protocol

The basic idea of the protocol for a particular connection is very simple. In order to help us explain the protocol clearly, we make some assumptions. *It must be remembered that the basic distributed protocol can be extended to relax all these assumptions.* We assume that all the nodes have an identical view of the network and that the network topology does not change during the course of a join request. We also assume that the network is connected at all times and that messages are reliably delivered in FIFO-order. We would like to re-iterate that these assumptions are being made only for simplicity.

The protocol supports *Participant-Initiated Join*, where the participant node initiates the request to join a multicast group and also computes the path used to join the multicast tree. There are two phases of the protocol. In the first phase called the *Find* phase, the node finds out the identity of the nodes that are already on the multicast tree. In the second phase called the *Join* phase, it tries to join the nearest node that is already on the multicast tree. In the *Find* phase, a node that wants to join the multicast group sends a REQUEST message to all its neighbors on the shortest path tree, rooted at the node. Every neighbor forwards this message downstream to its neighbors along the links of the shortest path tree rooted at the originating node. This continues till the REQUEST message reaches either a node that is already on the multicast tree or a leaf node of the shortest path tree rooted at the originating node. Every node that is already on the multicast tree replies back to the request, with the cost set to 0. The leaf nodes that are not on the multicast tree reply to the request with cost set to -1. On receiving all the replies, a node determines the nearest node and forwards the REPLY message from the nearest node upstream towards the originator of the REQUEST. Before forwarding, the node updates the cost function to reflect the cost from this node to the nearest node. The originator receives all the replies and determines the nearest node. In the *Join* phase, the originator node computes the path to the nearest node and sends a JOIN-REQ to this node. The path is embedded in the message so that the intermediate nodes are not involved in path computation. The nearest node replies with a JOIN-ACK and all the nodes in the path become a branch of the multicast tree.

2.2 Handling multiple events

When more than one node tries to join the multicast session concurrently, the distributed protocol should ensure that the correct multicast tree is computed. We say that the multicast tree is correct if the same tree can be computed by a sequential, centralized algorithm based on Waxman's heuristic using one of the many possible orders in which the join requests are processed. Note that there are no loops in the computation of a correct multicast tree. Extensions are needed to the simple distributed protocol explained in Section 2.1 to ensure correctness. Two interesting extensions are time-stamps and the use of RETRY messages.

Time-stamps are used to enforce a total ordering on the REQUEST messages, based on the sequence of events. The ordering is used to impose serializability of the join requests. The RETRY message is a mechanism used to enforce sequential processing of join requests in order to arbitrate conflicts. The originator of a REQUEST message inserts the time-stamp information along with the message using the time-stamp information available at the node. After receiving a REQUEST message, a node uses the time-stamp value in the message to update its local time-stamp. The REQUEST message is then processed as explained in Section 2.1. State information pertaining to the message is created before the message is forwarded to neighboring nodes. The state information is cleared when the REPLY message is sent back to the originator of the REQUEST message. It must be remembered that the state information is pertinent only to a particular connection and has no relevance to other connections.

If a node receives another REQUEST message for the same connection before it has sent a REPLY message to a previous REQUEST message from another node, it creates a conflict and can lead to potentially incorrect computations of the multicast tree. The node, therefore, uses the time-stamp information in the messages to order them. The message with higher priority based on the ordering is processed as usual and the state information pertaining to the other message is cleared. A RETRY message is also sent to the originator of the cleared message. A RETRY message indicates to the originator to wait for some time and then send the REQUEST message again with the same time-stamp value, but a different value of "retry-count". The retry-count is used to distinguish the new REQUEST message from the previous REQUEST. The REQUEST message is also repeated if the originator does not receive all the REPLY messages within a specified time interval.

2.3 Finite State Machine

As explained earlier, each node maintains an independent state machine pertaining to each connection. The node could be in any of the following states:

- **IDLE :** All nodes are initially in this state. In this state, the node has no information pertaining to the multicast tree.
- **WAIT :** The node enters this state after it has sent/forwarded a request to join the multicast tree and is waiting for a response. In this state, the node maintains the state information for exactly one requester. Other requests are replied back, without maintaining state information.
- **TENTATIVE :** The node enters this state when it has sent/forwarded a JOIN-REQ message to the nearest node which is already on the multicast tree. In this state, the node is waiting for an acknowledgment.
- **ACTIVE :** The node enters this state when it is on the multicast tree. This state is reached after a node receives a JOIN-ACK from a node that is already on the tree. In this state, the node

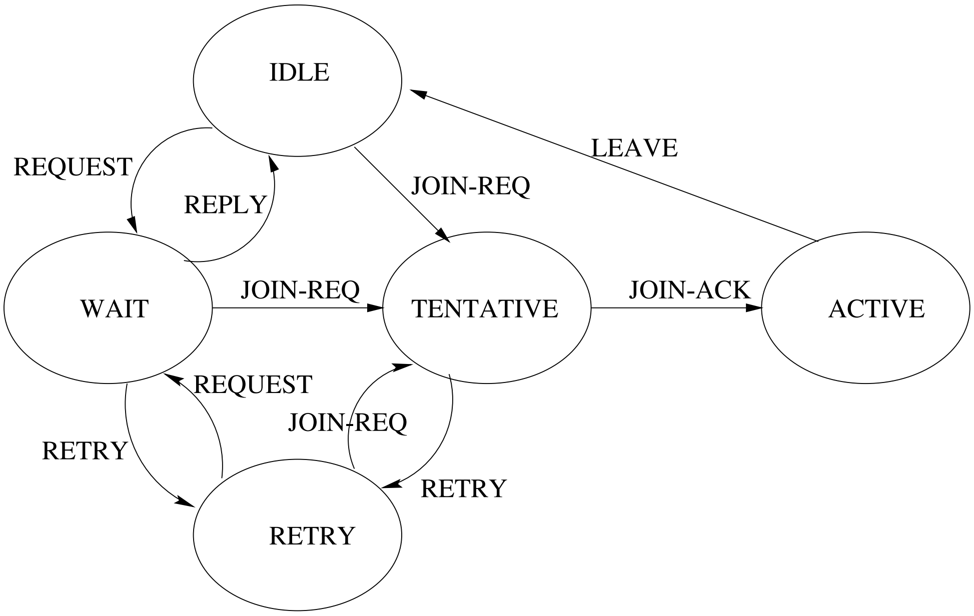

Figure 1: State Transition Diagram

maintains the multicast tree information regarding all the links that are incident on it and belong to the multicast tree. It does not maintain any state information about new requests to join the tree.

- **RETRY :** The node reaches this state when it has originated a REQUEST to join the multicast tree and gets a RETRY message on one of the links. In this state, the node does not maintain any state information about other requests. It waits for a random period of time before re-issuing the original REQUEST. The retry count keeps track of the number of attempts made by the node to join the multicast tree.

The node in a particular state makes a transition to another state on receiving certain messages. The state transition diagram is shown in Figure 1.

3 Proof of Correctness

We will show that our protocol is correct, that is, the multicast tree produced by our protocol is the same as that produced by a sequential, centralized algorithm based on Waxman's heuristic using one of the many possible orders in which the requests are processed.

We first introduce the concept of a *pruned tree*, rooted at a given node. A pruned tree rooted at a given node is the shortest path tree rooted at that node after pruning all the nodes and links that appear below an ACTIVE node†. The REQUEST message originating at a node is *not* received by any node other than those that belong to the pruned tree rooted at that node. For example, consider Figure 2(a). In this figure, the ACTIVE nodes C,F and H are shown as dark nodes. The shortest path tree rooted at node A is shown in Figure 2(b). The pruned tree rooted at node A is derived from the shortest path tree after pruning the nodes that appear below nodes C (prune nodes E, H and I) and F (prune node G). This tree is shown in Figure 2(c). A REQUEST message originating at node A will not be seen by nodes E, H, I and G.

Lemma 1 *The ACTIVE node nearest to any given node is always on the pruned tree rooted at the given node.*

†a node that is already on the multicast tree.

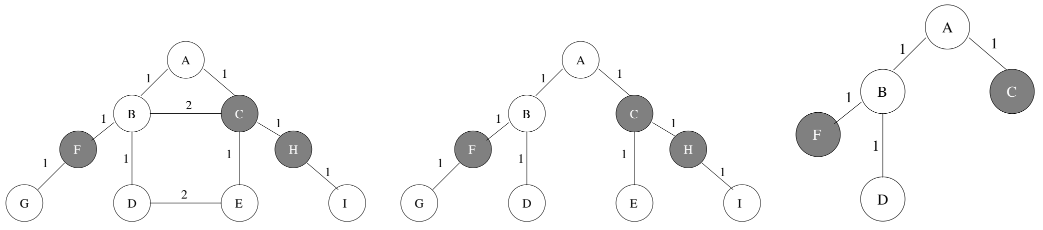

Figure 2: (a) The entire topology. (b) Shortest-path tree rooted at node A. (c) Pruned tree rooted at node A

Proof : The proof is based on the definition of the pruned tree. The pruned tree is formed from the shortest path tree rooted at the given node. On the shortest path tree, there can be no other ACTIVE node on the branch from the root node to its nearest ACTIVE node. All the nodes on this branch, including the nearest ACTIVE node are not pruned. Therefore, the nearest ACTIVE node will be on the pruned tree. ■

Lemma 2 *If there is only one JOIN request, the tree generated by the distributed protocol is correct.*

Proof : From Lemma 1, the nearest ACTIVE node is on the pruned tree. If there is only one request, this ACTIVE node responds with a REPLY message. The REPLY messages from other ACTIVE nodes reflect the cost of joining the multicast tree using these nodes. After receiving replies from other nodes, the joining node computes the nearest ACTIVE node and sends a JOIN-REQ to this node, using the shortest path. By our assumption, the topology information does not change during the course of the join. Therefore, the resulting tree is identical to the tree computed by the centralized algorithm. ■

Lemma 3 *The tree generated by the distributed protocol for two concurrent join requests is correct.*

Proof : Let nodes A and B be the nodes originating the join requests. The REQUEST messages from these nodes can be ordered based on the time-stamp values. Without loss of generality, let the message from A have a higher priority based on the ordering than the message from B. We have to consider two possible cases.
Case 1: *Node A is not present on the pruned tree rooted at node B and vice versa.*
In this case, the REQUEST from B does not reach A and vice versa. Therefore, the two concurrent join requests can proceed independently. Join of node A does not affect the join of node B. Therefore, the structure of the final multicast tree after nodes A and B have joined is not affected by the order in which the requests are processed. Using Lemma 2, we can conclude that the two joins will produce a correct tree.
Case 2: *Node A appears on the pruned tree rooted at node B and vice versa.*
Since message from A has a higher priority, the message from A that reaches B is processed normally. On the other hand, the message from B either reaches a node that is in WAIT state due to the REQUEST from node A or it reaches node A. In the former case, node B gets a RETRY message and has to send the REQUEST after some time. In the latter case, the message reaches node A when node A is in one of the 3 states WAIT, TENTATIVE or ACTIVE. If A is in ACTIVE state, it has already joined the tree. Therefore, B's join can be processed as a single join. From Lemma 2, the final tree generated is correct. If A is in the WAIT or TENTATIVE state, it sends a RETRY message to B. Therefore, B cannot join the tree unless A is in the ACTIVE state. Therefore, the final tree generated is correct. ■

Theorem 1 *The distributed protocol generates correct trees when there are k concurrent join requests.*

Proof : We can prove this using induction on k. ■

Theorem 1 can be easily extended to the case when there are multiple join and leave requests. The correctness of the protocol is not compromised by the inclusion of leave requests. The next lemma illustrates this fact.

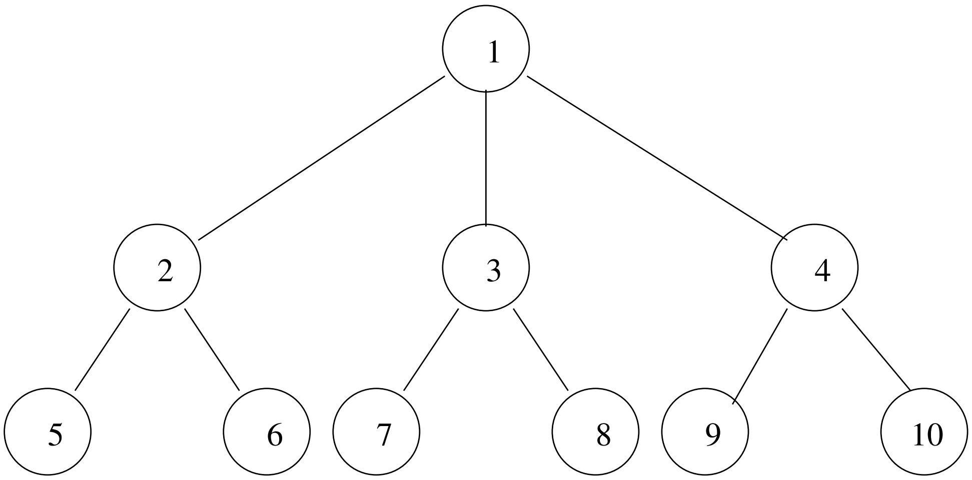

Figure 3: Complete Shortest Path Tree for $d = 3$ and depth 3

Lemma 4 *Only a finite number of leave requests can be processed during the time taken by a new node to join.*

Theorem 2 *The protocol is correct when there are k concurrent join and m leave requests.*

Proof : Using induction and results from Theorem 1 and Lemma 4. ■

4 Average Case Analysis of the Protocol

To analyze the average case scenario, we assume a very simple model of the network and compute the number of messages that are required so that a new node can join the multicast tree.

Given a network with N nodes and average degree d, let h be the average depth of the shortest path tree rooted at any one node. In particular, for d=3, we define h as:

$$h = 1 + \lfloor \log_2 \frac{(N+2)}{3} \rfloor \tag{1}$$

The shortest path tree for a simplified network with d=3 is shown in Figure 3. Let k be the number of nodes already on the existing multicast tree. Then the total number of messages sent, on an average, before a new node can join the multicast tree is given by :

$$M(h,k) = 2.L(h,k) + 2.\frac{h}{k} \tag{2}$$

Here, the first term corresponds to the ***Find*** phase of the protocol and the second term corresponds to the ***Join*** phase of the protocol. $L(h,k)$ is the average number of links on the pruned shortest path tree. In particular, for d=3,

$$L(h,k) = (\frac{N-k}{N})(3 + \sum_{i=0}^{k} \sum_{j=0}^{k-i} P(i,j)(L_2(h-1,i) + L_2(h-1,j) + L_2(h-1,k-i-j))) \tag{3}$$

where, $N = (3 * 2^{h-1} - 2)$ is the number of nodes on the tree and $P(i,j) = \frac{C_i^k \ C_j^{k-i}}{3^k}$ is the probability that there are exactly i and j ACTIVE nodes on the left and middle subtree respectively.

The term $L_2(h,k)$ is specific if the shortest path tree is a complete binary tree. It is used in Equation 3 because, for d=3, all the subtrees attached to the root in Figure 3 are complete binary trees. A complete binary tree with depth h has $(2^h - 1)$ nodes. The k ACTIVE nodes that are a part of the multicast tree are assumed to be uniformly spread on this binary tree. After pruning nodes and links that are below an ACTIVE node, the number of links in the pruned tree is used to compute the number of messages required to join the multicast tree. The average number of links on the pruned tree can be recursively calculated as

$$L_2(h,k) = \begin{cases} 0 & \text{if } 2^h - 1 \leq k \\ (\frac{2^h-1-k}{2^h-1})(2 + \sum_{i=0}^{k} \frac{C_i^k}{2^k}(L_2(h-1,i) + L_2(h-1,k-i))) & \text{otherwise} \end{cases} \quad (4)$$

In this equation, $(\frac{2^h-1-k}{2^h-1})$ denotes the probability that the root node is *not* ACTIVE. $\frac{C_i^k}{2^k}$ denotes the probability that the left subtree has exactly i ACTIVE nodes. $L_2(h-1,i)$ and $L_2(h-1,k-i)$ are the average number of links after pruning in the left and right subtrees respectively and are used recursively in the above formula.

Simulation results show that this analysis compares well with the simulated value of the number of messages. The above analysis can be easily extended to graphs with average degree $d > 3$.

5 Simulation

We studied the performance and behavior of our protocol using simulations. A simulation platform was developed using OPNET. For each of the experiments, we generated 200 random graphs and measured the various parameters of interest like the number of messages and the join latency. Random graphs are generated by randomly placing the specified number of nodes on a Cartesian plane of size 100x100. Links are added between two nodes u and v with a probability $P(u,v)$. $P(u,v)$ is the Waxman's probability function[5] and is given by:

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

Here, $l(u,v)$ is the distance between the nodes and L is the maximum distance between any two nodes. α and β are parameters in the range (0,1]. A small value of α results in graphs with more links between closer nodes. A large value of β increases the average degree d of each node. Each link has a corresponding weight and a delay factor associated with it. The weight, in some sense, corresponds to the cost of using the corresponding link. We use the distance between the nodes as the weight of the link. Such graphs are called *distance graphs*. The delay factor assigned to the links are random. This factor is used in computing the join latency for analysis. The cost of our Steiner Tree is the sum of the weights of links that belong to the tree.

For each graph, the initial multicast group is empty. Random nodes are chosen to be added to the multicast group. Similarly, random nodes are chosen to be deleted from the multicast group. After every operation, the value of the number of active nodes is updated. For each join operation, the number of messages required to join the multicast tree and the join latency is computed. The probability of concurrent joins is relatively small. This keeps the number of RETRY messages very low.

Using this simulation setup, we first compared the theoretical, average case analysis with simulated results. For a 50-node graph with average degree d=3, the depth h is computed using Equation 1 as 5. Similarly for a 100-node graph, h is computed as 6. The average number of messages required before a node can join the multicast tree is computed using Equation 3 and also by simulations. The results are plotted in Figure 4. From the figure, we can see that there is a close match between the theoretical analysis and experimental results. The small discrepancies can be attributed to the fact that the theoretical analysis assumes a very simple model of the network and the shortest path tree.

We performed extensive simulations on random graphs generated by varying the number of nodes of the graph and the average degree of the nodes. The number of nodes in our simulation varied from 50 to 100. We measured the parameters of interest, namely the number of messages and the join latency. The results of our simulations are shown in Figures 5 and 6.

To study the effect of the average degree of the nodes on the parameters of interest, the experiments were performed by varying the average degree d of the nodes, keeping the number of nodes constant. We see that the number of messages increases as the degree increases. This is because the shortest path tree for a distance graph is wider and has a smaller depth. Therefore, pruning does not eliminate too many nodes. For a complete graph, pruning does not eliminate any of the nodes. Therefore, the number of

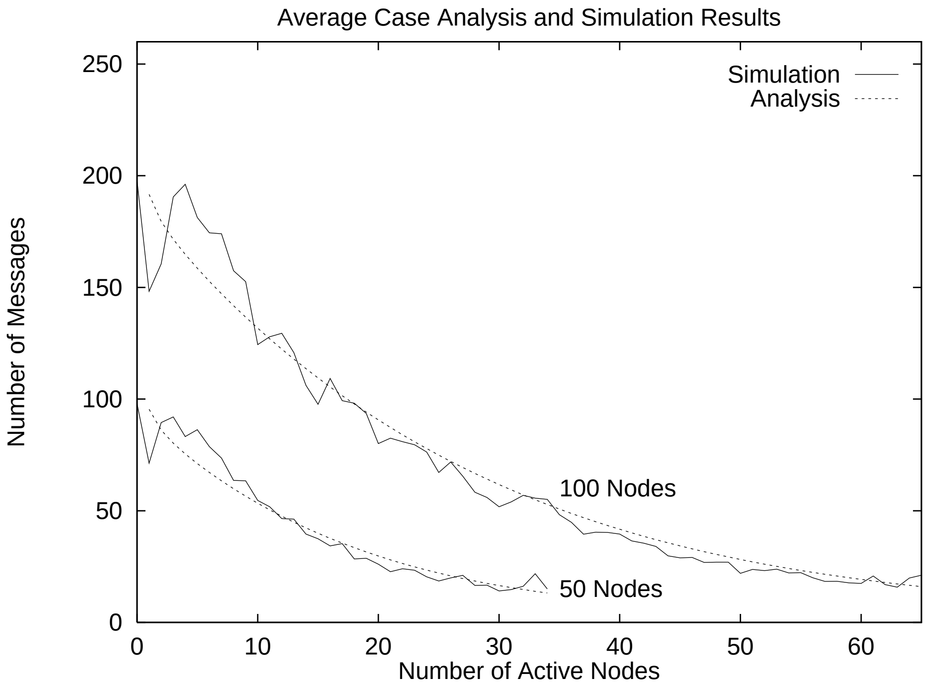

Figure 4: Comparison of theoretical analysis and simulations for $d = 3$

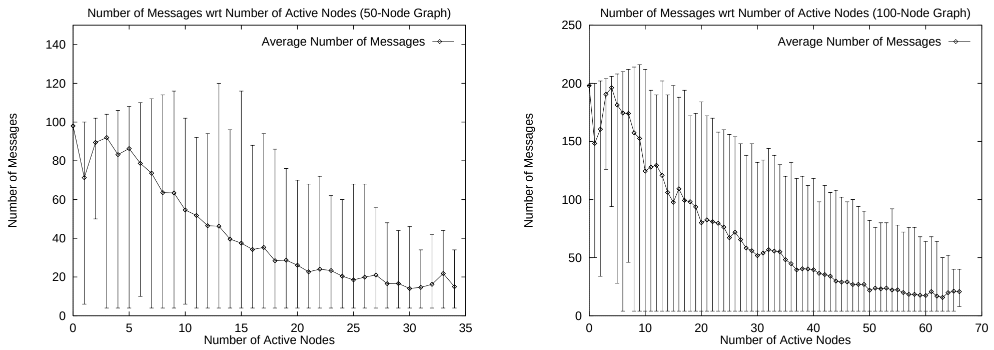

Figure 5: Number of Messages for (a) 50 Node and (b) 100 Node Graphs with $d = 3$

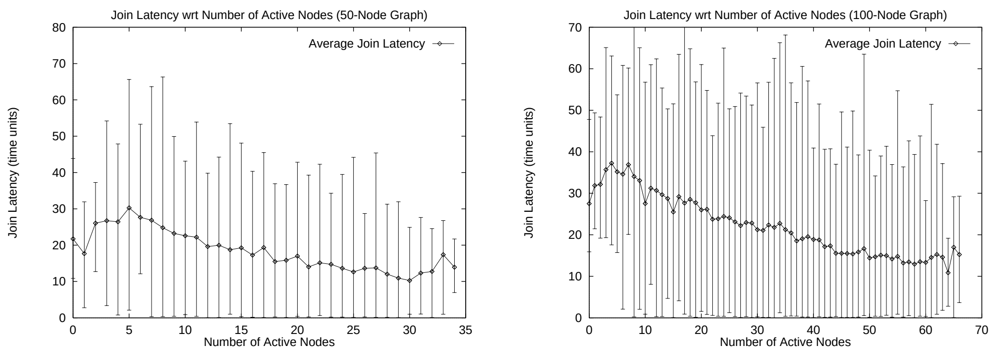

Figure 6: Join Latency for (a) 50 Node and (b) 100 Node Graphs with $d = 3$

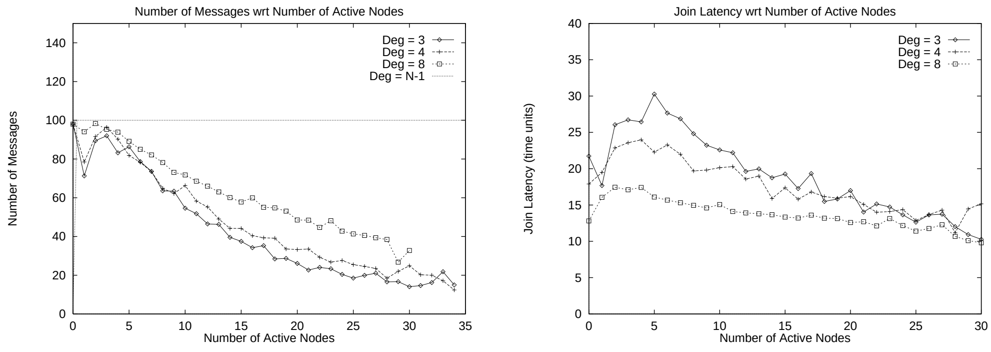

Figure 7: Dependence of the parameters on average degree of the node (50-Node graph)

messages required is a constant $2N$. Though the number of messages increases, more messages are sent simultaneously in parallel. Therefore, the join latency decreases with increase in the average degree of the node. This can be seen in Figure 7.

6 Conclusion

We have proposed a distributed protocol for computing efficient multicast trees with changing set of receivers. The protocol is deadlock-free and is a correct implementation of a centralized heuristic for the Steiner Tree problem. The protocol can be easily incorporated into the routing and signaling protocols under the PNNI framework. Therefore, it is scalable to large networks. The multicast trees computed are efficient in the utilization of network resources like bandwidth due to the sharing of links by various receivers. The bandwidth resource encompasses other traffic parameters, which contribute to the computation of effective bandwidth. The multicast data is sent to only those nodes that have receivers downstream or in their switching systems. Extensive simulations indicate that the number of messages and the join latency for a new node to join an existing multicast tree are small. This makes the DMRP protocol an attractive candidate to be included in the new version of PNNI protocols.

References

[1] T. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT) - An Architecture for Scalable Inter-Domain Multicast Routing. In *Proc. of the ACM SIGCOMM*, pages 85–95, Sept. 1993.

[2] F. Bauer and A. Varma. Distributed Algorithms for Multicast Path Setup in Data Networks. *IEEE/ACM Transactions on Networking*, 4(2):181–191, Apr. 1996.

[3] F. Hwang, D. Richards, and P. Winter. *The Steiner Tree Problem.* North-Holland, New York, 1992.

[4] The ATM Forum Technical Committee. *PNNI Specification Version 1.0*, 1996.

[5] B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, Dec. 1988.