A Distributed Delay-Constrained Dynamic Multicast Routing Algorithm

Quan Sun   Horst Langendőrfer

Institute of Operating Systems and Computer Networks
Technical University of Braunschweig
Bűltenweg 74/75, 38106 Braunschweig, Germany

Abstract

Many new distributed multimedia applications involve dynamic multiple participants, have stringent end-to-end delay requirement and consume large amount of network resources. In this paper, we propose a distributed delay-constrained dynamic multicast routing algorithm (DCDMR) to support these applications. DCDMR scales well because the source of the multicast tree needs only limited computation or may even not be involved in the route computation. When group membership changes, the existing multicast tree is perturbed as little as possible and the average resulting tree cost is very satisfactory.

1 Introduction

Many new distributed real-time applications of computer networks such as distance education and videoconferencing involve multiple participants, have stringent end-to-end delay requirement and consume large amount of network resources. In order to support these new applications efficiently, multicast routing algorithms computing least cost multicast trees that satisfy a given end-to-end delay constraint are desirable. Because the membership of the multicast group changes frequently in some applications (e.g. teleconferencing), the routing algorithms should also have the ability to alter an existing multicast tree to accommodate membership changes. Although a number of delay-constrained multicast routing algorithms (e.g. [1] [2] [3]) have been proposed in the past few years, there exist few delay-constrained multicast routing algorithms that explicitly support dynamic multicast groups. Some existing multicast routing protocols used in the Internet (MBone) can very well support dynamic multicast groups. However, the underlying multicast routing algorithms have been designed only for best-effort delivery.

In this paper, we propose a distributed delay-constrained dynamic multicast routing algorithm which perturbs the existing multicast tree as little as possible when group membership changes. The proposed algorithm can be easily incorporated into the distance vector based routing protocol [12]. The benefits of this algorithm are that a given end-to-end delay constraint is satisfied and efficient network resource utilization can be achieved. In the next section, we give our network model and the formal definition of the Delay-Constrained Dynamic Multicast routing problem (DCDM). In section 3 and 4, we give our solution for the DCDM and the simulation results. Section 5 concludes the paper.

2 Network Model and Problem Definition

A network is modeled as a connected, directed graph *G=(V,E)* where V is the set of nodes and E is the set of links. Two non-negative real value functions are associated with each link e ($e \in E$): delay *D(e)* and cost *C(e)*. The link delay *D(e)* is the delay a data packet experiences on the corresponding link, the link cost *C(e)* is a measure of the utilization of the corresponding link's resources. Links are asymmetrical, namely the cost and delay for the link *e=(i,j)* and the link $e^{'} = (j, i)$ may not be the same.

Given a source node s ($s \in V$), a set of destination nodes S ($S \subseteq V - s$), a tree T ($T \subseteq G$) rooted at s and spanning all of the nodes in S (with all leaf nodes $\subseteq S$) is called a *multicast tree*. The cost of tree T is defined as follows:

$$C_T = \sum_{e \in T} C(e)$$

The nodes in S are often called multicast *group members*, $|S|$ is the multicast group size. Let $P_T(s, v)$ denote the unique path from s to $v \in S$ on the multicast tree T, if the delays of all source-destination paths on the tree are kept within a given bound $\triangle$, namely:

$$\sum_{e \in P_T(s,v)} D(e) < \triangle \quad \forall v \in S$$

then tree T is called a *delay-constrained* multicast tree for the given delay constraint $\triangle$. The problem of finding a least cost delay-constrained multicast tree is NP-complete [1]

This paper focuses on the Delay-Constrained Dynamic Multicast routing problem (DCDM), this problem can be formally defined as follows: for a directed network *G=(V,E)* with non-negative link costs C and non-negative link delays D (note that C and D may change with time), a delay tolerance $\triangle$, the initial delay-constrained multicast tree T_0 with the source node s ($s \in V$) and a series of requests $\{r(t_1), r(t_2), ..., r(t_i), ...\}$ where $r(t_i)$ represents a request of adding a new group member or deleting an existing group member at time $t_i (t_1 \leq t_2 \leq t_3 \leq ...)$, find a series of delay-constrained multicast trees $\{T_1, T_2, ..., T_i, ...\}$ such that the members of T_i are those of T_0 modified by the requests $r(t_1), r(t_2), ..., r(t_i)$ and the cost of T_i is minimum among all possible choices for tree T_i.

A simple solution for DCDM is to reconstruct a new delay-constrained multicast tree using a static delay-constrained multicast heuristic like [2] for each request. Rebuilding a new tree in this way, however, may cause large disruptions for the current multicast sessions. This is especially unacceptable for real-time multimedia applications.

Most existing dynamic multicast routing heuristics consider only one link metric (link cost or link delay) [6] [7] [8] [9] [10]. The only known existing dynamic multicast routing algorithm that considers more than one link metric is the WAVE algorithm [11]. WAVE works as follows: when a new member v_n joins an existing tree, v_n contacts the source node s with a request Req. Starting from s, this request is propagated throughout the existing tree. When a node v_i on the tree receives a Req, it will send a response Rsp $= (v_i, cost, delay)$ to v_n along the shortest delay path from v_i to v_n, where *cost* and *delay* in Rsp are the cost and delay of the connection from s to v_n via v_i respectively. The node v_n will receive a set of responses from which it selects one Rsp $= (v_k, cost, delay)$ and then sends a connection request to node v_k that generated this response. The node v_k will extend the existing tree from itself to v_n along the shortest path from v_k to v_n. WAVE has the following drawbacks: (1) the number of responses could be very large [11]; (2) The source node is always contacted whenever a new node is added, such a solution would not scale well and (3) WAVE does not consider explicitly a given delay constraint.

In this paper, we propose a distributed delay-constrained dynamic multicast routing algorithm (DCDMR) which perturbs the existing tree as little as possible. For a delete request, DCDMR functions exactly like the GREEDY algorithm [6], namely the node to be deleted is simply marked as 'deleted' and if this node is a leaf node on the existing tree, the branch of which it is a part is pruned. In the next section, we will concentrate on how an add request is processed by DCDMR.

3 The Proposed Algorithm (DCDMR)

3.1 Routing information

We first discuss the routing information which needs to be available for the proposed algorithm to do the route computation. Each node in the network should know the delays of all outgoing links and must maintain a delay vector and a cost vector.

The delay vector at node $v_i \in V$ consists of $|V| - 1$ entries, one entry for each other network node. The entry for node $v_j \in V$ $(v_j \neq v_i)$ contains the following items:

- The destination node identity: v_j
- The end-to-end delay of the least delay path $P_{ld}(v_i, v_j)$ from v_i to v_j: $D(P_{ld}(v_i, v_j))$
- The cost of the least delay path $P_{ld}(v_i, v_j)$ from v_i to v_j: $C(P_{ld}(v_i, v_j))$

• The next hop node on the least delay path $P_{ld}(v_i, v_j)$ from v_i to v_j: $id(P_{ld}(v_i, v_j))$

The cost vector at node $v_i \in V$ also consists of $|V| - 1$ entries, one entry for each other network node. The entry for node $v_j \in V$ ($v_j \neq v_i$) contains the following items:

• The destination node identity: v_j
• The end-to-end delay of the least cost path $P_{lc}(v_i, v_j)$ from v_i to v_j: $D(P_{lc}(v_i, v_j))$
• The cost of the least cost path $P_{lc}(v_i, v_j)$ from v_i to v_j: $C(P_{lc}(v_i, v_j))$
• The next hop node on the least cost path $P_{lc}(v_i, v_j)$ from v_i to v_j: $id(P_{lc}(v_i, v_j))$

The above delay vectors and cost vectors are both similar to the distance vectors of the existing distance vector based routing protocol [12]. The same procedures for maintaining the distance vectors[1] can be used (and slightly extended) for maintaining the delay vectors and the cost vectors. Because our focus is on the routing algorithm, we will not discuss these procedures in this paper. In the following, we assume that the contents of the delay vectors and the cost vectors do not change during the execution of the routing algorithm.

3.2 The Delay-Constrained Unicast Routing Algorithm (DCR)

DCDMR is based on a distributed delay-constrained unicast routing algorithm (denoted by DCR) recently proposed by us [5]. For a source node s, a destination node d and a delay constraint $\triangle$, DCR constructs a path from s to d satisfying the given delay bound $\triangle$. In the following, we give a short description of DCR (for more information please see [5]).

DCR constructs the path one node at a time, from the source node s to the destination node d. If $D(P_{lc}(s, d)) < \triangle$, s reads the next hop node on the least cost path towards d, namely $id(P_{lc}(s, d))$, from its cost vector and sets:

• $next\_node = id(P_{lc}(s, d))$
• $path\_direction = LC$

otherwise, s reads the next hop node on the least delay path towards d, namely $id(P_{ld}(s, d))$, from its delay vector and sets:

• $next\_node = id(P_{ld}(s, d))$
• $path\_direction = LD$
• $delay\_so\_far = \mathrm{D}(s, id(P_{ld}(s, d)))$

where $\mathrm{D}(s, id(P_{ld}(s, d)))$ is the delay of the link $(s, id(P_{ld}(s, d)))$.
The source node s then constructs a *Path_construction* message and sends it to *next_node*. The *Path_construction* message contains the destination node d and the *path_direction*. If $path\_direction = LD$, the *Path_Construction* message

[1]These procedures are very simple: the contents of the vector at each node are required to be periodically transmitted to direct neighbors of that node only. As one of the reviewers has pointed out, the critical point for our case is how frequently the cost vector should be transmitted.

contains additionally the given end-to-end delay constraint $\triangle$ and *delay_so_far*, which represents the delay of the already constructed path from s to *next_node*.

When a node v ($v \neq d$) receives a *Path_construction* message, if *path_direction* in the received message is LC or if $delay\_so\_far + D(P_{lc}(v,d)) < \triangle$, node v reads the next hop node on the least cost path towards d, namely $id(P_{lc}(v,d))$, from its cost vector and sets:

- $next\_node = id(P_{lc}(v,d))$
- $path\_direction = LC$

otherwise, v sets

- $next\_node = id(P_{ld}(v,d))$
- $path\_direction = LD$
- $delay\_so\_far = delay\_so\_far + D(v, id(P_{ld}(v,d)))$

Node v then constructs a *Path_construction* message and sends it to *next_node*. Similarly, the *Path_construction* contains destination node d and *path_direction*. If $path\_direction = LD$, the *Path_Construction* message contains also the delay constraint $\triangle$ and *delay_so_far*.

If the destination node d receives the *Path_Construction*, it means that the delay-constrained path has been successfully constructed.

The shortest path algorithm in the existing distance-vector routing protocol [12] guarantees to find a loop-free path if the contents of the distance vectors at all nodes are up-to-date and when the network is stable. Under similar conditions, namely if the contents of the delay vectors and the cost vectors at all nodes are up-to-date and do not change during the path construction process, we have also [5]:

Theorem 1: The paths constructed by DCR are loop-free.

Theorem 2: DCR can always find a path from a source node s to a destination node d satisfying a given delay constraint if such a path exists.

Theorem 3: The worst case message complexity of DCR is $O(|V|)$, where $|V|$ is the number of network nodes.

Simulation results in [5] have also shown that DCR has a very good performance regarding the costs of the computed paths.

3.3 The Dynamic Multicast Routing Algorithm (DCDMR)

In the following, we let $D(P_T(s, v_i))$ denote the delay of the unique path from s to v_i on the existing tree T where s is the root (source node) of T and v_i is any other node on T. $D(P_T(s, v_i))$ should be known to node v_i and $D(P_T(s,s)) = 0$.

In DCDMR, when a new member v_n joins an existing tree, v_n contacts node v_p on the existing tree with a request $\mathsf{Req} = (v_n, mode)$ along the shortest delay path where v_p can be any node of T (favourably a node of T in the vicinity of v_n) and $mode$ can take one of the two values: FAST and SLOW. DCDMR can work in two different modes, namely the FAST mode or the SLOW mode.

<u>If the *mode* value in Req is FAST, DCDMR takes the following steps:</u>

(F1) Set $v_i \leftarrow v_p$.

(F2) If $v_i = s$ and $D(P_{ld}(s, v_n)) \geq \triangle$, there exists no delay-constrained path from s to node v_n and DCDMR terminates (s may send a "No path found" message to v_n along the shortest delay path); otherwise go to next step.

(F3) If $D(P_T(s, v_i)) + D(P_{ld}(v_i, v_n)) < \triangle$, node v_i computes a path from itself to the node v_n satisfying the delay constraint of $\triangle - D(P_T(s, v_i))$ using DCR (see section 3.2); otherwise go to next step.

(F4) v_i sends the received Req to v_i's parent node v_q on the existing tree (note that according to the following Lemma 1 we should not send Req to v_i's child node(s) on the existing tree). Set $v_i \leftarrow v_q$ and go to (*F2*).

If the *mode* value in Req is SLOW, DCDMR takes the following steps:

(S1) Set $v_i \leftarrow v_p$.

(S2) If $v_i = s$ and $D(P_{ld}(s, v_n)) \geq \triangle$, there exists no delay-constrained path from s to node v_n and DCDMR terminates (s may send a "No path found" message to v_n along the shortest delay path); otherwise go to next step.

(S3) If $D(P_T(s, v_i)) + D(P_{ld}(v_i, v_n)) \geq \triangle$, node v_i sends the received Req to its parent node v_q on the existing tree (note that according to the following Lemma 1 we should not send Req to v_i's child node(s) on the existing tree), set $v_i \leftarrow v_q$ and then go to (*S2*); otherwise go to next step.

(S4) Node v_i constructs a response Rsp $= (v_n, mode, node, cost)$, where *mode*=SLOW, *node*=v_i and *cost* is determined as follows: if $D(P_T(s, v_i)) + D(P_{lc}(v_i, v_n)) < \triangle$, *cost* is set to be $C(P_{lc}(v_i, v_n))$; else *cost* is set to be $C(P_{ld}(v_i, v_n))$. The Rsp is then sent to v_i's parent node and all v_i's child nodes on the existing tree (if $v_i = s$, Rsp is sent to s's child nodes only). When a node v_j on the tree receives the Rsp, it checks

$$D(P_T(s, v_j)) + D(P_{ld}(v_j, v_n)) < \triangle \qquad (1)$$

If (1) is not satisfied, according to the following Lemma 1, the Rsp must come from v_j's parent node on T, and Rsp is sent to v_n along the shortest delay path. If (1) is satisfied, v_j determines the cost $C(v_j, v_n)$ of the path from itself to node v_n in the following way: if $D(P_T(s, v_j)) + D(P_{lc}(v_j, v_n)) < \triangle$, $C(v_j, v_n)$ is set to be $C(P_{lc}(v_j, v_n))$; otherwise $C(v_j, v_n)$ is set to be $C(P_{ld}(v_j, v_n))$. The originally received Rsp=$(v_n, mode, node, cost)$ will be modified if $C(v_j, v_n) < cost$. In this case, *node* is set to be v_j and *cost* is set to be $C(v_j, v_n)$. Rsp is then sent to all v_j's adjacent nodes (parent/child nodes) on the existing tree except the node from which v_j received the original Rsp.

Node v_n will receive a set of responses from which it selects the one with the least *cost* value. Let this selected Rsp be $(v_n, mode, v_k, C_k)$. Node v_n then sends a connection request to node v_k that generated this response. Node v_k will compute a path from itself to v_n that satisfies the delay constraint of $\triangle - D(P_T(s, v_k))$ using DCR.

Lemma 1: Let s denote the root of the existing tree T, v_i any non-leaf node on T, v_j any one of v_i's child nodes on T and v_n the new node to be added. If $D(P_T(s, v_i)) + D(P_{ld}(v_i, v_n)) \geq \triangle$, then $D(P_T(s, v_j)) + D(P_{ld}(v_j, v_n)) \geq \triangle$.

Lemma 2: When DCDMR works in the SLOW mode, the largest possible number of responses received by the new node to be added is $L + 1$, where L is the number of leaf nodes on the existing tree.

The proofs of Lemma 1 and Lemma 2 are omitted here. The above distributed algorithm computes a path from the existing tree to the new node to be added. The worst case message complexity for the SLOW mode is $O(|V|^2)$ and the message complexity for the FAST mode is $O(|V|)$. The computed path and the existing tree can be easily combined to give a new delay-constrained multicast tree. The message complexity of this combination process is $O(|V|)$.

Compared with WAVE, our DCDMR algorithm working in the SLOW mode has the following advantages: (a) the number of responses sent to the new node could be much smaller; (b) In most cases, the source node will send Rsp to its child node(s) instead of the new node to be added, therefore less computation at the source node is needed in DCDMR than in WAVE. Furthermore, some nodes on the tree can stop propagating the Rsp to other tree nodes (for example the source node) and simply send the Rsp directly to the new node and therefore relieve other tree nodes from doing the route computation. (c) DCDMR is specifically targeted to consider a given delay constraint. If DCDMR works in the FAST mode, even less computation at tree nodes is needed and the route computation is very fast.

The DCDMR algorithm is very flexible in the route computation. For example, if a node finds that its multicast capability is depleted (e.g. it can provide only a small degree of multicast) or if not enough resources at the node are available, it can simply send the received Req or Rsp forward to other tree nodes immediately.

4 Performance Evaluation

We have used simulation experiments to evaluate the cost performance of the proposed algorithm and have run our simulations on the multicast routing simulator MCRSIM [4] developed at North Carolina State University. MCRSIM simulates actual ATM networks with fixed cell sizes. The simulator implements a modified version of the random network generator described in [6]. In MCRSIM, each network node represented a non-blocking ATM switch. Each link was assumed to have a small output buffer. The propagation speed through the links was taken to be two thirds the speed of light. The propagation delay was dominant under these conditions, and the queueing component was neglected when calculating the link delay *D(e)*. In the simulation, the link cost *C(e)* was set to be the total currently reserved bandwidth on the link. Because the NAIVE algorithm [9] is the only known dynamic multicast algorithm that can compute trees satisfying a given delay constraint (note that NAIVE uses only one link metric) and the bounded shortest multicast algorithm (BSMA) [2] has the best cost performance among all the proposed delay-constrained static multicast heuristics [4], both NAIVE and BSMA were included in the simulation for comparison. In the simulation, the original request Req of the node to be added was sent to the nearest node on the existing tree.

For each run of the experiment we generated a random network with homogeneous link capacities of 155 Mbps and an average node degree of 4. Random background traffic for each link of the network was also generated. The equivalent bandwidth of each link's background traffic was a random variable uniformly distributed between 10 Mbps and 120 Mbps. We then generated an initial multicast tree T_0, which consists of only a randomly generated source node s, and a random series of requests $\{r_1, r_2, ..., r_m\}$. Because there are situations for some applications in which members rarely leave the added group until the end of the application, we first consider the case where every r_i (i=*1,2,...,*m) is an add request (note that $r_i \neq r_j$ if $i \neq j$ for i,j=*1,2,...,*m). The algorithm DCDMR or NAIVE was applied to generate a new delay-constrained multicast tree T_1 for the request r_1 joining T_0, T_2 for the request r_2 joining T_1, ..., T_m for the request r_m joining T_{m-1}. The BSMA algorithm was used directly to generate the tree T_m for the source node s and all the m members to be added without considering the request sequence of the nodes to be added. The experiment was run repeatedly until confidence interval of less than 5% using 95% confidence level was achieved for the cost of the tree T_m. Fig.1(a) shows the percentage excess cost over BSMA versus number of network nodes for the number of add requests of 15 and the delay bound of 50 *ms*. Fig.1(b) shows the percentage excess cost over BSMA versus number of add requests for the number of network nodes of 60 and the delay bound of 50 *ms*. From these figures, we see that the DCDMR in the SLOW mode has a cost performance that is always within 10% worse than BSMA, while the DCDMR in the FAST mode and the NAIVE are up to 40 % and 70% worse than BSMA respectively.

Next, we consider the situation where the request may be *add* or *delete*. To determine whether the next request is *add* or *delete*, we compute the following function (similar like that introduced in [6]):

$$P_a(k) = \frac{\alpha(n-k)}{\alpha(n-k)+(1-\alpha)k}$$

where n is the number of network nodes, k is the number of *group* nodes on the current tree and α is the fraction of group nodes on the tree at equilibrium. We then compute a random number r $(0 < r < 1)$. If $P_a(k) < r$, the new request is *delete* and randomly one of the group members on the current tree is determined as the node to be deleted; if $P_a(k) \geq r$, the next request is *add* and a non-group member is randomly selected as the node to be added.

In the simulation, we set α=0.15. Similarly, for each run of the experiment

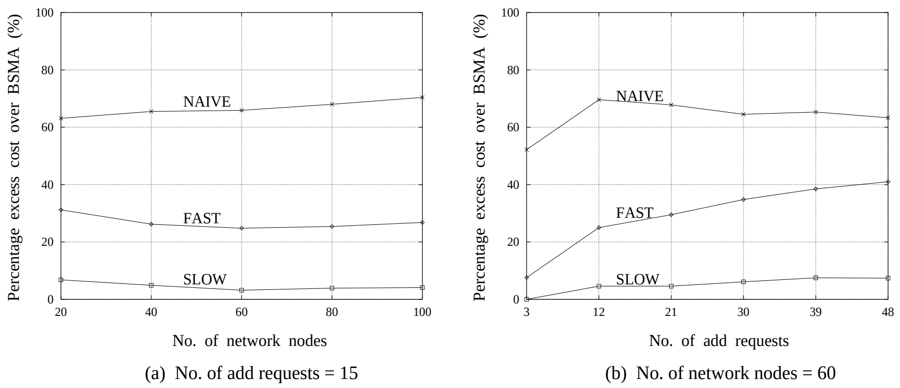

Figure 1: (a) percentage excess cost over BSMA versus number of network nodes and (b) percentage excess cost over BSMA versus number of add requests, for delay bound=50 ms

we generated a random network with an average node degree of 4 and random background traffic for each network link. The initial delay-constrained multicast tree T_0 was the resulting tree after processing a random series of 15 add requests for an initial empty tree consisting of only one randomly generated source node (for BSMA T_0 was generated without considering the request sequence). We then generated a series of requests $\{r_1, r_2, ..., r_m\}$ using the method described above. We recorded the cost of the resulting tree after processing the requests $r_1, r_2, ..., r_m$ for DCDMR and NAIVE and the cost of the tree computed using BSMA directly for those group nodes after the request r_m. The experiment was run repeatedly until confidence interval of less than 5% using 95% confidence level was achieved for the recorded cost. Fig.2(a) and Fig.2(b) show the comparison of the cost of the resulting tree obtained by BSMA and that obtained by DCDMR and NAIVE (note that the tree for DCDMR and NAIVE was dynamically evolving with each request). Fig.2(a) shows the results when the number of requests is small and Fig.2(b) shows the case for large number of requests. From both figures it is easy to see that the cost inefficiency for DCDMR in both FAST and SLOW mode increases slightly with the number of requests when the number of requests is small (see Fig.2(a)). For large number of requests ($\geq$50), the cost inefficiency for DCDMR remains at about the same level. The cost inefficiency for NAIVE has not changed much for any number of requests. However, NAIVE has a cost performance that is always much worse than DCDMR. NAIVE is at least 20% worse than DCDMR in the FAST mode, which in turn is more than 20% worse than DCDMR in the SLOW mode for small to several hundred number of requests in our simulation.

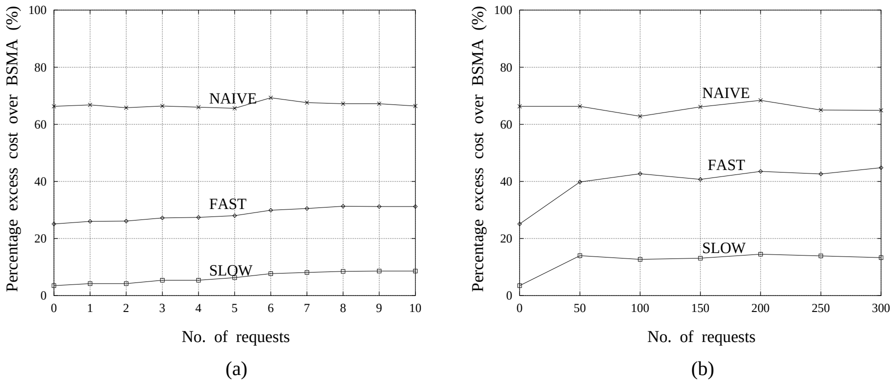

Figure 2: Percentage excess cost over BSMA versus number of *add/delete* requests, for number of network nodes=60 and delay bound=50 *ms*

5 Concluding Remarks

In this paper, we proposed a distributed delay-constrained multicast routing algorithm (DCDMR) supporting dynamic multicast groups. DCDMR perturbs the existing multicast tree as little as possible when group membership changes. DCDMR scales well because the source node may not be involved in the route computation or needs only small computation. DCDMR has also very good cost performance, simulation results show that DCDMR in the FAST mode is at least 20% better than NAIVE, and DCDMR in the SLOW mode is at least 20% better than DCDMR in the FAST mode. When DCDMR works in the FAST mode, the route computation is very fast; when it works in the SLOW mode, very low cost trees can be computed.

There are a few areas for further research: (a) mechanisms that enable the proposed algorithm to cope with dynamic network environment, (b) the interaction with the routing protocols and resource reservation components, (c) how to deal with the situation where network nodes enter and leave the multicast group simultaneously, etc.

References

[1] Kompella, V.P., et al., Multicast Routing for Multimedia Communication, *IEEE/ACM Transactions on Networking*, 1(3),pp.286-292, 1993

[2] Zhu, Q., et al, A Source-Based Algorithm for Near-Optimum Delay-Constrained Multicasting, *IEEE INFOCOM'95*, pp.377-385, 1995

[3] Wi, S. and Choi, Y., A Delay Constrained Distributed Multicast Routing Algorithm, *12th International Conference on Computer Communication, ICCC'95*, pp.833-838, 1995

[4] Salama, H.F., et al, Evaluation of Multicast Routing Algorithms for Real-Time Communication on High-Speed Networks, in *High Performance Networking VI, IFIP 6th International Conference on High Performance Networking*, pp.27-42, 1995 (the multicast simulator MCRSIM can be downloaded from ftp://ftp.csc.ncsu.edu/pub/rtcomm/)

[5] Sun, Q. and Langendőrfer, H., A New Distributed Routing Algorithm for Supporting Delay-Sensitive Applications, Internal Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany, March 1997 (available from: http://www.cs.tu-bs.de/~sun/dunicast.ps)

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

[7] Kadirire, J. and Knight G., Comparison of Dynamic Multicast Routing Algorithms for Wide-Area Packet Switched (Asynchronous Transfer Mode) Networks, *IEEE INFOCOM'95*, pp.212-219, 1995

[8] Bauer F. and Varma, A., ARIES: A Rearrangeable Inexpensive Edge-based On-line Steiner Algorithm, in *proceedings of GLOBECOM'95*, 1995

[9] Doar, M. and Leslie, I., How Bad is Naive Multicast Routing, *IEEE INFOCOM'93*, pp.82-89, 1993

[10] Effelsberg, W. and Mueller-Menrad, E., Dynamic Join and Leave for Real-Time Multicast, Tenet Report TR-93-056, Computer Science Division, University of California at Berkeley, Oct. 1993

[11] Biersack, E. and Nonnenmacher J., WAVE: A New Multicast Routing Algorithm for Static and Dynamic Multicast Groups, in *proceedings of 5th Workshop on Network and Operating System Support for Digital Audio and Video*, 1995

[12] Hedrick, C, Routing Information Protocol, Internet RFC 1058, June 1988