A Tree Building Technique for Overlay Multicasting in DiffServ Domains

Baijian Yang, Abdol H. Esfahanian, Lionel Ni
Department of Computer Science and Engineering
Michigan State University, East Lansing, MI

Prasant Mohapatra
Department of Computer Science
University of California, Davis, CA

***Abstract*— Overlay multicasting technology is now considered as a promising alternative to the native IP multicasting. While many studies are currently focused on how to build an overlay tree/mesh on top of conventional networks in a self-organized fashion, only limited attention has been directed toward QoS provisioning. QoS can be provisioned by leveraging the implementation of differentiated services in the Internet. In this paper, we have proposed an incremental insertion tree-building algorithm to avoid extensive signaling overhead in differentiated service domains. The simulation results depict that on an average our approach produces a tree with a cost of only 10% to 20% more than the fully cost-optimized tree. The relative delay penalties generated by the incremental insertion algorithm is comparable to that of the previously proposed Narada scheme. Although our scheme is designed for differentiated services, the algorithm can also be adapted to integrated services as well.**

Keywords: Differentiated Services, Incremental Insertion Algorithm, Overlay Multicast, Quality of Service.

I. Introduction

It has been widely accepted that network layer is the appropriate place to efficiently support multicast services. However, the deployment of IP multicast still advances in a slow pace even after over a decade of efforts. A number of reasons can be attributed to this slowness. First of all, IP multicast requires routers to maintain per group state, which is fundamentally at odds with the conventional stateless internet infrastructure, Second, IP multicast routing look-up entries are hard to be aggregated. Third, limited IP multicast address space also hinders its deployment. Furthermore, IP multicast lacks flow control and authentication mechanisms, which further slow down its process of being commercialy deployed.

This research was supported in part by the National Science Foundation through the grants CCR-0296070 and ANI-0296034.

To get around the inherent difficulties involved in deploying IP multicast, an alternative solution which is termed as *overlay multicasting* or *end-system multicasting* has received a lot of attention. The overlay multicasting approach assumes no multicasting support in the network layer, and constructs a multicast delivery tree in the application layer. An intuitive comparison of IP multicast, multiple unicast and overlay multicast is illustrated in Fig. 1.

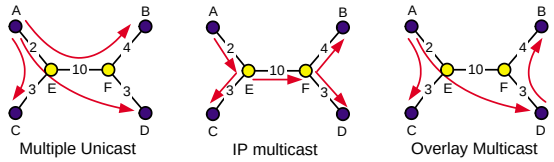


Fig. 1. Illustration of unicast, IP multicast and Overlay multicast.

In Fig. 1, host A is the sender while host B, C and D are receivers. Node E and F are routers. For the traditional unicast scheme, three identical copies of the data will be sent from A to B, C and D. While using IP multicast approach, only one copy of data is sent on each on-tree link, i.e., A $\rightarrow$ E $\rightarrow$ C, E $\rightarrow$ F $\rightarrow$ D, and F$\rightarrow$ B. In overlay multicast scenario, the multicast tree is built at the application layer. Sender A sends two identical copies to C and D. Upon receiving, D makes a copy and forwards it to B through links D $\rightarrow$ F $\rightarrow$ B. If we ignore the network and application layer overheads and let the number on each link denote link cost, then the total cost of IP multicast and overlay multicast are 22 and 27, respectively, while that of unicast is 38.

The advantages of overlay multicast includes easier deployment, better scalability and support of higher layer functionalities, such as security and congestion control. On the other hand, it is less efficient in terms of network resource usages and leads to longer transmission delay.

Provisioning QoS in overlay multicasting is

helpful for several reasons. First, typical multicast applications, such as video or audio conferencing, distant learning, include multimedia data stream transportation, and have certain bandwidth, delay or jitter requirements. Second, we argue that QoS provisioning in overlay multicasting is not merely an add-on service which only improves the quality of multicasting traffic. Rather, it helps to maintain a more stable overlay multicast delivery tree. For example, due to the dynamic changes in network conditions and group membership, previously proposed self-organized overlay multicast protocols [4], [6] use adaptive mechanisms to re-construct delivery trees by periodically evaluating the cost, delay, and available bandwidth of the links among each active end hosts. If the underlying networks have certain QoS support to guarantee or assure traffic on every on-tree multicast link, then such an overlay tree structure will be much less sensitive to the changes in network conditions. Ideally, with the help of network layer QoS support, overlay multicast trees only need to change when there are changes in group membership or node/link failures. In this context, many bandwidth or delay probing packets could be eliminated and the maintenance of overlay trees could be simplified.

Two fundamental solutions for supporting network layer QoS are Integrated Services (IntServ) [1] and Differentiated Services (DiffServ) [2]. In this paper, we focus on DiffServ architecture because of its scalability and ease of deployment. However, the tree building algorithm proposed in this paper could also be applied to IntServ as well.

The rest of paper is organized as follows. The Network models and design issues are outlined in Section II followed by a brief review of related works in Section V. Section III describes our proposed tree building technique and its performance is evaluted in Section IV. The paper concludes in Section VI.

II. Network Models and Design Issues

DiffServ architecture offers a number of features that will affect the design of overlay multicasting. We will briefly review the major characteristics of DiffServ model, and then address the design issues.

A. DiffServ Architecture and its Features

In DiffServ architecture, packets are differentiated into a series of QoS levels at the edge routers by marking the TOS byte of their IP headers. Packets are then grouped and mapped onto corresponding traffic aggregation. During transmission, higher level traffic aggregation will get better forwarding treatments including more bandwidth, less queuing delay, or both. When network congestion occurs, mission critical traffic can thus be protected in DiffServ environment by more aggressively dropping low priority packets. To avoid lower priority class of traffic from starvation, Service Level Agreements (SLAs) should be set up between the host domain and the servicing domain. Traffic that exceeds the specified SLA is termed as out-of-profile. It will be either marked as low priority or discarded. IETF has defined two major classes of forwarding behaviors: Expedited Forwarding (EF) and Assured Forwarding (AF). The former provides hard guarantee while the latter offers a less expensive better than Best Effort (BE) service.

When designing a scheme to support overlay multicasting in DiffServ domains, we should be aware of the following characteristics of DiffServ:

- *SLAs.* As mentioned above, there is no service guarantee for out-of-profile traffic. The fanout of each end host should be bounded by its maximum amount of traffic specified in SLAs, which is less than the actual available network bandwidth. As and when necessary, SLAs can be renegotiated.
- *Signaling overheads.* In DiffServ domains, dedicated nodes, termed as Bandwidth Brokers (BBs), are responsible for any intra-domain and inter-domain resource management. According to current QBone BB design[10], end hosts need to request BBs for resource allocation to ensure that SLAs are not violated and sufficient resources are available along the path. It implies that every link change in the overlay multicast tree may initiate a BB signaling.
- *Uni-direction.* DiffServ only provides QoS for one way traffic. In order to get a bi-

directional DiffServ link, both participants should signal their BBs for resource allocation. As the actual SLAs and network utilization could vary, such bi-directional links could fail. Thus, constructing a shared tree in DiffServ may not be feasible.

B. Design Issues

In this paper, we assume that the underlying network can provide either premium service of DiffServ or guaranteed service of IntServ. The overview of design issues follows.

- *Changes in tree topology.* Since every change of the tree topology would lead to BB signaling, the tree-building algorithms should construct a tree in a way such that it could avoid extensive topology changes when group members or network condition changes.
- *Source specific tree.* Since DiffServ is unidirectional, we prefer to construct per-sender-based multicast trees.
- *Direct tree construction.* There are two distinct approaches to construct overlay trees. One approach is to first set up a richer connected graph, termed as *mesh*. A tree is then constructed using DVMRP like routing protocol. Mesh first approach is good for constructing shared trees and is suitable when the underlying networks performance or reliability is undesirable. But it would be harder to detect mesh partitions, and resulting tree topology may changes dramatically from time to time. For the above considerations, we have chosen the other scheme, i.e. to build the tree directly.
- *Sub-optimized for tree cost.* We favor minimal or sub-optimal tree-cost. Saving overall network resources is one of the major strengths of multicast. However, fully optimized schemes, such as Minimum Spanning Tree, are less likely to be implemented in practice, and would produce a deeper tree with longer latency.
- *Incremental join at proper place.* Most overlay multicast protocols are self-organizing. It is a desirable feature when there are no service guarantees and no signaling overheads associated with the changes in the tree topology. In this paper, we present an approach that tries to insert a new member at the appropriate place of the tree, while considering the SLAs as well as the overheads associated with topology changes.

C. Performance Metrics

In summary, the solution we are seeking should yield fewer tree topology changes and have a good balance between overall tree cost and individual latency. In this paper, we use the following performance metrics to evaluate different approaches:

- *Tree Cost.* Tree cost is defined as the sum of the link costs. If we ignore the network layer and application layer overheads, this metric reflects the efficiency of system resource usage.
- *Link Changes.* The notation of link changes refers to the number of different links between the original tree and the new trees built after new members join. In DiffServ overlay multicasting environment, link changes incur overheads associated with BB signaling.
- *Relative Delay Penalty (RDP).* RDP is denoted as the ratio of overlay multicast delay to the unicast shortest-path delay. This metric is defined from the perspective of each host.

III. Group Management and Incremental Insertion Protocol

To support overlay multicasting in DiffServ domain, there could be two approaches. One approach is to make QoS provisioning totally independent of tree construction algorithms. That is, after the tree is constructed, each member host could signal the edge router for appropriate packet marking. However, it may happen that by then some of the nodes would have consumed all of the resources specified in their SLAs. The marker would then mark the multicast packets as out-of-profile. Unless the SLAs are renegotiated successfully, the QoS guarantee on certain branches would be compromised, and the unfairness among each multicast member would be aggravated. Thus we adopt another approach by taking each members available SLAs into account while building the tree.

A. Group Management Overview

In order to achieve better scalability, we propose a two-level hierarchical scheme which includes inter-domain and intra-domain group management. Logically, a tree topology for a cross-domain multicast group G with sender S, denoted as (S,G), is shown in Fig. 2. At inter-domain level, each domain except the source domain is assumed to have exactly *one* link connected with its parent domain through which the multicast tree is formed. There may or may not be other links connecting the domains, but are not used for multicasting. For example, link RA3 → RB1 is the cross-domain link which hooks up domain B with its parent domain A. In the source domain, the sender is the domain *multicast root*. In other domains, the node responsible for receiving inter-domain traffic is denoted as the domain *multicast root*. In Fig. 2, the domain multicast root for (S,G) at domain A, B, C and D are S, RB1, RC1 and RD1, respectively. It should be noted that the term *domain* refers to the actual DiffServ domain as defined by IETF. We do not propose the dynamic clustering approach suggested in [8], [9], because in DiffServ, inter-domain resources could be far more scare and its resource management is much more complicated. If the nodes from different DiffServ domains self-organize to form a *cluster* as the basic tree building block, then the resource management within each cluster would be complex.

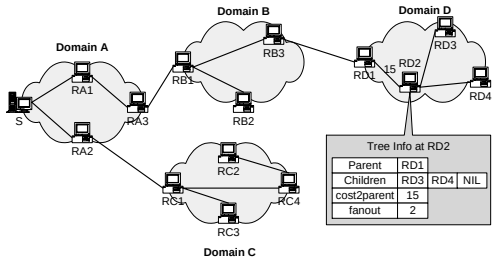


Fig. 2. Logical view of tree topology for (S,G).

In this paper, we focus on studying intra-domain group management and tree construction techniques. We further assume that certain bootstrap mechanisms exist such that any node j desiring to join a group (S,G) could obtain the address of the *multicast root* in its domain, and from which it could get a list of active members in the domain.

B. Incremental Insertion Algorithm (IIA)

We have proposed an *Incremental Insertion Algorithm* for building overlay multicast trees in a DiffServ Domain by constructing the multicast tree incrementally. Thus simultaneous joins should be serialized and then be processed one by one. This serialization could be accomplished by maintaining a queue at a dedicated node such as the domain multicast root.

As depicted in Fig. 2, each active member in the multicast tree should maintain its parent node, cost to its parent, children nodes and the number of its children (denoted as *fanout* in this paper). When node j wants to join a multicast group (S,G), it will first evaluate its cost to all the active members in the domain and then calculate the cost gain for two cases. One case is to attach itself to node i and become a new leaf in the tree, which is denoted as *leaf-join*. The other case is called *insertion*, that is inserting node j between node i and and i's parent node. For instance, in Fig. 3.a, node j is evaluating its cost gain toward node i. Leaf-join would produce a new tree structure as shown in Fig. 3.b, and Fig. 3.c depicted the new tree structure if node j is inserted between i and $i.parent$. If numbers in the figure denote costs of the links, the cost gain for leaf-join will be 4, while insertion would be 2. Since insertion cost less, node j will evaluate cost gain toward node i as 2, with an operation *insertion*. After node j goes through all the active members, it will join to the node having minimum cost gain with the corresponding operation. The formal presentation of the algorithm is shown in Algorithm 1.

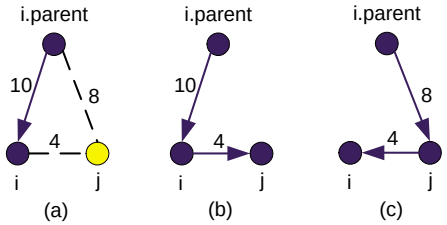


Fig. 3. Illustration of *leaf-join* and *insertion*.

One of the problems associated with the IIA is that it may increase the join latency. We have defined a non-QoS working mode to allow a new member to receive multicast traffic as quick as possible by joining a random node in the tree. The new

Algorithm 1 Incremental Insertion Algorithm at node *j*

Initialization: $j.cost2parent \leftarrow \infty, j.parent \leftarrow NIL, j.child \leftarrow NIL, j.fantout \leftarrow 0$ {For off-line mode only}
  for every active member i **do**
    if $i.fanout < FANOUT$ and $j.cost2parent > cost(i,j)$ **then**
      $j.cost2parent \leftarrow cost(i,j)$
      $j.parent \leftarrow i$
      $mode \leftarrow leaf\_join$
    end if
    if $i.fanout < FANOUT$ and $i.parent.fanout < FANOUT$ **then**
      $insertCost \leftarrow (cost(j,i) + cost(i.parent, j) - i.cost2parent)$
      if $insertCost < j.cost2parent$ **then**
        $j.cost2parent \leftarrow insertCost$
        $j.parent \leftarrow i.parent$
        $j.child \leftarrow i$
        $mode \leftarrow insertion$
      end if
    end if
  end for
  add j as a new child of $j.parent$
  $j.parent.fanout++$
  if $mode = insertion$ **then**
    remove $j.child$ from $j.parent$
    $j.parent.faout--$
    add $j.child$ as a new child of j
    $j.fanout++$
  end if

member will leave the non-QoS mode after it finishes processing Algorithm 1.

C. Partition Repair

There are two situations that would create tree partition problem: member leaves or a link/node fails. In both cases, the tree should be reconstructed. If we assume single point failure, it could be detected if every node periodically 'ping' its parent. Member leaving can be also handled in same way. But it would be quicker if we let the leaving node send a message to its children to trigger the re-building process.

We have proposed both on-line mode and off-line modes of repairing processes. The former aims at fast repairing, but would generate a tree with worse performance. The later process could produce a tree with better performance, but is only suitable when there is no multicast traffic on the flight. Both repairing algorithms are straightforward. For example, in Fig. 4a, node i leaves tree T. In on-line mode, i's children $C1$ and $C2$ will execute a slightly modified IIA by considering $C1$ and $C2$ as two nodes wanting to join the tree T. The only modification is that when a node joins during the on-line repairing mode, it should not go through the initialization part. A possible on-line repairing result is shown in Fig. 4.b. We can see that the on-line repairing mode would not change the trees rooted at i's direct children $C1$ and $C2$, and only require 2 link changes in the example. On the other hand, in off-line mode, all the descendants of node i will be treated as new nodes request to join and will carry out Algorithm 1 one by one. Such repairing algorithms could produce a quite different tree structure shown in Fig. 4.c, where the trees originally rooted at $C1$ and $C2$ are not retained.

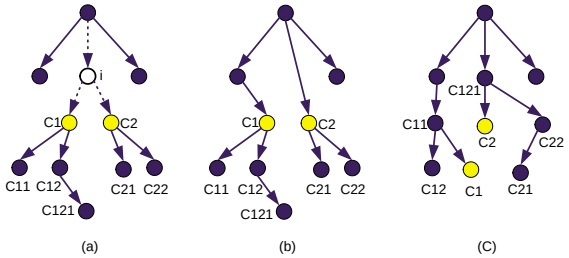


Fig. 4. On-line mode and off-line mode repair

IV. Simulations Results

In our simulation, we use delay as the metric of link cost. Three types of network topologies, i.e. Waxman1, Waxman2, and Locality model, are created by using GIT network topology generator [3]. Each type of network has 100 random graphs, and each graph consists of 1024 node, with cost distributed in a 100x100 plane. For every graph, we randomly pick a non-tree node and vary the group size from 2 to 100. The maximum fanout of any member is set to 6.

The proposed scheme is compared to Narada tree, minimum-cost tree, and the minimum-leaf join tree. In the implementation of Narada, we use 0.75 as the threshold of utility gains for link addition, and choose 0.25 as the lower bound of consensus cost to drop a link. After each member joins, we allow Narada tree to stabilize before processing the next new member.

The performance metrics we used is described in Section II-C. Since the results are similiar for all three different type of networks, we present the results of Waxman2 topology model in this paper. Table I summarizes the statistic of the results while the average results are ploted in Fig. 5

A. Tree Cost

Narada performs poorly for tree cost because it produces a tree in a DVMRP fashion, which is not optimized for overall cost. Another reason is that the mesh that Narada constructs is on a shared-tree basis. So the quality of the mesh may not be good for a source-specific tree. In Fig. 5.a, Narada's tree-cost is more than two times of that of the optimal case. As expected, minimum-cost leaf-join is worse than the IIA approach. Intuitively, IIA can be conceived as another optimization on top of *leaf-join* because it considers both *leaf-join* and *insertion*. The results show that the average tree cost of IIA is roughly 10% more than the optimal case, whereas the average tree-cost of minimum-cost leaf-join is around 20% to 30% more than the optimal case.

B. Link Changes

It is clear from Fig. 5.b that both Narada and the optimal tree cause extensive link changes. It is not a surprise since both of them do not consider link change as an optimization objective. Narada is even worse than the optimal tree in our results. One reason for this behavior is that the mesh is always changing, which makes the corresponding tree topology change more aggressively. Another reason is probably because of the threshold value we chose in the simulation, which may not be good enough. Minimum-cost leaf-join demonstrate the best performance in terms of this metric. The average link changes is equal to $n-1$ for IIA, where n is the total number of members. Although IIA causes nearly two times as many link changes as that of the minimum-cost leaf-join, it still is a linear function of group members (n) with the upper bound as $3n-3$.

C. Cumulative Relative Delay Penalties (RDP)

Optimal-cost tree incurs the lowest RDP as shown in Fig. 5.c. and Table I. Narada has slightly less average RDPs than IIA up to the 80th percentile. But its average is a little higher than that of IIA. MOreover, its worst case RDP is much larger. The possible explanation lies in Narada's shared mesh, which implies that shared delay optimized tree may not be fully optimized from each sender's perspective. Results also indicate that the IIA has better performance than minimum-cost leaf-join for all workload conditions.

V. Related Works

Narada [4]. Narada is an end system multicast scheme that uses an elegant and practical self-organized mesh-first approach. Narada can be considered as a generic solution for providing application layer multicasting. Since we assume guaranteed services at the network layer, Narada does not perform very well in such situation, as indicated in the simulation results in Section IV.

Switch Tree Protocol[7]. Switch Tree Protocol (STP) defines a family of tree-first protocols. The switch-any approach bears some similarities with our on-line mode IIA Protocol. The major difference is that STP is a self-organizing approach which takes a longer time to produce a stable tree structure and needs the support of loop avoidance mechanisms.

Overcast[6]. The Overcast work is optimized for bandwidth and the proposed Up/Down self-organizing protocol is scalable. The limitation of their approach is that it does not apply to the multicast applications with time constraints such as video/audio conferencing. It should be noted that when network resource (bandwidth) are sufficient, overcast would yield a concatenated unicast tree.

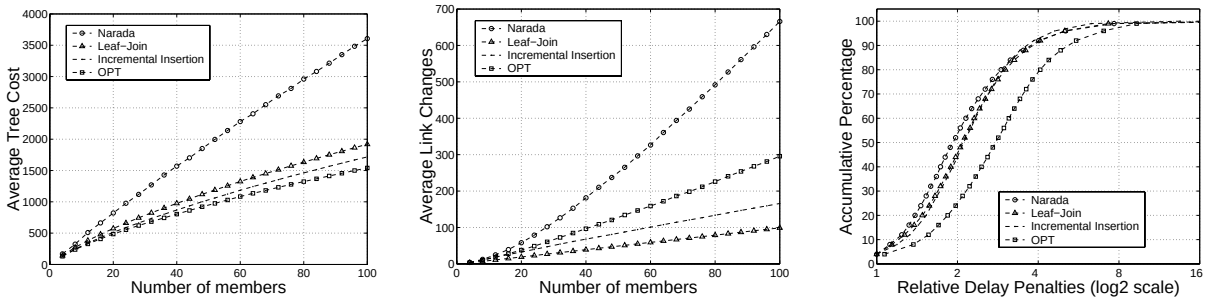
VI. Conclusions

In this paper, we have proposed an heuristic algorithm named Incremental Insertion to construct source-specific multicast overlay tree for DiffServ domains. The algorithm determines an appropriate location for inserting a new joining node while considering the QoS constraints. With the network QoS support, it is easier to build and maintain QoS-aware overlay trees. The simulation results depict that the proposed scheme has a balanced performance for various performance metrics analyzed on this paper. In the future, we will

TABLE I

STATISTIC OF 100 RUNS (100 MEMBER, WAXMAN2 MODE)

	Narada	Leaf-Join	IIA	OPT
Tree Cost(ave)	3605.76	1919.52	1716.34	1540.72
Tree Cost(max)	4251	2114	1898	1382
Tree Cost(min)	2915	1681	1534	1723
Tree Cost(std)	24.01	8.82	8.27	7.44
Link Changes (ave)	665.80	99	165.96	295.77
Link Changes(max)	1155	99	192	371
Link Changes(min)	331	99	138	236
Link Changes(std)	19.69	0	0.99	2.7
RDP(ave)	2.55	2.56	2.43	3.26
RDP(50th percentile)	1.92	2.09	2.11	2.77
RDP(95th percentile)	4.66	4.71	4.34	6.58
RDP(worst case)	33.0	23.20	13.55	18.75



(a) Average Tree Cost (b) Average Link Changes (c) Cumulative RDPs

Fig. 5. Results of Waxman2 model.

expand our intra-domain solutions to inter-domain environments.

REFERENCES

[1] J. Wroclawski *The Use of RSVP with IETF Integrated Services* RFC2210, September 1997.

[2] S. Blake et al. *An Architecture for Differentiated Services* RFC2475, December 1998.

[3] K. Calvert, M. Doar, and E. W. Zegura, *Modeling Internet Topology*, IEEE Communications Magazine, pp. 160-163, June 1997.

[4] Y. Chu, S. G. Rao, and H. Zhang, *A Case For End System Multicast*, Proceedings of ACM SIGMETRICS, pp. 1-12, June 2000,

[5] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, *ALMI: An Application Level Multicast Infrastructure*, 3rd USENIX Symposium on Internet Technologies, pp. 49-60, March, 2001.

[6] J. Jannotti, D. K. Gifford, and K. L. Johnson, et al., *Overcast: Reliable Multicasting with an Overlay Network*, Proceedings of 4th USENIX OSDI, pp. 197-212, October 2000.

[7] D. A. Helder and S. Jamin, *End-host Multicast Communication Using Switch-tree Protocols*, Proceedings of GP2PC on Large Scale Distributed Systems, May 2002.

[8] S. Jain, R. Mahajan, D. Wetherall and G. Borriello *Scalable Self-Organizing Overlays*, Submitted for review.

[9] S. Banerjee, B. Bhattacharjee and S. Parthasarathy, A Protocol for Scalable Application Layer Multicast, CS-TR 4278, Department of Computer Science, University of Maryland, College Park, July 2001

[10] QBone Signaling Design Team, *Final Report* http://qbone.internet2.edu/bb/