

TECHNIQUES OF RECONSTRUCTING OVERLAY MULTICAST TREES FOR MULTIMEDIA STREAMING

Mengkun Yang Zongming Fei

Department of Computer Science, University of Kentucky
Lexington, KY 40506-0046, U.S.A.

ABSTRACT

Overlay multicast is an efficient and deployable approach for delivering multimedia content to a large number of receivers. A key problem in overlay multicast is how to deal with node failures and ungraceful leavings. When a non-leaf end host fails or leaves the multicast session, all downstream nodes will be affected. In this paper, we adopt the proactive approach, which pre-calculates a candidate node (called *parent-to-be*) for each node to connect to in case its current parent dies. The goal is to recover the multicast tree quickly so that the disruption of service to those affected nodes is minimized. We combine the local parent-to-be locating and global parent-to-be locating schemes together, in order to take advantage of less interference among affected nodes in the local scheme and the flexibility of the global scheme. The quality of the recovered tree is improved while the responsiveness of the proactive approach is maintained.

1. INTRODUCTION

Overlay multicast is an efficient and deployable approach for delivering multimedia content to a large number of receivers [1]. It implements the multicast functionality at end hosts rather than routers. End hosts join the multicast tree not only as leaves as in IP multicast, but as non-leaf branching nodes as well. Because end hosts are potentially more susceptible to failures than routers and may leave the multicast group voluntarily, one of the key issues in maintaining the overlay multicast topology is how to reconstruct the overlay tree after nodes fail or leave. The time to resume the data flow to the affected downstream nodes is an important measure of the quality of the recovery mechanism.

In this paper, we specifically investigate the problem of restoring the overlay multicast tree for multimedia streaming applications. Usually, significant bandwidth is needed

for streaming multimedia data over the network. An end host only has limited bandwidth to participate in media streaming. The number of streams it can send and receive at the playout rate of the media is usually limited, and it can be abstracted as the *degree constraint* on the node. A node can accept more nodes as its children in the overlay tree only if it has not reached its degree constraint. These nodes are called *unsaturated nodes* in the later discussion of the paper. Due to the degree constraint on end hosts, a disconnected node cannot always successfully reconnect to the multicast tree by connecting to an arbitrary tree node (e.g., the grandparent) after its parent leaves.

There are two approaches to the recovery of overlay multicast trees. One is the *reactive* approach [2,3], in which the tree restoration process starts *after* node departures. In SpreadIt [2], the affected nodes find appropriate places in the subtree of the grandparent or the root after the parent fails. They may be redirected to other nodes if the contacted node cannot accommodate them. Therefore, they may have to contact several nodes in the tree before they find an appropriate location to connect to. It usually takes quite some time to repair the overlay multicast tree before data can flow to the affected nodes. *CoopNet* [3] depends on the complete tree topology maintained at the root of the tree for recovery. After each node fails, the root searches this locally maintained tree topology and finds new parents for the children of the failed node.

The other is the *proactive* approach proposed in our previous work [4], which plans for departures before they really happen. The basic idea is that each non-leaf node in the overlay multicast tree pre-computes a rescue plan before it fails or leaves. The *rescue plan for a node* is to pre-calculate a *parent-to-be* for each of its children. Once it leaves the tree, all its children contact their respective “parents-to-be” immediately.

In this paper, we extend our previous work by taking into account not only the recovery efficiency, but the quality of the restored tree as well. The specific performance metric we consider is the average latency from the root of the tree to those affected nodes, because it is an important performance measure for multimedia streaming. Another

THIS WORK WAS SUPPORTED IN PART BY THE NATIONAL SCIENCE FOUNDATION UNDER GRANTS CCR-0204304 AND EIA-0101242, AND A GRANT FROM THE KENTUCKY SCIENCE AND ENGINEERING FOUNDATION AS PER GRANT AGREEMENT #KSEF-148-502-05-139 WITH THE KENTUCKY SCIENCE AND TECHNOLOGY CORPORATION.

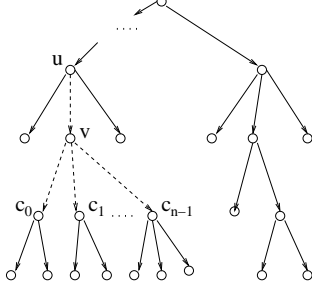


Fig. 1. Local Parent-to-be Assignment

extension is that we not only consider the local parent-to-be assignment, which assigns either the grandparent or one of the siblings of a node as its parent-to-be, but propose a global parent-to-be locating technique that allows a node to have any non-descendent unsaturated node as its parent-to-be, in order to improve the quality of the recovered tree and deal with the cases that cannot be solved by the local assignment. This gives us more flexibility for finding a parent-to-be for a node to minimize its latency after recovery. These two techniques are integrated to take the advantages of each approach. Simulations show that the integrated scheme effectively improves the quality of the recovered tree, while maintaining the recovery responsiveness of the proactive approach.

2. LOCAL PARENT-TO-BE ASSIGNMENT

2.1. The Idea of Local Proactive Recovery

The idea of proactive recovery is to make arrangement for affected nodes so that they know who should be their new parents in case of failure. Local parent-to-be assignment focuses on finding parent-to-be among the grandparent and siblings only. For example, in Figure 1, we want to calculate the rescue plan to deal with the failure of non-leaf node v . It should tell v 's children c_0, \dots, c_{n-1} who will be their new parents. The local assignment will find parent-to-be from grandparent u and siblings c_0, \dots, c_{n-1} .

The problem can be formulated as calculating a spanning tree among u and c_0, \dots, c_{n-1} with u as the root. The parent of c_i in this spanning tree will be its parent-to-be during recovery. Note the *difference between the multicast tree in use and the spanning tree to be calculated*. We do not actually add the edges in the calculated tree to the multicast tree in use. The only purpose is to find the parent-to-be for each child.

The remaining problems we need to figure out are the degree-constraint on each node and the performance measure to optimize when we calculate the spanning tree. We use *residual degree* of node x , denoted as $d_r(x)$, to represent how many more edges that a node can be connected to

in the *multicast tree*, and *available degree* of node x , denoted as $d_a(x)$, to represent how many edges that a node can be connected to in the *spanning tree to be calculated*. Note the available degree of node x is the degree constraint on node x in the tree to be calculated. In order not to increase the burden on the grandparent, we let $d_a(u) = 1$. For each child, it will get one degree released when v fails. Therefore, we have the degree constraint on child c_i to be $d_a(c_i) = d_r(c_i) + 1$.

The performance measure we want to optimize is the end-to-end latency from the root to the end hosts in the multicast tree. We formulate the spanning tree problem as a minimum average latency degree-constrained (MALDC) tree or a minimum weighted latency degree-constrained (MWLDC) tree problem.

GIVEN: 1) an undirected complete graph $G = (V, E, r)$ with a weight function $w : E \rightarrow \mathcal{R}$, where $r \in V$, \mathcal{R} is the set of real numbers and for an edge $e \in E$, $w(e)$ represents the latency (or cost) of edge e . 2) node degree constraints $d_a(i)$ for node $i \in V$.

FIND: 1) a spanning tree $T_{dc} = (V, E_{dc}, r)$ rooted at r such that $E_{dc} \subseteq E$, and T_{dc} satisfies the degree constraints that for each node $i \in V$, its degree is less than or equal to $d_a(i)$. This spanning tree is a *feasible solution* to the problem; or

2) a **MALDC** tree $T_{maldc} = (V, E_{maldc}, r)$, such that the average path latency $\frac{\sum_{i \in V} l(i)}{|V|}$ is minimum among all feasible solutions, where

$$l(i) = \begin{cases} l(p(i)) + w(p(i), i) & \text{if } i \neq r \\ 0 & \text{if } i = r \end{cases} \quad (1)$$

with $p(i)$ denoting the parent of node i and $l(i) = 0$; or

3) a **MWLDC** tree $T_{mwlcdc} = (V, E_{mwlcdc}, r)$ such that the weighted latency $\sum_{i \in V} \frac{l(i) \times s(i)}{\sum_{i \in V} s(i)}$ is minimum among all feasible solutions, where $s(i)$ is the size of the subtree rooted at node i in the multicast tree in use.

For example, in Figure 1, to find the parents-to-be for v 's children c_0, \dots, c_{n-1} , we calculate a spanning tree with $V = \{u, c_0, \dots, c_{n-1}\}$ and $r = u$. Since the latency of every c_i affects the latencies of all its downstream nodes, we hope c_i with a large subtree can get small latency after recovering from the parent failure. The MWLDC formulation achieves this goal by weighting the node latencies and considering the *weighted latency* of node i defined as $\frac{l(i) \times s(i)}{\sum_{i \in V} s(i)}$. The MALDC is a special case of the MWLDC problem with $s(i) = 1$. In the remaining parts of this paper, we mainly discuss the MWLDC tree. Both the MALDC and the MWLDC problems are NP-hard [5]. Next, we give the heuristic algorithm.

2.2. Heuristic algorithm

Given v 's parent u and its children $C = \{c_0, \dots, c_{n-1}\}$, the algorithm finds a spanning tree T among u and C with u as the root. Similar to the Dijkstra's algorithm, we have two sets. Set A includes the nodes already connected to the tree, while set B consists of the nodes not yet in the tree. We initialize $A = \{u\}$ and $B = C$. Every node $y \in B$ has a potential parent $p(y) \in A$ such that among all those nodes currently in A that have available degrees to accept children, connecting $p(y)$ to y will give y the smallest latency $l(y)$. At each step, we select a node $b \in B$ such that its weighted latency $l(b) * s(b)$ is the smallest among all nodes currently in B , where $s(b)$ is the size of the subtree rooted at b . We remove b from set B and insert it into set A , and decrease the available degree of b and $p(b)$ by 1. We also update all $p(y)$ and $l(y)$ for all the nodes y remaining in B . This process repeats until B becomes empty or all nodes in A have zero available degree.

2.3. Discussion

The local parent-to-be assignment only uses local nodes as the candidates for parents-to-be and the number of links on the grandparent before and after the actual recovery is the same. Hence, it can accommodate concurrent failures, such as two siblings fail at the same time. However, the local assignment cannot guarantee finding a parent-to-be for every affected node. This will happen when the total residual degree of c_i 's is not large enough. In general, if the total residual degree of c_i 's is k ($k < n - 1$), we can only find parents-to-be for $k + 1$ children. This leads us to consider global locating.

3. GLOBAL PARENT-TO-BE LOCATING

Global locating does not limit the parent-to-be to the grandparent and siblings. Rather, a node can have any non-descendent unsaturated node as its parent-to-be. This method can find a parent-to-be when the local assignment is not successful. Also it can be used to find another parent-to-be to get smaller latency even when we can find a local parent-to-be.

3.1. Maintenance of Unsaturated Node Information

A key problem of the global parent-to-be locating is to find unsaturated nodes. We avoid making every node to independently search the multicast tree, since it is neither bandwidth-efficient nor responsive. Rather, we let a *Rendezvous Point (RP)* maintain an unsaturated node list. Each unsaturated node x periodically sends to the RP its state, including its residual degree, *ancestor list* $\mathcal{A}(x)$, and the number of its *children-to-be*, which are those nodes that take x as their parent-to-be.

The RP may, or may not, have the information about latencies among nodes in the multicast tree. If it has, it will provide a more informed selection of parent-to-be candidates. However, we will not depend on the maintenance of accurate latency information in the following discussion. Also the residual degree information about the unsaturated nodes may even be out of date. It is individual node's responsibility to check out the latencies to the nodes returned from the RP and whether they really have residual degrees to accept children.

3.2. Global Parent-to-be Locating Scheme

When a node c wants to find a parent-to-be globally, it sends a request to the RP, which checks the unsaturated node list and returns to c a parent-to-be candidate set P . Then c probes set P , and selects a node $f \in P$ as its parent-to-be such that f leads to the smallest value $l(f) + w(f, c)$ among all nodes in P that have at least one residual degree, where $l(f)$ is f 's latency to the root of the multicast tree and $w(f, c)$ is the probed latency between f and c .

The ancestor list $\mathcal{A}(x)$ helps the RP determine whether a node is eligible to be a parent-to-be candidate. Specifically, node p is *globally eligible* to be the parent-to-be of c if and only if 1) c is not in the ancestor list $\mathcal{A}(p)$ of p , and 2) p has at least one residual degree. Also the RP tries to balance the load on the unsaturated nodes. For every node eligible to be the parent-to-be of the inquiring node c , RP computes the ratio of its number of children-to-be to its number of residual degrees, and then returns β nodes with the smallest such ratios as the parent-to-be candidate set to c , where β is a system design parameter specifying the size of the parent-to-be candidate set. Main benefits to do such load balancing include the following. First, it can reduce the probability of parent-to-be conflicts in the face of concurrent node failures. Second, the departure of any unsaturated node does not lead too many nodes who are its children-to-be to update their parents-to-be.

3.3. Integrated Parent-to-be Assignment

The integrated parent-to-be assignment scheme is based on the local scheme, and gets help from the global scheme when necessary. Each non-leaf node v first tries to find the parents-to-be for its children c_0, \dots, c_{n-1} only among its parent u and its children. Then v lets some of its children find their parents-to-be through the global locating, if 1) v is not able to find the parents-to-be locally for these children due to lack of enough total residual degree among c_0, \dots, c_{n-1} ; or 2) the locally assigned parents-to-be of these children will cause a latency degradation greater than α , i.e., their latencies in the recovered tree will be more than $(1 + \alpha)$ times their latencies in the current multicast tree.

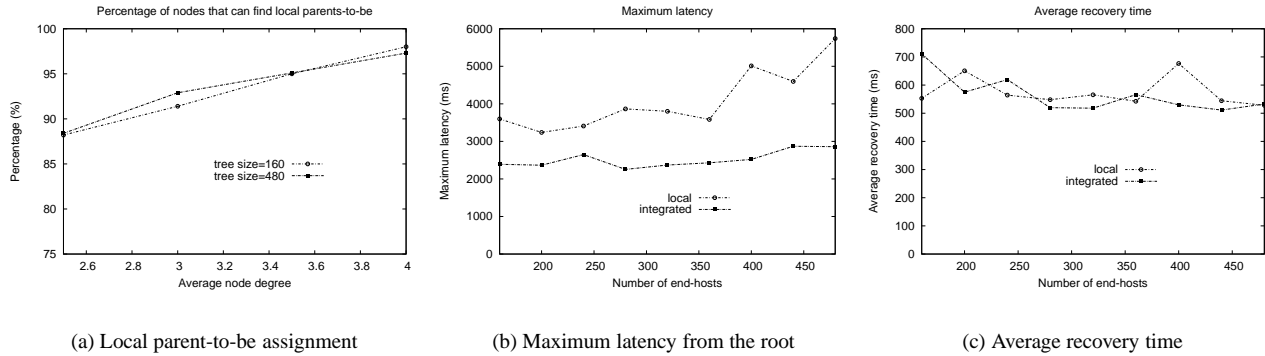


Fig. 2. Proactive recovery: local parent-to-be assignment VS integrated parent-to-be assignment

4. PERFORMANCE EVALUATIONS

We use GT-ITM [6] to generate 1600 node transit-stub topologies as the underlying network. The multicast source and other end-hosts are randomly distributed in stub domains. The network-layer link latencies range from 1ms to 245ms. The application-level latency between two end-hosts is the sum of link latencies on the shortest path between them. The total degree of each node is uniformly distributed between 2 and 6, if not mentioned otherwise. It is the maximal number of application layer connections it can establish with other end hosts. All experiments begin with a multicast tree established by a algorithm similar to Section 2.2. Then end-hosts join and leave the tree dynamically modeled as a Poisson process with rate $\lambda = 0.5/second$. Each experiment lasts for two hours.

Local parent-to-be assignment is not always feasible: As discussed in Section 2.3, some nodes cannot find parents-to-be locally. Hence they have to resort to the global parent-to-be locating technique. Figure 2(a) depicts the percentage of such tree nodes. The x axis is the average number of node degrees D_{avg} varying from 2.5 to 4. The node degree is uniformly distributed in range $[2, 2 \times D_{avg} - 2]$. Different curves represent the results for multicast trees with different sizes. When $D_{avg} = 2.5$, about 13% of nodes cannot find parents-to-be locally.

Integrated parent-to-be assignment improves the quality of the restored trees: The quality of the restored tree is measured by the latency from the root to the tree nodes after recovering from node failures. Figure 2(b) plots the maximum such latency. For the integrated parent-to-be assignment, the maximum latency ranges between 2254ms and 2870ms. In contrast, the local assignment produces much higher maximum latency ranging between 3239ms and 5739ms, and its curve climbs up fast as the size of the multicast tree increases.

Integrated parent-to-be assignment has good recovery responsiveness: We compare the average recovery time of

the local and the integrated parent-to-be assignment schemes in Figure 2(c). The average recovery time of the integrated scheme is comparable to and sometimes even better than that of the local assignment. So the benefits of the integrated scheme is achieved without sacrificing the recovery responsiveness. We also compare the proactive approach with the reactive approach. As expected, the reactive approach takes about double of the time by the proactive approach. Due to the lack of space, we will not present the figure here.

5. CONCLUDING REMARKS

This paper proposed a global scheme for overlay tree reconstruction. It was integrated with the local recovery scheme to obtain the better performance for multimedia applications. Simulations demonstrate that the quality of the recovery tree is improved while the responsiveness of the proactive approach is maintained.

6. REFERENCES

- [1] Yanghua Chu, Sanjay Rao, and Hui Zhang, "A case for end system multicast," in *Proceedings of ACM Sigmetrics*, June 2000, Santa Clara, CA.
- [2] Hrishikesh Deshpande, Mayank Bawa, and Hector Garcia-Molina, "Streaming live media over a peer-to-peer network," 2001, Technical Report CS-2001-31, Stanford University.
- [3] V. Padmannabhan, H. Wang, and P. Chou, "Resilient peer-to-peer streaming," in *Proceedings of the 11th ICNP'03*, November 2003.
- [4] Mengkun Yang and Zongming Fei, "A proactive approach to reconstructing overlay multicast trees," in *Proceedings of INFOCOM'04*, March 2004.
- [5] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proceedings of INFOCOM'03*, March 2003.
- [6] Ellen W. Zegura, Ken Calvert, and Samrat Bhattacharjee, "How to model an internetwork," in *Proceedings of INFOCOM'96*, 1996.