A Proactive Approach to Reconstructing Overlay Multicast Trees

Mengkun Yang   Zongming Fei
Department of Computer Science
Universtiy of Kentucky
Lexington, KY 40506-0046
Email: {myang0,fei}@cs.uky.edu

***Abstract*— Overlay multicast constructs a multicast delivery tree among end hosts. Unlike traditional IP multicast, the non-leaf nodes in the tree are normal end hosts, which are potentially more susceptible to failures than routers and may leave the multicast group voluntarily. In these cases, all downstream nodes will be affected. Thus an important problem in overlay multicast is how to recover from node departures in order to minimize the disruption of service to those affected nodes. In this paper, we propose a proactive approach to restore overlay multicast trees. Rather than letting downstream nodes try to find a new parent *after* a node departure, each non-leaf node pre-calculates a *parent-to-be* for each of its children. When this non-leaf node is gone, all its children can find their respective new parents immediately. The salient feature of the approach is that each non-leaf node can compute a rescue plan for its children independently, and in most cases, rescue plans from multiple non-leaf nodes can work together for their children when they fail or leave at the same time. We develop a protocol for nodes to communicate with new parents so that the delivery tree can be quickly restored. Extensive simulations demonstrate that our proactive approach can recover from node departures 5 times faster than reactive methods in some cases, and 2 times faster on average.**

I. Introduction

Overlay multicast (also known as application-layer multicast) [1]–[3] implements the multicast functionality at end hosts rather than routers. It uses a virtual overlay network topology based on the underlying unicast mechanism to transport data between end hosts. This may require more overall bandwidth than IP multicast because duplicate packets travels the same physical links multiple times, but it provides an inexpensive, deployable method of providing point-to-multipoint communication. One example application using overlay multicast is live media streaming, which constructs a delivery tree at the application layer from the source to all participating end hosts. Usually it lasts for a long period of time and needs significant bandwidth constantly.

One of the key issues in maintaining the overlay multicast topology is how to reconstruct the overlay tree after a node fails or simply leaves the multicast session voluntarily. The time to resume the data flow after a node departure (failure or leave) is important for multicast applications such as live media streaming. In the traditional network-layer multicast, the non-leaf nodes in the delivery tree are routers, which are relatively stable and do not leave the multicast tree voluntarily. End hosts join the multicast tree only as leaves, and their departure does not require restoring the delivery tree. However, this is not the case for overlay multicast. When a non-leaf end host leaves the multicast session, all the nodes in the subtree rooted at it are affected. Because end hosts are potentially more susceptible to failures than routers and may leave the multicast group voluntarily, it is important that we can find an efficient mechanism to restore the overlay multicast tree in these cases.

There are two approaches to the recovery of the overlay multicast tree. One is the *reactive* approach [4], [5], in which the tree restoration process starts *after* node departures. It usually takes quite some time to repair the overlay multicast tree before data can flow to the affected nodes. The most difficult task is to find a new parent for each disconnected node. Usually they must contact several nodes in the tree before they find an appropriate location.

The other is the *proactive approach*, which plans for departures *before* they happen. The Probabilistic Resilient Multicast (PRM) [6] proposed a *Randomized forwarding* method, in which each overlay node chooses a constant number of other overlay nodes uniformly at random and forwards data to each of them with a low probability. It can achieve high delivery ratios in case there is a node failure. The extra data are sent on the data forwarding plane, and the volume of traffic generated can be significant for some overlay multicast applications, such as live media streaming.

We propose a proactive solution that works on the control plane in this paper. 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 node x* is to pre-calculate a *parent-to-be* for each of x's children. Once the departure of x really happens, all of its children can contact their respective "parents-to-be" immediately. We face several challenges.

First, we need to consider the degree constraints in overlay multicast. They can be easily observed in streaming applications. For example, assume the media playback rate is B and the bandwidth of the connection of an end host is b_i. The total number of streams it can have is $\lfloor b_i/B \rfloor$. It is called the *total degree* of this node, representing the total number of connections that this node can establish with the outside

This work was supported in part by the National Science Foundation under Grants CCR-0204304 and EIA-0101242.

world. Even for non-streaming applications, the total number of connections that a node can handle is also limited.

With the degree constraints, finding the rescue plan for node x becomes more complicated. We cannot use the parent of x (i.e., the grandparent of x's children) as the *parent-to-be* for all of x's children as used in the network-layer fault-tolerant multicast routing protocol [7], because this may violate the degree constraint of the grandparent.

Second, we need to be able to deal with multiple leaves, and the interference between their rescue plans should be minimized. Siblings may have their independent rescue plans, such as letting their children contact the grandparent. When they both fail, the collective effect may result in the traffic concentration at the grandparent and the violation of its degree constraint.

The idea of our scheme is to refrain from increasing the number of children of x's parent before and after x leaves. The plan lets only *one* of x's children to contact the grandparent once x fails. This is similar to the practice of selecting one of x's children as the crowned prince to inherit x's position. This intends to deal with both the first challenge that x's children compete with each other for the available degree at the grandparent, and this second challenge that siblings' children compete with each other. We formulate the problem as a degree-constrained spanning tree problem [8] among x's parent, x's children, and sometimes descendants of x's children. The solution to the problem not only selects the special "crowned" child, but arranges "parents-to-be" for other children as well.

Third, we need to be able to deal with the cases in which the rescue plan is not available at a child. For example, the tree topology just changed and x does not have enough time to finish computing the rescue plan before it fails. Therefore, the *parents-to-be* of x's children are empty. We develop a recovery protocol that can make use of the *parent-to-be* information whenever possible so that the affected nodes can find new parents quickly. At the same time, it still works and finds a new parent for the affected node, even if the parent-to-be information is not available.

Several principles guided our design of the proactive scheme to solve the problem of reconstructing overlay multicast trees.

First, it should be *responsive*. We should minimize the time from a node failure to the time we resume the normal delivery.

Second, it should be a *distributed* method. We cannot depend on a global solution to reconstruct the whole multicast tree from scratch after each leave. Instead, we want to keep the number of nodes involved in the reconstruction process as few as possible so that the disruption of service is minimized.

Third, it should be *scalable*. The method should be able to deal with an overlay multicast session with a large number of nodes. More importantly when multiple nodes leave or fail, it should not create an implosion problem for any node in the multicast tree.

The *contributions* of this paper can be summarized as follows:

1) We propose a proactive approach to reconstruct overlay multicast trees by pre-calculating parents-to-be for possible affected nodes.

2) We formulate the problem as a degree-constrained spanning tree problem and find a sufficient and necessary condition that decides whether there is a feasible solution to the problem.

3) We design an algorithm to solve the specific spanning tree problem.

4) We develop a protocol for nodes to communicate with new parents so that a new delivery tree can be quickly restored.

5) Our simulations demonstrate that our proactive approach can recover from node departures 5 times faster than reactive methods in some cases, and 2 times faster on average, while the quality of the tree restored and the overhead of the recovery is comparable to these methods.

The rest of the paper is structured as follows. Section II describes the problem of restoring overlay multicast trees. Section III proposes a proactive reconstruction scheme and presents the algorithm and the protocol in detail. Performance evaluations are presented in Section IV and related work is discussed in Section V. We conclude the paper in Section VI.

II. The Problem of Restoring Overlay Multicast Tree

We start with the methods of constructing the overlay multicast trees and then describe the problem of restoring the overlay multicast trees when nodes fail or leave.

A. Construction of Overlay Multicast Trees

Construction of an overlay multicast tree can be modeled as a degree-constrained spanning tree problem.

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

FIND: 1) a spanning tree $T = (V, E')$ such that $E' \subseteq E$, and T satisfies the degree constraints that for each node $i \in V$, its degree is less than or equal to k_i. This spanning tree is a *feasible solution* to the problem; or

2) a minimum spanning tree $T = (V, E')$, such that the total weight is minimum among all feasible solutions. This spanning tree is an *optimal solution* to the problem.

Most of existing work on the degree-constrained minimum spanning tree [8], [9] or degree-constrained minimum steiner tree [10] assumes $k_i \geq 2$. We do not have this assumption here because we may encounter cases in which the degree constraints are less than 2. One example is the pre-computation of rescue plans in the following subsections. Another difference is that we assume G is a complete graph, because in overlay multicast, theoretically we can have an edge between any two end hosts.

When $k_i < 2$, it is possible that we cannot find a feasible solution to the problem, even for a complete graph. Here we give a sufficient and necessary condition for the degree-constrained spanning tree problem to have a feasible solution. The proof can be found in Appendix A.

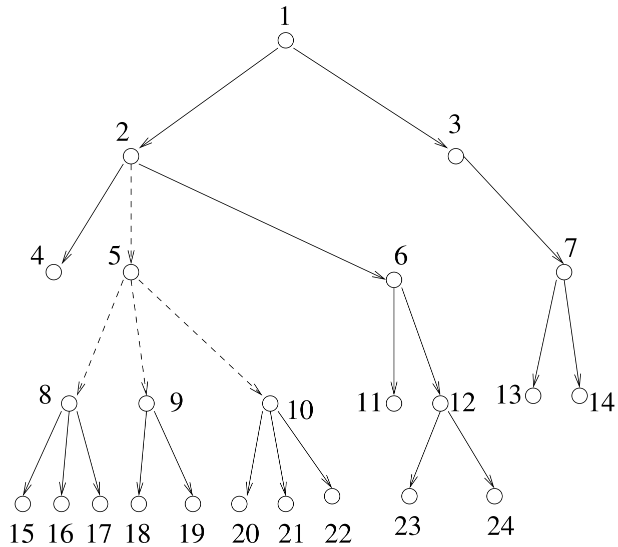

Fig. 1. An example of node failure

Theorem 1: For the degree-constrained spanning tree problem of a complete graph $G = (V, E)$ with m nodes and degree constraints k_i for node i ($0 \le i \le m-1$), it has a feasible solution *if and only if* $k_i \ge 1$ and $\sum_{i=0}^{m-1} k_i \ge 2m-2$.

The degree-constrained minimum spanning tree problem is an NP-complete problem [8], [11], and many heuristics have been proposed to solve the problem. One simple yet efficient heuristic is based on Prim's algorithm for the minimum spanning tree problem [9], [12]. Starting with an arbitrary node, at each step it includes the cheapest eligible edge – one connecting a node currently in the partial spanning tree with one not in it – that does not violate the degree constraints. The step repeats until all nodes are connected.

This heuristic can be used to construct the initial overlay multicast tree when there are a number of interested end hosts ready for joining a multicast session. After the initial overlay multicast tree is set up, more nodes may want to join the multicast session. Several heuristics have been proposed to deal with the problem. Optimization heuristics can be used to adjust the organization of the nodes between joins. We will not go further to discuss these heuristics because this problem is not the focus of this paper. Interested readers are referred to [4], [5], [13].

B. The Problem of Restoration

The focus of this paper is on the problem of restoration of the overlay multicast tree after nodes leave or fail, especially those non-leaf nodes in the tree. When a node leaves a multicast tree, it may send a message to inform affected nodes. When a node fails, we expect that other nodes can detect the failure by some heartbeat mechanism. We have the same problem of restoring the overlay multicast tree after failures or voluntary leaves. Therefore we use "fail" and "leave" interchangeably in the rest of the paper to mean either or both cases.

In Fig. 1, we show an example of node failure. When node 5 fails, all the links (shown as dotted lines) between 5 and other nodes will be affected. The affected nodes are 8, 9, 10, and 15-22. We need to find new parents for them, or at least 8, 9, 10.

Let us first look at some existing *reactive* methods [5]. They invoke the repair process *after* node 5 fails.

- *Grandfather*: Only the children of the failed node contact the grandfather. The grandfather will try to accommodate them; if its degree is exhausted, it will redirect them to its descendants. For example, after node 5 fails, nodes 8, 9, 10 will contact node 2, which accepts some of them as its children, and directs others to its descendants such as nodes 4 and 6, and possibly newly accommodated nodes. The parents of nodes 15 to 22 remain unchanged.
- *Root*: Only the children of the failed node contact the root, which uses the same strategy as the grandfather above.
- *Grandfather-All*: All descendants of the failed node try to recover by contacting the grandfather. For example, nodes 8, 9, 10, and 15-22 will all contact node 2.
- *Root-All*: All descendants of the failed node try to recover by contacting the root of the tree.

Obviously we can find that these strategies may create a concentration of traffic at the grandfather or the root. The time for the tree recovery can be long.

III. The Proactive Reconstruction of Overlay Multicast Trees

A. A Proactive Approach

The idea of the proactive approach is to find a rescue plan *before* the failure happens. For each non-leaf node, it must find *parents-to-be* for all its children. Once it dies, each of its children will know exactly who should be its new parent. One single message from each child will complete the recovery process. This is much faster than the reactive approach, where all children will go through a process of finding an appropriate node to be their parents.

Two decisions have been made during the formation of the problem. First, we want to make use of existing parent-child relationships as much as possible, to reduce the impact of the node failure. Therefore, we only find the *parents-to-be* for the children, instead of all descendants. For example, in Fig. 1, node 5 only plans the *parents-to-be* for nodes 8, 9, and 10. All other descendants (nodes 15 to 22) will keep the current parent. Second, we want to keep the impact on the grandparent low. We want to have only one child contact the grandparent so that the degree of the grandparent will be the same before and after the node failure.

If a non-leaf node (say x) leaves the tree, the subtree rooted at it will become a forest of smaller subtrees, with each of x's children being a root of one subtree. A simple way to find a *parent-to-be* for each child is to form a spanning tree among the roots of these subtrees. For example, the task becomes finding a spanning tree among nodes 8, 9, 10 in Fig. 1. There are two questions.

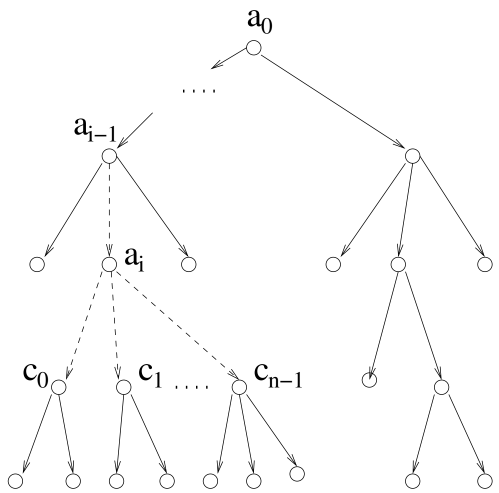

Fig. 2. The problem

- First, this spanning tree does not always exist. Because of the degree constraints, we are not necessarily able to find a feasible solution, as stated in the theorem in Section II. We will present a method that organizes the forest into a tree in this case.
- Second, it is possible the distance between the root of the formed tree and the grandparent is very large. We need to choose an appropriate root for such a tree. Therefore, we include the grandparent in the formation of the spanning tree problem.

B. Spanning Tree Problem

In general, we can formulate the problem as follows. Suppose node a_i in the multicast tree has n children $\{c_0, c_1, \cdots, c_{n-1}\}$ (see Fig. 2). Each of them may in turn have downstream nodes. The path from the root to a_i is $a_0, a_1, \cdots, a_{i-1}$. We need to calculate a tree among a_{i-1} and *subtrees* rooted at $c_0, c_1, \cdots, c_{n-1}$, with the degree constraints on each node. The goal is to minimize the total cost of the tree.

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 of the result is to find the *parent-to-be* for each child. The difference from the problem in Section II is that we cannot use the total degree of each node as its degree constraint in the tree to be calculated, because these nodes already connect with other nodes. So we define *used degree* of a node as the number of (in and out) edges connected to the node in the current multicast tree, and *residual degree* as the difference between the total degree and the used degree. The residual degree reflects how many more edges that a node can be connected to. The total degree, the used degree and the residual degree of node x are represented by $d_t(x), d_u(x)$ and $d_r(x)$, respectively. Obviously $d_t(x) = d_u(x)+d_r(x)$. Also we expect that $d_u(x) \geq 0$ and $d_r(x) \geq 0$. In the example in Fig. 1, the used degrees of nodes 8, 9, 10 are $d_u(8) = 4, d_u(9) = 3, d_u(10) = 4$. If we assume that the total degree of each node is 4, the residual degrees will be $d_r(8) = 0, d_r(9) = 1, d_r(10) = 0$.

Ideally we would like to construct a spanning tree among $a_{i-1}, c_0, \cdots, c_{n-1}$. We first consider the degree constraints on these nodes in the spanning tree to be calculated. Note each node gets one degree released because its connection with a_i would be removed, should a_i leaves. For node a_{i-1}, we do not want to increase the number of streams it connects to. So we want its degree constraint to be $k_{a_{i-1}} = 1$. For node c_j $(0 \leq j \leq n-1)$, the degree constraint is $k_{c_j} = d_r(c_j) + 1$. Whether we can form a spanning tree among a_{i-1} and c_j's can be determined by the residual degrees $d_r(c_j)$'s of c_j's in the current tree, as described in the following theorem. The proof is given in Appendix A.

Theorem 2: Assume the residual degree of node c_j $(0 \leq j \leq n-1)$ is $d_r(c_j)$ in the current tree, we can form a feasible spanning tree among a_{i-1} and c_j's $(0 \leq j \leq n-1)$ with the degree constraint on a_{i-1} equal to $k_{a_{i-1}} = 1$, and the degree constraint on c_j equal to $k_{c_j} = d_r(c_j) + 1$, *if and only if* $\sum_{j=0}^{n-1} d_r(c_j) \geq n-1$.

We calculate the sum of the residual degree of children of a_i as $D_R = \sum_{j=0}^{n-1} d(c_j)$. If $D_R \geq n-1$, we can form a degree-constrained spanning tree among nodes a_{i-1} and $c_0, c_1, \cdots, c_{n-1}$. If $D_R < n-1$, we cannot. For example, in Fig. 1, we assume that each node has a total degree of 4. Then the residual degrees of them are $d_r(8) = 0, d_r(9) = 1, d_r(10) = 0$. The total is $1 < n-1$, because $n = 3$ here. As a matter of fact, if node 5 is gone, we cannot form a spanning tree among nodes 2, 8, 9, 10, without violating their degree constraints.

Our strategy in this case is to find descendants of nodes 8, 9, 10. Those with a non-zero residual degree can contribute to the construction of the spanning tree. For example, we can find node 16, which has a residual degree $d_r(16) = 3$. We then form a spanning tree among node 2 and the three subtrees, by adding edges between nodes $2, 8, 9, 10$, and 16. We give an example spanning tree in Fig 3. Node 5 will derive that the parent-to-be of node 8 is node 9, the parent-to-be of node 9 is node 2, and the parent-to-be of node 10 is node 16.

In general, in case of $\sum_{j=0}^{n-1} d_r(c_j) < n-1$, we will find downstream nodes of c_j's $(0 \leq j \leq n-1)$, that have a residual degree of at least 1. Specifically, we will find l_j $(l_j \geq 0)$ nodes, $c_{j0}, c_{j1}, \cdots, c_{j,l_j-1}$, in the subtree rooted at root c_j $(0 \leq j \leq n-1)$, so that $\sum_{j=0}^{n-1}(d_r(c_j) + \sum_{l=0}^{l_j-1} d_r(c_{jl})) \geq n-1$. We can establish a degree-constrained spanning tree among a_{i-1} and these subtrees.

C. Precomputation Algorithm

We now describe the algorithm to solve the spanning tree algorithm described in the previous subsection III-B. It will be executed at *every* non-leaf node. At node a_i, it calculates a spanning tree among its parent a_{i-1} and the subtrees rooted at its children c_j's $(0 \leq j \leq n-1)$. From the tree, a_i derives the parents-to-be for its children. It will then inform its children of their respective parents-to-be.

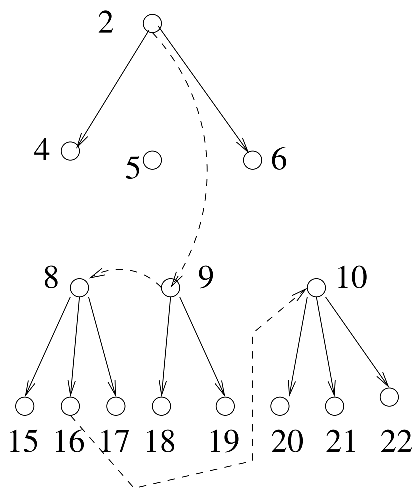

Fig. 3. Finding a spanning tree

In the algorithm, we use variable $d_a(x)$, the *available degree* of node x, to represent the number of links that can further be connected to node x in the tree to be formed. If x is a set of nodes, it means the total of the available degrees of nodes in set x.

In Fig. 4, we outline the algorithm based on Prim's algorithm to set up a spanning tree. We first calculate the total residual degree (D_R) of the children of the current node. If it is not large enough, we find those descendants of its children to make the total greater than or equal to $n-1$. In step 6, we calculate the degree constraints for each node in the spanning tree to be formed. Then in steps 7 and 8, we select the child that has the smallest latency from the grandparent to it and adjust its available degree. Similar to the Prim's algorithm, we have two sets, set A representing those nodes already selected, and set B representing child nodes to be selected. Then in steps 12 to 15, we chose the smallest edge that connects one node in set A to one node in set B, satisfying the degree constraints. In step 16, we add the edge to the tree and adjust the variables.

One remaining question is how to find the descendants of $c_0, c_1, \cdots, c_{n-1}$, so that the total residual degree is greater than or equal to $n-1$, as described in step 3. We assume that the node has the information about the residual degree ($d_r(c_j)$) of each of its children, and the total residual degree ($d_r(T_j)$) of the subtrees rooted at each of its children. How this information is obtained and updated by each node will be described in the next subsection. Each node will use the residual degrees of all of its children (D_R) first, and then find the descendants of its children for the difference, i.e., $G = n - 1 - D_R$. We describe an algorithm in Appendix B to allocate G among subtrees rooted at each child. The inputs also include z_j's, where $z_j = d_r(T_j) - d_r(c_j)$, is the total residual degree of descendants of child c_j. The outputs are y_j's, such that $\sum_{j=0}^{n-1} y_j = G$ and the minimum of $(z_j - y_j)$'s is maximized. For the sake of clarity of the description, we

Precomputation Algorithm

1: Calculate the residual degree $D_R = \sum_{j=0}^{n-1} d(c_j)$.
2: **if** $D_R < n-1$ **then**
3: Invoke the process to find the descendants of $c_0, c_1, \cdots, c_{n-1}$, such that $\sum_{j=0}^{n-1}(d_r(c_j) + \sum_{l=0}^{l_j-1} d_r(c_{jl})) \geq n-1$.
4: **else**
5: set $l_j = 0$ for $0 \leq j \leq n-1$.
6: Initialize the available degree: $d_a(c_j) = d_r(c_j) + 1$ for $0 \leq j \leq n-1$, and $d_a(c_{jl}) = d_r(c_{jl})$ for $0 \leq j \leq n-1$ and $0 \leq l \leq l_j - 1$.
7: Select c_{j_1} among $\{c_0, c_1, \cdots, c_{n-1}\}$, such that $w(a_{i-1}, c_{j_1})$ is the smallest, and $d_a(c_{j_1}) + \sum_{l=0}^{l_{j_1}-1} d_a(c_{j_1 l}) > 1$ if $n > 1$.
8: Record edge (a_{i-1}, c_{j_1}); Set $d_a(c_{j_1}) = d_a(c_{j_1}) - 1$.
9: Initialize $A = \{c_{j_1}\} \cup \{c_{j_1 l} : 0 \leq l \leq l_{j_1} - 1\}$.
10: Initialize $B = \{c_j : 0 \leq j \leq n-1 \wedge j \neq j_1\}$
11: **while** $B \neq \emptyset$ **do**
12: **if** B has a single element **then**
13: Assume the single element is c_{j_2}.
Select $u \in A$, such that weight $w(u, c_{j_2})$ is the smallest among those u's with $d_a(u) \geq 1$.
14: **else**
15: Select edge (u, c_{j_2}), where $u \in A,\ c_{j_2} \in B$, such that weight $w(u, c_{j_2})$ is the smallest among those u's and c_{j_2}'s with $d_a(A \cup \{c_{j_2}\} \cup \{c_{j_2 l} : 0 \leq l \leq l_{j_2} - 1\}) \geq 3$.
16: Record edge (u, c_{j_2});
Set $d_a(u) = d_a(u) - 1$ and $d_a(c_{j_2}) = d_a(c_{j_2}) - 1$;
Set $A = A \cup \{c_{j_2}\} \cup \{c_{j_2 l} : 0 \leq l \leq l_{j_2} - 1\}$ and $B = B - \{c_{j_2}\}$.

Fig. 4. Precomputation Algorithm

assume that z_j's are in the decreasing order [1].

For each $y_j > 0$, the node will ask its child c_j to find a set of its descendants with the total residual degree of y_j. This is a recursive process until we find l_j descendants of c_j. Assume they are c_{jl}'s, where $0 \leq l \leq l_j - 1$.

Other information we need for the modified spanning tree algorithm is the weight, which is the latency between two nodes. The current node will send a *MEASUREMENT* request to all of its children, and possibly those c_{jl} selected. This request contains a list of addresses of its children. When child c_j receives the *MEASUREMENT* request, it will measure the latency from its grandparent to it, and if $d_r(c_j) > 0$, it will also measure its latencies to all of its siblings. When c_{jl} receives the *MEASUREMENT* request, it will measure its latencies to c_k's ($0 \leq k \leq n-1 \wedge k \neq j$). All these distance information will be sent back to the node making the *MEASUREMENT* request.

[1] We can sort z_j's in the decreasing order first and record the mapping. After we get the solution y_j's, we then use the inverse mapping to get the corresponding y_j for the original z_j.

D. Recovery Protocol

We now describe a protocol for nodes to repair the overlay multicast tree once node failures really happen. We will also cover the methods used by those nodes to obtain the information needed for the modified spanning tree algorithm, and to communicate the results of the algorithm to other relevant nodes.

Each node knows the addresses of its parent and children. In addition, it maintains the following information: 1) a list of ancestors, from the root to its grandparent; 2) the parent-to-be, if any; 3) the residual degree of each child; 4) the total residual degree of the nodes in the subtree rooted at each child.

When a node joins a multicast session or rejoins it after the parent failure, it will obtain a list of ancestors from its parent. Its parent-to-be will be initialized to empty. Also it reports its residual degree and the total residual degree of nodes in the subtree rooted at it. A degree report from any child will trigger a degree report to a higher level. Once a non-leaf node generates a rescue plan by running the algorithm in the previous subsection, it will inform its children of their respective parents-to-be.

We assume that there is a heartbeat mechanism for nodes to detect the departure of their parent and children. When the parent leaves, each of its children needs to contact a new parent by sending a *JOIN* message. The protocol can be summarized as follows.

- Upon receiving a *JOIN* request, 1) if its residual degree is greater than or equal to 1, it will **accept** the request; otherwise, 2) if it has children, it will **redirect** the request to the child with the largest residual degree, with the total residual degree of the subtree rooted at the child as a tie breaker; otherwise, 3) it will **reject** the request. The response will be sent to the node making the request.
- Upon detecting a change in children (a new node joins as a child, a child leaves, or the residual degree changes), it may initiate the pre-computation algorithm described in the previous subsection. When the algorithm finishes, it will inform its children of their respective new *parents-to-be*. It will not initiate the pre-computation algorithm if the joined node is its first child or the leaving node is the last child. It will delay the pre-computation if it expects that joins and leaves of other children will happen soon. For example, after a child leaves, it may expect that its grandchildren, if any, will join soon.
- Upon detecting the parent leaving, the node will start the join process. If the *parent-to-be* is not empty, it will first send a *JOIN* request to the parent-to-be and set a timer. If the parent-to-be is empty, or the *JOIN* request is **rejected**, or the timer expires, it tries to join one of its ancestors, starting from the grandparent (skip the grandparent if it is the same as the parent-to-be), until the oldest ancestor – the root. If the *JOIN* request is **redirected**, it will send a new *JOIN* request to the target contained in the redirect response. If the *JOIN* request is **accepted**, it will choose the node as the new parent. It can obtain a new list of ancestors from the parent, and reports its residual degree and the total residual degree of the subtree to the parent. The parent-to-be is initialized to empty and may be updated when it receives information from the new parent in the future.

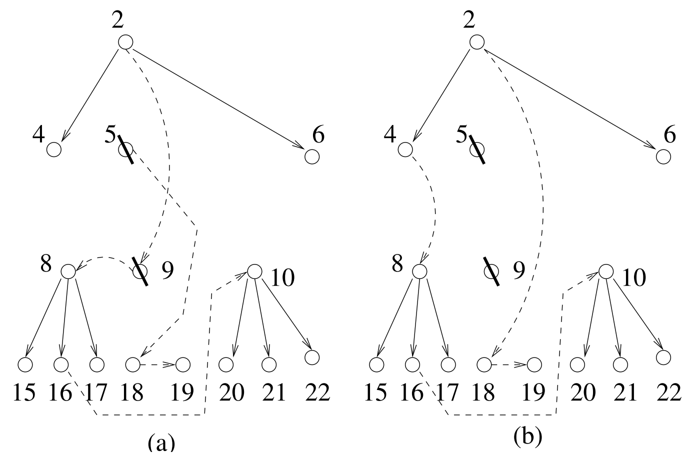

Fig. 5. Using parents-to-be partially

E. Analysis

In most cases, a node can find a new parent by contacting the parent-to-be immediately after the current parent leaves. We want to point out that the protocol still works even if the parent-to-be information is not available at a node. Here are some cases. First, a node just joins a tree. Its parent has not been able to finish the computation and leaves the tree before informing it of its *parent-to-be*. Second, it is possible that the parent was too busy with the delivery task and scheduled this background proactive computation for a later time. In these situations, the ancestor list will be used for these nodes to find a new parent as described in the above protocol. The difference of our approach is that we also try to use the degree information about downstream nodes and redirect JOIN request to a subtree with a high probability of success. Randomly selecting a child to redirect the request may result in a reject at the leaf node because the degree of the leaf node may be exhausted.

In some rare cases, even if the parent-to-be information is available at a node, we still need to use the ancestor list for finding a new parent. This will happen when the current parent and the parent-to-be of a node fail at the same time. Node 18 in Fig. 5(a) is such an example. We use dotted lines to show the parents-to-be of relevant nodes. Assume that the rescue plan for the failure of node 5 is that the parents-to-be of its children 8, 9, 10 are 9, 2, and 16, respectively. The rescue plan for the failure of node 9 is that the parents-to-be of its children 18 and 19 are 5 and 18, respectively. Once nodes 5 and 9 fail at the same time, node 18 will try to contact its parent-to-be (node 5). However, it will not get any reply and will timeout. It then contacts the next one in its ancestor list – node 2 and is accepted as a child. While node 8 tries to contact its parent-to-be (node 9) after detecting the failure of

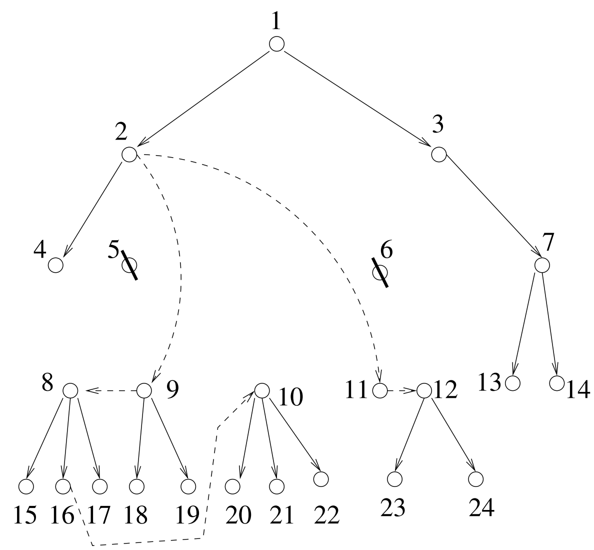

Fig. 6. Multiple failures at the same time

its parent (node 5), it will not get the reply back either. Then it contacts its grandparent and will be redirected to node 4. The final tree is shown in Fig. 5(b). An interesting point is that the parent-to-be information can still be used. For example, node 19 is able to connect to its parent-to-be (node 18) and successfully find a new parent, and node 10 is able to connect to its parent-to-be (node 16).

A last observation is that the protocol can deal with most multiple failures cases efficiently. For example, in Fig. 1, if both nodes 5 and 7 fail, their respective set of children will be able to recover independently. Even if two siblings fail at the same time, their children can still find their parents-to-be immediately without interference. Look at a case in which nodes 5 and 6 in Fig. 1 both fail. Fig. 6 shows the rescue plans for node 5 and node 6. Once nodes 5 and 6 fail, their children will be able to find their respective parents-to-be. We notice the importance of keeping the degree requirement for the grandparent the same before and after the node failure. The children of node 5 and the children of node 6 will not compete the resources at their grandparent in this case. Every node can find their respective parents-to-be after node failures successfully.

IV. Performance Evaluations

We evaluate the performance of the proactive approach using simulations. We are mainly interested in the *responsiveness* of various approaches, i.e., how fast the delivery tree can be restored. Other interesting measures are the *quality* of the tree reconstructed, and the *overhead* of the recovery process. We compare our proactive scheme with several reactive methods, including *grandfather*, *grandfather-all*, *root*, and *root-all*, as described in Section II. We also compare the overhead with PRM [6].

A. Simulation Setup

In the simulation, we use GT-ITM [14] to generate 1600 node transit-stub topologies as the underlying network. One stub node is randomly chosen as the source and other end-hosts are randomly distributed in stub domains of the network. The total number of end-hosts varies from 80 to 400 in our experiments. The weights of the edges in the graph represent the network-layer link latencies, which range from 1ms to 245ms. The application-level distance 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. It represents the maximal number of application layer connections it can establish with other end hosts. Note this is not the degree in the graph for the network layer topology.

We assume that the overlay multicast is used for live media streaming. All experiments begin with a delivery tree established by a heuristic for solving the degree-constrained minimum spanning tree problem [9]. Then end-hosts join or leave the tree dynamically. The join and leave are both modeled as a Poisson process with rate $\lambda = 4/minute$, which means that on average, there is a node joining and a node leaving the overlay tree every 15 seconds. Each experiment lasts for two hours.

B. Responsiveness

Responsiveness indicates how fast each scheme can restore the delivery tree after a node fails or leaves the tree. We use the average recovery time as a performance measure. It is the average time for an affected node to find a new parent. For each node, we calculate the sum of latencies of the links the control message travels for finding the new parent as its recovery time.

Fig. 7 plots the average recovery time. The average recovery time for the four reactive methods ranges from 459ms to as high as 2900ms. The average recovery time of the proactive scheme is always less than 400ms and can be as low as 225ms. The average recovery time of the proactive scheme is only about half of the time of the best reactive approach – *grandfather*. As the number of end-hosts increases from 80 to 400, the recovery time for *proactive* and *grandfather* methods slightly decreases. One possible reason is that the average distance between two nodes is shorter. So the children of the failed node have a shorter distance to their grandparent and can quickly find their new parents. However, the recovery time increases for other three methods as the number of end hosts increases. The reason is that the nodes close to the root or the grandparent are more saturated when there are more nodes in the delivery tree. So most affected descendents have to search for more nodes before finding a new parent.

Fig. 8 depicts the cumulative distribution of the recovery time when the number of end hosts is 240. We can find that for the proactive scheme, less than 1% recovery time is more than 500ms. However, the *grandfather* method has more than 20% recoveries using more than 500ms, while other reactive methods have more than 20% recoveries with more than 2500ms.

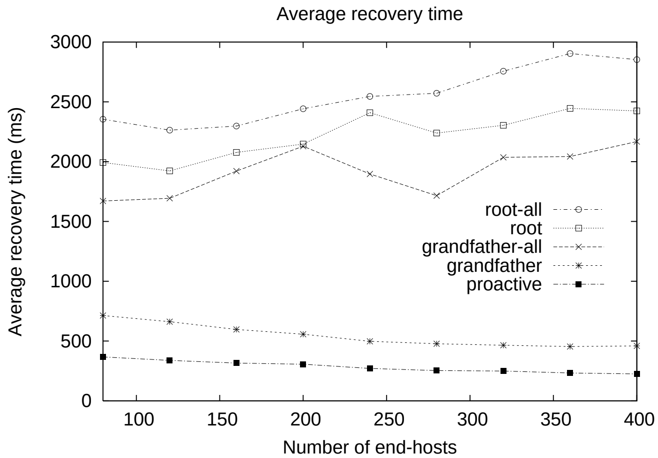

Fig. 7. Average recovery time

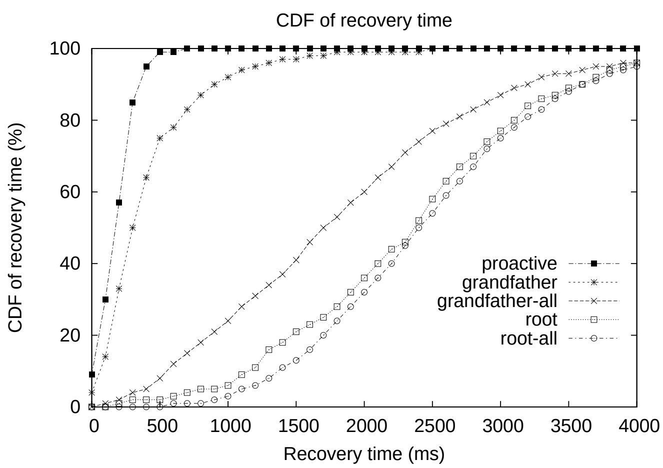

Fig. 8. Cumulative distribution of the recovery time

A bigger difference is observed when we look at the maximum recovery time, as shown in Table I. While the maximum recovery time for the proactive scheme is always less than 950ms, the maximum recovery time for the reactive methods can be as large as 7906ms. Even the best reactive method (*grandfather*) has a maximum recovery time of 4742ms in the case of 160 end hosts, more than 5 times that of the proactive scheme.

Another indirect measure of responsiveness is the average number of end hosts that need to be contacted during the recovery process, as shown in Fig. 9. It demonstrates similar pattern as Fig. 7. We can find that the number of end hosts contacted in the proactive scheme is almost equal to 1. This means that an affected node only needs to contact one node to find its new parent in most cases. The average number of end hosts contacted by the reactive methods ranges from 1.6 to more than 5. This partly explains why the reactive approach is less responsive than the proactive approach.

Schemes	Number of end-hosts				
	80	160	240	320	400
proactive	944	850	772	692	594
grandfather	5406	4742	2572	3612	2827
root	5574	4790	6018	4616	5992
grandfather-all	5532	7008	6560	7330	7906
root-all	6230	5074	6150	6974	7506

TABLE I

MAXIMUM RECOVERY TIME (MS)

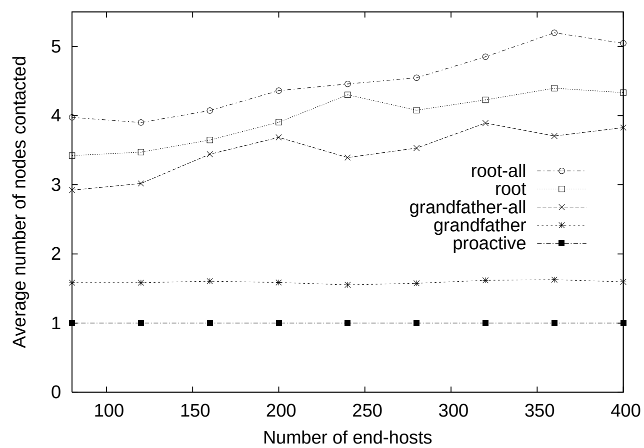

Fig. 9. The number of end hosts contacted

C. *Quality of the Restored Tree*

The quality of the restored tree can be measured in two aspects. One is the tree cost, which measures the resource usage of the tree. The other is the maximum delay of the nodes in the tree from the source. Fig. 10 plots the average tree cost when the number of end hosts in the tree varies from 80 to 400. After the recovery from each node departure, we calculate the total tree cost, which is the sum of weights (latencies) of all application layer links in the tree. These costs are averaged over all the restored trees generated during the experiment time. When the number of end hosts in the tree increases, the total cost increases in all cases. We observe that the *proactive* scheme has the smallest average tree cost. Only *grandfather* has a comparable tree cost. Other reactive methods generate trees that double or triple the cost of the trees generated by the proactive scheme.

Also depicted in Fig. 10 is the lower bound of the average tree cost. After each node departure, we use the global spanning tree algorithm to calculate the tree from scratch and record its cost. The average of these tree costs is the best we can use as the lower bound. We can find that the tree cost generated by the proactive method is very close to the lower bound in all cases.

Fig. 11 shows the maximum delay of the restored tree. Interestingly the maximum delay of the nodes in the tree generated by the lower bound algorithm is also relatively short. We can find that the maximum delay of the proactive scheme is

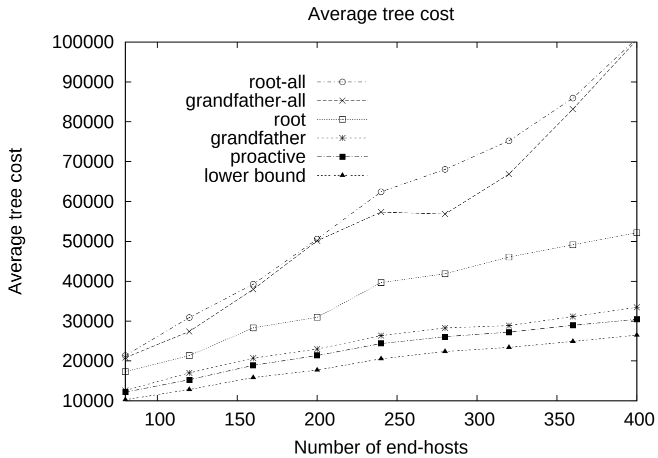

Fig. 10. Average tree cost with varying number of end-hosts

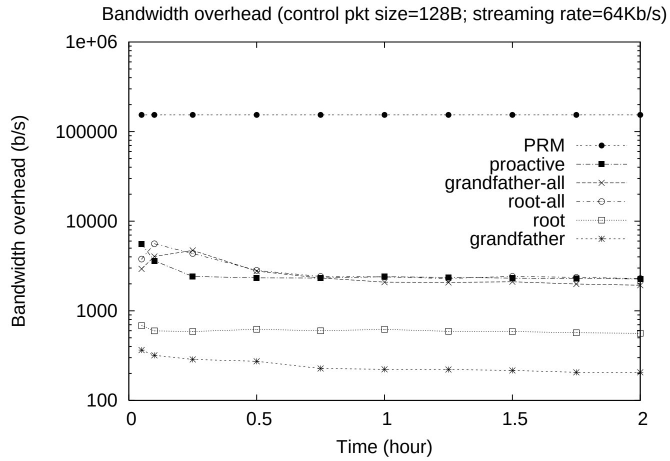

Fig. 12. Average bandwidth overhead

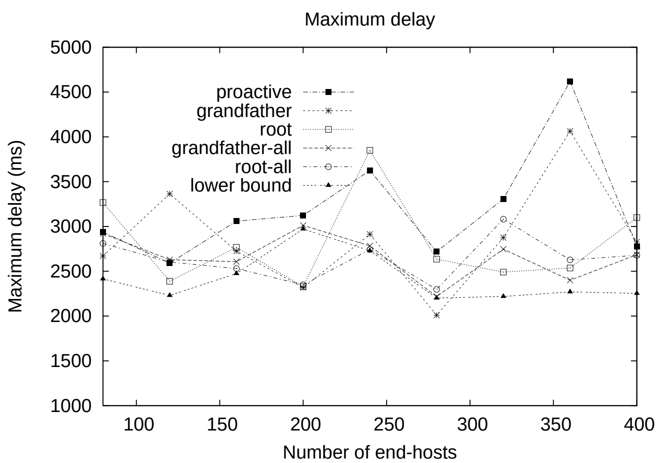

Fig. 11. Maximum delay from the root to leaves

comparable to that of the trees generated by other approaches.

D. Recovery Overhead

Now we analyze the recovery overhead by measuring the bandwidth consumed for the recovery purpose. It is the average number of bytes per second sent in the multicast tree, in addition to the normal streaming data.

For the reactive methods, the control overhead comes from the control messages exchanged for the affected nodes to find new parents. For the proactive scheme, the control messages consists of two parts. 1) Similar to reactive methods, control messages are exchanged for the children of failed nodes to find new parents, though we may need fewer steps. 2) In addition, every non-leaf node (except the root) precalculates parents-to-be for its children. The control messages are needed to exchange degree and distance information as the input to the precomputation algorithm. They are also used to communicate the outputs (parents-to-be) of the algorithm to the children. Note this part includes the overhead of rerunning the precomputation algorithm after nodes join and leave.

Fig. 12 compares the overhead when the average number of end-hosts in the multicast session is 240. After each node leave, we calculate the total traffic up to now and divide it by the time. Initially, the *proactive* approach has a very high overhead, because all the precomputation traffic is averaged over a short period of time. As the time goes on, the number of node leave events increases and the amortized cost decreases. After half an hour, the overhead of the proactive approach is very close to the *grandfather-all* and *root-all* methods.

We also show the overhead of Probabilistic Resilient Multicast (PRM) method [6], the proactive method working on the data plane for overlay multicast. We set the parameters to low end numbers, that is, every node sends data with a very low probability (1%) to only 1 extra node, and the data streaming rate is $64kb/s$. We can see from the plot that it still incurs much higher traffic overhead than other schemes. We observe that even the highest point in the "proactive" curve is only about $1/28$ of that of PRM. The difference in data rate on the control plane and data plane is very large.

Fig. 13 depicts the average traffic generated as the number of end-hosts in the session is varied. In PRM, the bandwidth overhead increases linearly with the multicast group size, while the curve for the *proactive* method remains flat and generates almost the same volume of traffic as the *grandfather-all* and *root-all* methods. Its very slow increase also indicates that the proactive scheme has a good scalability as the group size increases.

V. Related Work

Overlay multicast network has been studied extensively in recent years. Early work on overlay multicast includes Yoid [2] and Scattercast [3], and the end system multicast [1]. More recently, degree constraints are considered in establishing multicast tree [15], [16]. Various degree-constrained spanning tree problems are defined and usually they are NP-complete. Heuristics are given to solve these problems. The degree-

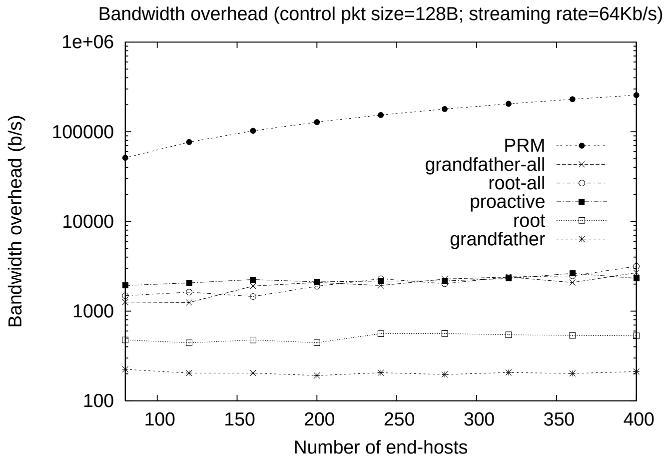

Fig. 13. Average bandwidth overhead with varying number of nodes

constraint in multicast communication can be dated back to the early work by Bauer and Varma [10]. An evolutionary algorithm was given by Raidl to solve the degree-constrained minimum spanning tree problem [8]. All these schemes are centralized solutions to the problem. A distributed method [17] is given by Banerjee et al, but the problem is to minimize the average delay rather than the total cost.

The proactive approach has been used in recovering link or node failures in multicast tree in the context of the traditional *network-layer* multicast. The fault-tolerant multicast routing [7] proposed to use backup paths from the grandparent to deal with link or path failures. More recently, a dual-tree scheme [18] proposed that in addition to the primary tree normally used, a secondary tree is calculated that connects the leaf nodes in the primary tree together. Once a failure occurs, it will activate a path in the secondary tree. It needs to recalculate the secondary tree using global knowledge once the recovery is complete. Their difference from our work is that they dealt with the *network-layer* fault-tolerant multicast routing problem and did not consider the important degree constraints on the nodes in overlay multicast.

The problem of dealing with node failures in overlay multicast has been recognized in more recent work [4]–[6]. SpreadIt [4], [5] proposed *reactive* strategies to deal with node leave or failure in overlay multicast. They find appropriate places in the subtree of the grandparent or root for the affected nodes after failure happens. Because it uses the reactive approach, the time to find an appropriate place may be long and those affected nodes may even compete with each other.

Resilient multicast using overlays [6] uses proactive method for overlay multicast. It is complementary to our work because it works on the data plane while our scheme works on the control plane. Our scheme generates less extra traffic than their work. We also discuss the restoration of the multicast tree. We can make use of their idea of probabilistic forwarding to improve the transient behavior of our scheme.

VI. Concluding Remarks

Overlay multicast differs from traditional IP multicast in that the problem of degree constraints is more prominent and non-leaf nodes in the multicast tree are unstable. This makes the problem of restoring multicast tree after node failures or leaves quite different. This paper proposed a proactive approach which lets each non-leaf node pre-compute the recovery plan for its children in case it fails. We develop a protocol for nodes to communicate with each other and can deal with various failure situations. The recovery process is much faster than reactive approaches while the quality of the tree and the amortized cost is comparable to those methods.

References

[1] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM Sigmetrics*, June 2000, Santa Clara, CA.

[2] P. Francis, "Yoid: Extending the Internet multicast architecture," http://www.icir.org/yoid/docs/yoidArch.ps.

[3] Y. Chawathe, S. McCanne, and E. A. Brewer, "An architecture for Internet content distribution as an infrastructure service," Feb. 2000, available at http://www.cs.berkeley.edu/ yatin/papers/scattercast.ps.

[4] M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers and streaming media," in *Proceedings of HotNets 2002*, October 2002, princeton, NJ.

[5] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peer-to-peer network," 2001, technical Report CS-2001-31, CS Dept. Stanford University.

[6] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," in *Proceedings of ACM Sigmetrics 2003*, June 2003, san Diego, CA.

[7] W. Jia, W. Zhao, D. Xuan, and G. Xu, "An efficient fault-tolerant multicast routing protocol with core-based tree techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 984–999, October 1999.

[8] G. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*. La Jolla, California, USA: IEEE Press, 6-9, pp. 104–111.

[9] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning trees," *Computers and Operations Research*, vol. 7, pp. 239–249, 1980.

[10] F. Bauer and A. Varma, "Degree-constrained multicasting in point-to-point networks," in *Proceedings of INFOCOM'95*, April 1995, boston, MA.

[11] M. R. Garey and D. S. Johnson, Eds., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, second edition*. The MIT Press, 2001.

[13] Z. Fei, E. Zegura, and M. Ammar, "Multicast server selection: problems, complexity and solutions," *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 1399–1413, September 2002.

[14] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of INFOCOM'96*, 1996, pp. 594 –602, san Francisco, CA.

[15] S. Y. Shi, J. S. Turner, and M. Waldvogel, "Dimensioning server access bandwidth and multicast routing in overlay networks," in *Proceedings of NOSSDAV*, June 2001.

[16] S. Y. Shi and J. S. Turner, "Routing in overlay multicast networks," in *Proceedings of INFOCOM 2002*, June 2002, new York, NY.

[17] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proceedings of IEEE INFOCOM 2003*, March 2003, san Francisco, CA.

[18] A. Fei, J. Cui, M. Gerla, and D. Cavendish, "A dual-tree scheme for fault-tolerant multicast," in *Proceedings of ICC 2001*, June 2001, helsinki, Finland.

Appendix

A. Proofs of the Theorems

Theorem 1: For the degree-constrained spanning tree problem of a complete graph $G = (V, E)$ with m nodes and degree constraints k_i for node i $(0 \leq i \leq m-1)$, it has a feasible solution *if and only if* $k_i \geq 1$ and $\sum_{i=0}^{m-1} k_i \geq 2m-2$.

Proof: A m-node spanning tree has $m-1$ edges. Each edge results in 2 degrees used, one for each node. So the total used degree must be $2(m-1) = 2m-2$. Also each node must be connected, and thus $k_i \geq 1$. So they are necessary conditions.

Conversely, assume $k_i \geq 1$ and $\sum_{i=0}^{m-1} k_i \geq 2m-2$. For $m = 2$, we can obviously generate a feasible tree by connecting the two nodes. For a graph with m $(m > 2)$ nodes, we assume that node v has the smallest degree constraint k_v. If $k_v = 1$, we know the total degree of the rest $m-1$ nodes will be $(\sum_{i=0}^{m-1} k_i) - k_v \geq 2m-3$. If $k_v \geq 2$, then $k_i \geq 2$ for $0 \leq i \leq m-1 \wedge i \neq v$. We also have $(\sum_{i=0}^{m-1} k_i) - k_v \geq 2m-3$. So we can construct a spanning tree using the rest of $m-1$ nodes and the total used degree of the tree is $2m-4$. This tree must have a node with free degree of at least 1. Connecting this node with node v will form a spanning tree with m nodes. Thus they are sufficient conditions.

Theorem 2: Assume the residual degree of node c_j $(0 \leq j \leq n-1)$ is $d_r(c_j)$ in the current tree, we can form a feasible spanning tree among a_{i-1} and c_j's $(0 \leq j \leq n-1)$ with the degree constraint on a_{i-1} equal to $k_{a_{i-1}} = 1$, and the degree constraint on c_j equal to $k_{c_j} = d_r(c_j) + 1$, *if and only if* $\sum_{j=0}^{n-1} d_r(c_j) \geq n-1$.

Proof: We have $n+1$ nodes in this case. From Theorem 1, we know the sufficient and necessary condition is that the degree constraint for each node is greater than or equal to 1 and the total degree constraint is greater than or equal to $2(n+1)-2$. Obviously we have $k_{a_{i-1}} \geq 1$ and $k_{c_j} \geq 1$ because they all get one degree released from the link connecting it with node a_i. The total of the degree constraints is $k_{a_{i-1}} + \sum_{j=0}^{n-1} k_{c_j} = 1 + \sum_{j=0}^{n-1}(d_r(c_j)+1) = n+1+\sum_{j=0}^{n-1} d_r(c_j)$. It is greater than or equal to $2(n+1)-2$ if and only if $n+1+\sum_{j=0}^{n-1} d_r(c_j) \geq 2(n+1)-2$, that is $\sum_{j=0}^{n-1} d_r(c_j) \geq n-1$.

B. Degree Assignment Algorithm

INPUT: z_0 to z_{n-1} in decreasing order, target G.
OUTPUT: y_0 to y_{n-1}.

for $i = 0$ to $n-1$ **do** $x_i = z_i$;
if $\sum_{j=0}^{n-1} x_i < G$ **then**
  return(no solution);
$x_n = 0$;
for $i = 0$ to $n-1$ **do** $y_i = 0$;
$j = 0$; $Q = 0$;
while $(Q < G)$ **do**
  while $(x_j == x_{j+1})$ **do** $j = j+1$;
  $\delta = x_j - x_{j+1}$;
  if $(Q + \delta * (j+1) \leq G)$ **then**
    $Q = Q + \delta * (j+1)$;
    for $i = 0$ to j **do**
      $x_i = x_i - \delta$; $y_i = y_i + \delta$;
  else
    $dd = \lfloor (G-Q)/(j+1) \rfloor$;
    $rr = G - Q - dd * (j+1)$;
    for $i = 0$ to $rr-1$ **do**
      $x_i = x_i - (dd+1)$; $y_i = y_i + (dd+1)$;
    for $i = rr$ to j **do**
      $x_i = x_i - dd$; $y_i = y_i + dd$;
    break;
return(y);