Stability Oriented Overlay Multicast for Multimedia Streaming in Multiple Source Context

Thilmee M. Baduge, Kazushi Ikeda, Hirozumi Yamaguchi and Teruo Higashino
Graduate School of Information Science and Technology, Osaka University
1-5 Yamadaoka, Suita, Osaka 565-0871, JAPAN
Email:{thilmee, k-ikeda, h-yamagu, higashino}@ist.osaka-u.ac.jp

Abstract—**In this paper, we propose a new overlay multicast protocol designed for inter-active multimedia streaming applications. The protocol considers the heterogeneity of end-hosts and tries to minimize the negative impact (data outage) of end-hosts' unannounced departures. For this purpose, it concentrates on end-hosts' reliability (lifetime for instance) and constructs a shared tree called ms-DDBMSST (multiple-source Degree and Delay Bounded Maximum Stability Spanning Tree) as an overlay network that involves all the participants of the application, in a distributed manner. For a given set of nodes where some of them are senders, ms-DDBMSST is a spanning tree where the receive path stability of the entire tree is maximized while satisfying the delay-from-source constraint and degree constraint for each node. We believe that this is the first approach that defines ms-DDBMSST construction problem and presents a distributed protocol for the purpose. Our performance evaluation is based on experiments in both simulated networks and PlanetLab that strongly shows the efficiency and usefulness of the proposed protocol.**

I. Introduction

Overlay multicast (Application Layer Multicast, ALM in short) has become an important research topic in past decade because of its flexibility. This flexibility is caused by ALM's nature of using end-hosts to perform one-to-many data forwarding, which is known as multicast. ALM has the biggest potential to realize the small to large scale group communication infrastructures such as Internet games, white-board, audio or video streaming. In contrast, reliability problems of end-hosts in ALM induce the one of major drawbacks in terms of spreading through real world applications. One major issue that comes up here is the heterogeneity of end-hosts (referred to as nodes below) to the ALM session in which it is involved; since less-desired nodes leave the session sooner and often without any prior notification or cause delay and harmful jitter even they stay the session, the participants in the downstream suffer considerable degradation of quality. This paper addresses this issue, the *reliability* of nodes. An example metric is *lifetime* of nodes, which helps to build reliable and stable ALM. We argue that this kind of node specific factors, which are independent from the topological factors, have a great influence on the efficiency of the underlying ALM scheme.

Meanwhile, we also consider the topological factors, which are also important to support interactive multimedia applications. We can point out the characteristics of interactive multimedia applications as follows. (1) Such an application may have several *sources*. For example, in video-conferencing, pictures of some primary persons should be continuously delivered to the other audience. (2) The delays from these sources to each sender is bounded by a maximum delay constraint to make each node interactive. (3) Each node is associated with a bandwidth limitation (or degree in other words), where the number of outgoing streams that can be handled by that node is restricted. Considering the heterogeneity of Internet end-hosts, multicast topology may be composed with nodes with variety of degrees including the large portion of *zero-contributors* [1]. And (4) the nodes may show different desires. For example, some survey studies conducted with real world data traces have shown that the older nodes have longer residual lifetimes [1]. This property helps to predict nodes remaining lifetime to make the multicast topology more robust to node departures.

In this paper, an ALM scheme to meet the above requirements is proposed which aims at constructing a *multiple-sourced* tree T, minimizing the bad affect caused by the heterogeneity of nodes' reliability on the entire tree. At the same time it assures each node's delay-from-sources to be within the provided delay constraint under degree constraints. The contributions of this paper can be summarized into, (i) formulation of a new problem associating nodes' reliability factors such as lifetime with delay and degree constraints in a multiple-source context, (ii) proposing of a decentralized heuristic algorithm which gradually transforms a simple initial tree to the targeted tree, and (iii) discussion of the extensive experiments conducted in PlanetLab to show the proposal's usability.

II. Related work

A lot of ALM schemes have been proposed so far targeting the variety of Internet group applications in the current world. They can be categorized into the following 3 major categories focusing on each one's design concern: (i) latency-first approaches, (ii) restore-first approaches and (iii) other QoS concerned approaches. The first one focuses on minimizing the latency between nodes. Generally tree based approaches like [2], [3], [4] are considered to meet these latency requirements. The second one mainly focuses on restoring the overlay topology in case of collapse due to nodes' leaving. This is also critical when using ALM in real world applications. Multiple path approaches where the redundant paths are used in case of original path failure ([5], [6]), or restore schemes where the overlay topology is re-established using a previously or immediately calculated restore mission ([7], [8], [9]) are often used to realize this. The third one corresponds to the ALM schemes like [10], [11], [12], which are targeted to provide some other QoS features like bandwidth.

All the above schemes have only considered topological factors like node-to-node overlay link latency and node degree in their protocols, and hence non-topological characteristics such as heterogeneity of node lifetime and forwarding capabilities have not been addressed for a long time. To our best knowledge, [13] has first addressed this issue inspired by an analysis of some real-world data traces. They have proposed using nodes' lifetime characteristics, where older nodes are selected as peers for new nodes in their *longest-first algorithm*. After that, [14] has also addressed this as "priority", by which nodes lifetime and/or bandwidth are referred. This has conducted simulation experiments with some real-world data traces to find out which metric among lifetime and bandwidth gives the better reliability in terms of affect of node failures, when prioritizing them. Their conclusions state that bandwidth prioritizing performs better. In contrast to the centralized approach taken in [14], [15] proposes a decentralized algorithm to build a reliable tree considering nodes' lifetime.

Our proposal differs from the above lifetime aware approaches in the following aspects. (i) Taking the multiple-source issue into account to make the scheme more applicable to interactive multimedia applications, (ii) associating a delay bound from all sources which is indispensable to realize the interactivity, and (iii) conducting experiments in PlanetLab to verify the usability in real-world.

III. Problem analysis

A. Problem definition

Let $G = (R, E)$ denote a given undirected complete graph where R denotes a set of receiver nodes (or simply nodes) and E denotes a set of potential overlay links which are unicast connections between nodes. Also the followings are given. $d_{max}(r)$ denotes the degree bound of node $r \in R$, $h(i, j)$ denotes the delay of overlay link $(i, j) \in E$, $sc(r)$ denotes the normalized *stability coefficient* of node $r \in R$ ($sc(r) \in [0, 1]$), *e.g.* node lifetime, and S denotes the set of source nodes. We note that a source node is also a receiver node ($S \subseteq R$).

Our goal is to find a spanning tree T of G with maximum *tree stability*, while the maximum overlay delay from S to nodes in R does not exceed a given delay bound D_{max} and degree of each node $r \in R$ (denoted by $d(r)$) does not exceed $d_{max}(r)$. We name this *multiple-source Degree and Delay Bounded Maximum Stability Spanning Tree (ms-DDBMSST)* construction problem. The tree stability of T (denoted as $stab_T$) is defined as the sum of *path stabilities* of all the source-receiver paths on T. The path stability of the source-receiver path from $s \in S$ to $r \in R$ on T (denoted as $stab_T(s, r)$) is the multiplication of the stability coefficients of nodes on the path. Their formal definitions are given below.

$$\begin{aligned} stab_T &= \sum_{(s,r)\in S\times R} stab_T(s, r) \\ stab_T(s, r) &= \begin{cases} 1 & (s = r) \\ sc(r') \cdot stab_T(s, r') & (s \neq r) \end{cases} \end{aligned}$$

Here, r' is the upstream neighbor of r on the path from s to r on T.

The decision problem of ms-DDBMSST, that is, the problem to find a degree and delay bounded, stability-bounded spanning tree, is NP-complete. To prove this, we first need to say that for any tree on G it can be verified in polynomial time whether the tree satisfies the given bounds of degree, delay and tree stability. Obviously this holds. Then we set the number of source nodes and the stability coefficient of each node to one. Then the *Degree and Delay Bounded Spanning Tree* decision problem, which is known as a NP-complete problem [16], can be transformed to our ms-DDBMSST decision problem in polynomial time.

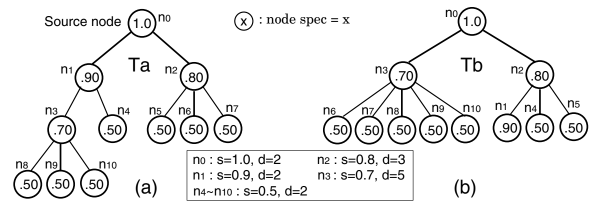

Fig. 1. Two simple methods that construct DBMSST: (a) stability-first (s-first) and (b) stability-degree product-first ($s{\cdot}d$-first)

Further, *Degree Bounded Maximum Stability Spanning Tree (DBMSST)* construction problem has also been addressed in few previous lifetime-aware schemes ([13], [14], [15]). The DBMSST construction problem is a version of the ms-DDBMSST construction problem, and it is considerably simplified from the ms-DDBMSST construction problem in such a way that delay is unbounded and the number of source nodes is set to one. Though this problem is not our key concern, some characteristics and methodologies discussed here are common with our ms-DDBMSST construction. More precisely, the ms-DDBMSST construction described in section IV consists of a refinement phase, where nodes with ancestor-descendant relationship are swapped to improve the stability of tree. Therefore, it is important to know which property of nodes should be considered for this swapping and the rest of this section discusses which property of nodes should be prioritized to maximize the stability of tree.

B. Learning from DBMSST

1) Existing approaches: Fig. 1 illustrates two simple methods of constructing DBMSST presented in [14]. The stability coefficients and degrees are given in the box of Fig. 1. In Fig. 1(a) the tree is constructed by greedily choosing the nodes with maximum stability coefficient (this is called s-first), while Fig. 1(b) chooses the ones with maximum stability-degree product (called $s{\cdot}d$-first). The chosen nodes are connected to the current tree at the peer node which provides the maximum path stability in both methods. The tree stability of T_a is calculated as $stab_{T_a}$=$\sum_{i=1}^{10} stab_{T_a}(n_0, n_i)$= $(1.0{\cdot}1) + (1.0{\cdot}1) + (0.9{\cdot}1.0) + (0.9{\cdot}1.0) + (0.8{\cdot}1.0) + (0.8{\cdot}1.0) + (0.8{\cdot}1.0) + (0.7{\cdot}0.9{\cdot}1.0) + (0.7{\cdot}0.9{\cdot}1.0) + (0.7{\cdot}0.9{\cdot}1.0)$ =8.09. In the same way, $stab_{T_b}$ becomes 7.9, and that says s-first approach is better for this example.

2) Our approach to DBMSST problem: We describe a novel and better centralized mechanism, which considers the remaining node count in addition to the lifetime and degree. Later, by simulation experiments, we show that this centralized scheme performs better than the s-first and $s{\cdot}d$-first approaches, which have been used in the early-stage lifetime aware schemes such as [13], [14] and [15]. And also we apply the key idea of this centralized approach to the construction of ms-DDBMSST, which is our main concern, where neither s-first nor $s{\cdot}d$-first approach is associated.

This consists of two main steps. First, it simply selects the peer node p from the current tree, where p has the maximum path stability with at least one residual degree. Second, the node for adding to the tree is selected. For this purpose, a new metric called *estimated tree stability* is introduced, and the node that maximizes this value is selected at the second step. This procedure is repeated until all the nodes are added to the tree. This is the key feature in our approach. Note that the s-first and $s{\cdot}d$-first approaches have greedily selected the node with maximum lifetime or maximum lifetime-degree product. In contrast, we estimate the stability of the final tree at each selection step.

We define the estimated tree stability. Hereafter, T_{cur+u} denotes the tree after adding node u to the current tree T_{cur}, R_{rem} denotes the set of nodes that have not been included in T_{cur+u}, $\hat{sc}$ denotes the average stability coefficient of nodes in R_{rem}, $\hat{d}$ denotes the average degree of nodes in R_{rem}, R_{res} denotes the set of nodes that have residual degrees on T_{cur+u} and $outd(t)$ denotes the total out degree of tree t. For the tree T_{cur+u}, we expect that the final tree T is obtained by organizing the nodes in R_{rem} into the set of $outd(T_{cur})$ sub-trees and adding them to T_{cur+u}. For the convenience, we denote these sub-trees by $subT$. Then assuming $sc(w)$=$\hat{sc}$ and $d(w)$=$\hat{d}$ for all $w \in R_{rem}$, we can define the stability of $subT$ as follows (s' is the root node of $subT$).

$$\begin{aligned} stab_{subT} &= \sum_{r' \ on \ subT} stab_{subT}(s', r') \\ &= 1 + \hat{sc} \cdot \hat{d} + \hat{sc}^2 \cdot \hat{d}^2 + \cdots + \hat{sc}^m \cdot \hat{d}^m \\ &= \frac{(\hat{sc} \cdot \hat{d})^{m+1} - 1}{\hat{sc} \cdot \hat{d} - 1} \end{aligned}$$

Here, $m = log_{\hat{d}}|subT|$ where $|subT| = \frac{|R_{rem}|}{outd(T_{cur+u})}$ (m is the number of maximum hops from root in $subT$). The estimated tree stability of the final tree T, $E_u[stab_T]$ can be defined as follows.

$$E_u[stab_T] = stab_{T_{cur+u}} + \sum_{r' \in R_{res}} stab_{subT} \cdot stab_{T_{cur+u}}(s, r') \cdot d(r')$$

Before we add a new node to the tree, $E_u[stab_T]$ is calculated for each $u \in R_{rem}$ and the node that gives maximum value is selected.

IV. ms-DDBMSST construction

Before going deep into ms-DDBMSST construction methodologies, we discuss the nature of our problem. Our mission is to find a spanning tree T that maximizes the tree stability $stab_T$ in a manner where the degree bound of each node is not violated and the delay of any source-receiver path on T does not exceed the given maximum bound D_{max}.

Hereafter we let $depth_T$ denote the maximum delay of source-receiver paths on T. As we have seen before, $stab_T$ is influenced by nodes' stability coefficients and degrees, while overlay link-delays and node degrees affect $depth_T$. This implies that stability coefficients of nodes and delays of overlay links play key roles in optimizing $stab_T$ and $depth_T$ successively. On the other hand, it is known that there is no correlation between node's lifetime and link delays [13] and similar argument can be applied to the case of forwarding capabilities of nodes and link delays. Therefore, it is easily understood that $stab_T$ and $depth_T$ are non-correlated metrics. This makes it clear that ms-DDBMSST construction is a task of optimizing two independent metrics, which prevents us from going for easy solutions, such as using a variation of minimum depth spanning tree algorithm [2] with expressions like $\alpha \cdot stab_T + \beta \cdot depth_T$.

Following the above observations, we apply a two-step tree construction algorithm to separate $stab_T$ and $depth_T$ optimization. There are (a) the *initial tree construction* step that optimizes $depth_T$ and (b) the *tree refining* step that optimizes $stab_T$. Considering the fact that $depth_T$ should not exceed the delay constraint D_{max}, we use the minimum depth spanning tree algorithm [2] to build the initial tree [1]. By this we can achieve almost the best possible $depth_T$ which is probably lower than the given delay constraint D_{max}. Then, the tree refining process takes place by moving nodes throughout the tree so that $stab_T$ is improved. It is easily understood that this refining step makes $depth_T$ increased in higher possibility as it destroys the $depth_T$-optimized initial tree in being transformed into the $stab_T$-optimized one. So we have to make sure that no refining step violates the maximum bound for $depth_T$. Note that if $depth_T$ is greater than D_{max} after the initial tree construction step due to too tight D_{max}, no refining takes place for such cases obviously, and this initial tree remains as it is. Therefore, we assume that D_{max} suitable for real world applications is larger than the $depth_T$ found by the initial tree construction algorithm.

We assume that most of the participant nodes arrive before the session starting time and a few cannot make it to time. So it is feasible to build the initial tree using a centralized algorithm, as mentioned above, because the streaming session is not still started. However, no centralized scheme is preferred once the streaming session is started to maintain a seamless streaming session. Therefore, we go for a decentralized approach as the tree refining procedure. Though there may be inter-session node joinings and departures, we assume that they are handled by some existing decentralized protocols such as [9] and mainly concentrate on the initial tree construction and refining procedures. These procedures are described in detail in the following sections [2].

A. Initial Tree Construction

The initial tree is built in such a way that maximum delay from source nodes ($depth_T$) is minimum. If all the source nodes are located close to each other with small overlay link delays between them, we can treat them as a single source after arranging them into a single group. This makes it easy to build the required tree. In this case we can use minimum depth algorithm [2] to build a spanning tree with minimum delay (depth) from the group of source nodes, because this group stands for the root node of tree. However, generally sources may be located anywhere and do not necessarily have smaller overlay link delays between them. So making them into a group will result in unnecessary overlay delays to the entire tree and may make $depth_T$ larger. Therefore, we refrain from using the grouping concept and use the following algorithm inspired from the minimum depth algorithm. This builds the initial tree starting from a randomly selected source node s_0 as the root node.

[1]Although minimizing the depth is not an objective of our scheme, the construction of minimum depth spanning tree is necessary at this stage to check whether the given maximum depth constraint is reachable.

[2]Considering the scalability, a decentralized minimum delay spanning tree scheme can be used to build the initial tree when the maximum delay constraint is not so strict.

```
01: $T_{node} \Leftarrow \{s_0\}$, $T_{edge} \Leftarrow \emptyset$, $S \Leftarrow \{s_0\}$, $R \Leftarrow R - \{s_0\}$;
02: while ($R \neq \emptyset$)
03:     find $r \in R \backslash T_{node}$ and $r' \in T_{node}$ that minimizes
            $depth_{(T_{node} \cup \{r\}, T_{edge} \cup \{(r', r)\})}$
04:     $T_{node} \Leftarrow T_{node} \cup \{r\}$;
05:     $T_{edge} \Leftarrow T_{edge} \cup \{(r', r)\}$;
06:     $R \Leftarrow R \backslash \{r\}$;
07:     if ($r$ is a source node) then $S \Leftarrow S \cup \{r\}$;
08: endwhile;
09: return $(T_{node}, T_{edge})$;
```

B. Tree refinement

1) Protocol outline: Let us remind our mission here: improve tree stability ($stab_T$) in a decentralized manner, without violating degree constraints and the upper bound for maximum delay from sources ($depth_T$). For simplicity, let us suppose that there is only one source node positioned at the root of the tree. The most common decentralized way for improving $stab_T$ in this case is, to exchange nodes u and v on T whenever it yields a better $stab_T$. For instance, suppose that node u is the upstream neighbor of node v, $sc(u) < sc(v)$ and $d(u) = d(v)$. According to the definition of tree stability, swapping u and v improves $stab_T$. So it sounds like quite an easy task and various combinations of u and v like parent-child, grand parent-grand child, siblings and random combinations can be considered to reach the target. However, the above transformations cannot be simply performed because $depth_T$ may be violated. So a delay controlling mechanism should be associated with these node transformations. We propose a scheme for this decentralized controlling of $depth_T$, where each sub-tree has privilege to use only a *portion* of total *remaining maximum delay-play*, which is expressed by $D_{max} - depth_T$. And by assigning these portions in such a way where the summation of them is equal to $D_{max} - depth_T$, we can always transform nodes in each sub-tree without violating $depth_T$ restriction.

Our sub-trees for the above task are identified by making each sub-tree have maximum K hops. Basically, in each sub-tree, its root node u re-organizes the sub-tree to improve $stab_T$. This sub-tree is denoted by $RefT_{u,K}$. And this process is done in a bottom-up manner.

2) Protocol design: The refinement procedure consists of the *information collection* phase which gathers the information required for the refinement and the *node exchange* phase which takes place right after the information collection phase and executes the required node exchanges depending on the collected information.

a) Information collection: In order to refine the tree, all the nodes periodically collect the information required to execute refine procedure by message exchange along the current tree. The root node starts the information collection in the collection phase for every (regular) interval by broadcasting *synchronization messages* on the current tree. Obviously, the number of synchronization messages is $n - 1$ where n is the number of nodes on the current tree. When the root node (s_0) sends synchronization messages to its neighboring nodes, it assigns a node ID 0 to itself and also assigns node IDs 1,...,$d(s_0)$ to those neighboring nodes. The information included in this message are: (i) lists of node IDs on the paths from s_0 to other source nodes, (ii) delays to source nodes from s_0, (iii) the value of D_{max}, and (iv) the value of K. We assume the root node knows last two values, which are application specific parameters.

Similarly, if a node v receives a synchronization message from a neighboring node and if it knows that node ID n is assigned to itself, it assigns node IDs starting from $n \times d_{max} + 1$ up to $n \times d_{max} + d(v) - 1$ to the rest of its neighboring nodes when it sends messages to them (d_{max} denotes the maximum degree bound of all the nodes). Finally all the nodes in the tree have unique node IDs. Note that these IDs are used to identify parent-child relationship between neighboring nodes (a smaller ID indicates a parent) as well as to determine descendant node positions in node exchange phases. In addition to the information received from the upstream node, node v includes to the synchronization message for children; (v) $D(s_0, v)$, which is the delay from the root node, and (vi) $H(s_0, v)$, which is the hop count from the root node.

And also, using above (i), (ii) and (v), each node (say v) easily calculates the delay (denoted by $D(s_f, v)$) to the furthest source, s_f, which is used in the refine procedure.

b) Node exchange: A non-leaf node enters the collection phase if it has received a synchronization message from its parent and has sent synchronization messages to all its children. A leaf node does not send synchronization messages. Instead, when it receives a synchronization message, it enters the collection phase and replies a *collection message* to its parent. Each non-leaf node, except s_0, sends a collection message to its parent node whenever it receives collection messages from all the child nodes. The information included in the collection message from node u to its parent is described later in this section. Note that the number of collection messages required here is $(n - 1)$. So, totally, the number of messages required for the collection of the current status is only $2(n - 1)$, that is, only two messages are exchanged on each link of the tree.

Each node selects replace candidates among the child nodes[3]. Generally, node u selects its descending node w as u's replace candidate only if that replacement improves the tree stability of the subtree $RefT_{u,K}$. The tree stability improvement (denoted by $\Delta_{u,w}$) that can be achieved by replacing u with w is given by the followings.

$$\Delta_{u,w} = \begin{cases} \sum_{n=1}^{d(u)} \Big(sc(w) - sc(u)\Big) \cdot stab_{T_{x_n}} + \\ \sum_{n=d(u)+1}^{d(w)} \Big(sc(w) - sc(u) \cdot R \cdot sc(w)\Big) \cdot stab_{T_{y_n}} \\ \qquad\qquad (\text{if } d(w) \geq d(u),\ \text{Fig.2(b)}) \\ \sum_{n=1}^{d(w)} \Big(sc(w) - sc(u)\Big) \cdot stab_{T_{x_n}} + \\ \sum_{n=d(w)+1}^{d(u)} \Big(sc(u) \cdot R \cdot sc(w) - sc(w)\Big) \cdot stab_{T_{x_n}} \\ \qquad\qquad (\text{if } d(w) < d(u),\ \text{Fig. 2(c)}) \end{cases}$$

where $R = sc(v) \cdots sc(x_1)$ (v is the parent of w), and here T_{x_1} exceptionally stands for the sub-tree excluding the descending sub-trees $T_{y_1} \cdots T_{y_{d(u)}}$.

The next step is to find out which candidate results a replacement within $RefT_{u,K}$'s delay portion. We consider each node involved in node replacement and denote it as x. For instance, x stands for u, w_j (child nodes of u) and z (child nodes of w_j) in case of replacing u by w_j in Fig. 3(a). Then, the delay portion

[3]The scope of this paper is restricted to $K = 1$ case, where $K > 1$ cases will be some of our future work.

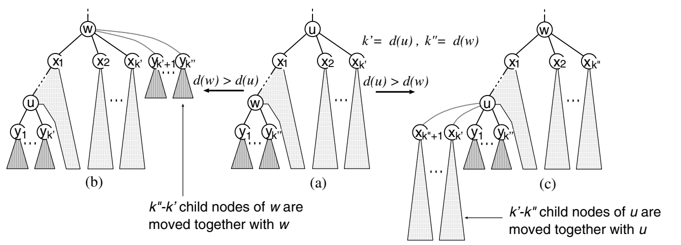
k′= d(u) , k″= d(w)
d(w) > d(u)
d(u) > d(w)
(b)
(a)
(c)
k″-k′ child nodes of w are moved together with w
k′-k″ child nodes of u are moved together with u

Fig. 2. Concept of replace candidate selection: (a) the initial tree (b) the tree after swapping u and w when $d(w) > d(u)$ (c) the tree after swapping u and w when $d(w) < d(u)$

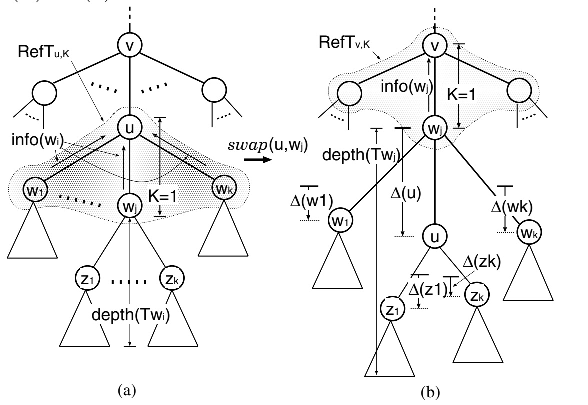

Fig. 3. refinement of $RefT_{u,K}$ (a) before refining (b) after refining

allowed for each x is calculated as follows.

$$\delta(x)_{max} = \{D_{max} - D(s_f, x) - depth(T_x)\}/H(s_f, x) \ (x \neq z)$$
$$\delta(z)_{max} = \delta(w_j)_{max}$$

Here, $depth(T_x)$ is the maximum delay in the sub-tree rooted at x and $H(s_f, x)$ is the hop count in the path from s_f to x. This equation assigns the remaining delay-play at node x equally among the nodes in the path from the furthest source node s_f.

Finally u selects the child w, which gives the maximum $\Delta_{u,w}$ and $\delta(x)$ is less than or equal to $\delta(x)_{max}$ for all involving x, where,

$$\delta(x) = D(s_f, x)_{new} - D(s_f, x)_{prev}$$
$$D(s_f, w_j)_{new} = D(s_f, u)_{prev} - D(v, u) + D(v, w_j)$$
$$D(s_f, u)_{new} = D(s_f, w_j)_{new} + D(w_j, u)$$
$$D(s_f, w)_{new} = \begin{cases} D(s_f, w_j)_{new} + D(w_j, w) & \text{if } d(u) \geq d(w_j) \\ \quad (w \text{ is not moved along with } u) & \\ D(s_f, u)_{new} + D(u, w) & \text{else} \\ \quad (w \text{ is moved along with } u) & \end{cases}$$
$$D(s_f, z)_{new} = \begin{cases} D(s_f, u)_{new} + D(u, z) & \text{if } d(w_j) \geq d(u) \\ \quad (z \text{ is not moved along with } w_j) & \\ D(s_f, w_j)_{new} + D(w_j, z) & \text{else} \\ \quad (z \text{ is moved along with } w_j) & \end{cases}$$

After replacement takes place, w_j sends its new parent v a new collection message with (i) $sc(w_j)$, (ii) $depth(T_{w_j})$ and (iii) $D(w_j, v)$. If no replacement takes place, u sends a similar collection message and refinement of $RefT_{v,K}$ is started when v receives collection messages from all its child nodes. This refinement procedure is propagated towards the top of the tree until the root node receives the collection messages from all of its child nodes.

c) Handling multiple sources: Though it is not clearly stated, we did not pay much attention to multiple sources in the tree refinement. Actually, it does not cause problems as long as the exchanging nodes do not exist on a path between two source nodes (in other words, no source node exists in the sub-trees routed at exchanging nodes). That is because node re-organization by such a sub-tree only makes sense to the nodes in that sub-tree in terms of tree stability or maximum delay from sources. However, when the refinement procedure progresses towards the top of the tree, the nodes which do not follow the above requirement come across, including source nodes. We call these nodes *irregular nodes*.

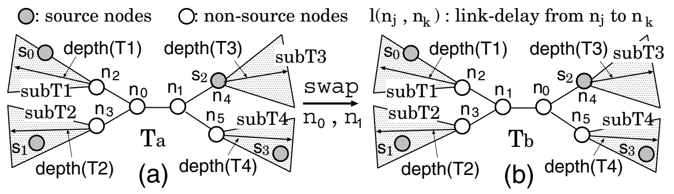

Fig. 4. Dealing with multiple sources: Exchanging nodes on paths between source nodes (a) before exchange (b)after exchange

Here we describe the way that such nodes are treated using the example in Fig. 4. Nodes n_0 and n_1 can be exchanged if tree stability of the whole tree is increased and maximum delays are not violated. Let us calculate the resulted tree stability gain, Δ_{n_0,n_1}.

$$\Delta_{n_0,n_1} = stab_{T_b} - stab_{T_a}$$
$$stab_{T_a} = \sum_{0 \leq k \leq 4} \sum_{r \in R} stab_{T_a}(s_k, r)$$
$$\sum_{r \in R} stab_{T_a}(s_0, r) = sc(s_0) \cdot sc(n_2) \cdot sc(n_0) \cdot (stab_{subT_2} + sc(n_1) \cdot (stab_{subT_3} + stab_{subT_4}))$$

$\sum_{r \in R} stab_{T_a}(s_i, r)$ $(i = 1, 2, 3)$ can also be expressed in the same way and then $stab_{T_a}$ is soon calculated. $stab_{T_b}$ can also be known by similar method and then Δ_{n_0,n_1} can be evaluated to validate exchanging n_0, n_1. Note that the maximum delay from sources of T_b is validated before the actual replacement takes place. This can be easily calculated by combining $depth(T_{subT_k})$ $(k = 0, 1, 2, 3)$ values and link delays around n_0 and n_1.

One important issue here is, it is impossible to cope with irregular nodes at different places simultaneously, as each of them is dominant on each other. Therefore, independent movements may not give the expected result or may violate the delay constraint. So when the refining process reaches an irregular node, it asks root node for the permission and root node permits once all regular node refining is done. And also the information about each irregular node is sent to root node, where each of them receives required data for the above calculations[4].

V. Experiments

A. Simulation Experiments

Extensive simulations have been conducted to evaluate how the nodes on the tree can receive multimedia streams without outages. The tree stability ($stab_T$) is the metric used for this purpose, because, higher the tree stability, lower the data outages occur from the upstream. Experiments were carried out with up to 1,000 overlay nodes using our GUI-assisted ALM protocol simulator, which is based on our middleware [17] for ALM protocols and will be open to the public soon.

[4]lock-free control mechanism is not much time consuming as the amount of irregular nodes is usually smaller

Here, we denote the setting of our experiments. The node-to-node delays on the overlay network (a full mesh) were generated from a standard distribution, where $\mu = 100[ms]$, $\sigma = 20[ms]$ (i.e. $N(100, 20^2)$). The degrees of nodes were set to follow a Pareto distribution [14] with parameters $a = 0.6$ and $b = 20$. We use Pareto distribution in GNU scientific library where the distribution function is defined as $p(x) = (a * b^a)/x^{(a+1)}$ $(x \geq b)$. The lifetimes of nodes were used as stability coefficients of nodes, and were set to follow a Pareto distribution with parameters $a = 1.2$ and $b = 1$.

number of nodes		100	200	500	1000
Initial tree	$depth_T$[ms]	258	279	289	306
	$stab_T$	0.37	0.27	0.18	0.13
D_{max}= 500$[ms]$	$depth_T$[ms]	480	472	481	463
	$stab_T$	0.46	0.46	0.38	0.34
	# of link restore	72	107	378	301
D_{max}= 1$[s]$	$depth_T$[ms]	751	879	901	934
	$stab_T$	0.55	0.53	0.48	0.43
	# of link restore	136	235	456	502
D_{max}= 1.5$[s]$	$depth_T$[ms]	1207	1324	1372	1402
	$stab_T$	0.62	0.62	0.58	0.52
	# of link restore	216	292	691	673

TABLE I
PERFORMANCE OF PROPOSED PROTOCOL FOR VARIOUS DELAY BOUNDS

[Effect of maximum delay bounds $(depth_T)$**]***:* First we have checked for the proposed protocol's behavior at various delay bounds with different numbers of nodes. Since our protocol improves $stab_T$ under the given delay bound, the improvement is small when the delay bound is very small. Table I shows this situation. When the delay bound becomes large, the tree is refined based on the given delay constraints and the value of $stab_T$ can be improved. The value of $stab_T$ is degrading gradually with increase of the number of nodes since in general a bigger tree is required for a bigger delay constraint in order to maximize $stab_T$.

	centralized		decentralized		
	$stab_T$	$depth_T$ [ms]	$stab_T$	$depth_T$ [ms]	# of link restore
proposed	0.82	460	0.64	1238	843
$s \cdot d$-first	0.80	536	0.62	1165	818
s-first	0.78	418	0.61	1330	827

TABLE II
SWAPPING POLICY COMPARISON

[Effect of node swapping policies]*:* We have compared the performance of our node swapping policy with $s \cdot d$-first and s-first based swapping, where nodes with higher $s \cdot d$ product and higher s are moved upwards. Here, basically a node u is replaced with its child v if it improves the value of $stab_{T_u}$ where T_u denotes the sub-tree rooted at u. This $s \cdot d$ product has been used in some of the previous lifetime aware schemes like [14] and [15]. Table II compares the performance of those methods. Here we set $depth_T$ to be large enough (=2[s]) to allow each method to work freely and also limited the number of source nodes to one as those existing schemes only consider a single source context. Our simulation results for the number of nodes = 500 show that the proposed method achieves better $stab_T$ for both the centralized and de-centralized schemes. For the centralized schemes, there exists no node swapping, but exists a policy for node selecting order. Therefore, Table II compares the centralized algorithm described in Section III-B2, with the $s \cdot d$ product and s prioritized methods. The results show that our protocol is better for both the centralized and de-centralized schemes.

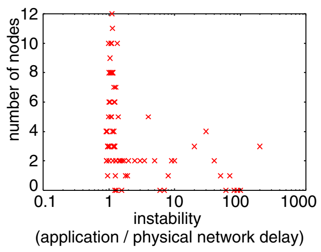

Fig. 5. PlanetLab instability distribution

[Effect of source node count]*:* Another interesting aspect to see is how our protocol works with different numbers of source nodes. $stab_T$, $depth_T$[ms] values for source node count 1, 2, 3, 4 and 5 were (0.65,1427), (0.63,1398), (0.62, 1390), (0.58, 1452) and (0.56,1421) respectively. Here, D_{max} and the number of nodes were set to 1.5[s] and 500 respectively. This results state that the performance degrades slightly when the number of sources increases. This is because the nodes are more restricted by multiple delay bounds to move when there are more sender nodes.

B. PlanetLab experiments

We have evaluated the performance of the proposed protocol through experiments on PlanetLab. The experiments were carried out using our middleware [17]. This middleware supports implementation and evaluation of various kinds of ALM protocols. We have implemented our protocol and two greedy algorithms that construct a minimum delay tree and a maximum bandwidth utilization tree respectively with greedy schemes. These greedy algorithms selects current tree T's next adding node u from the remaining nodes $R - T$ in such a way, where $T + u$ gives the minimum depth (or maximum bandwidth utilization) among all nodes in R. We have compared the performance of multimedia streaming delivery among these three protocols.

1) Experimental environment: First, we describe our experimental environment. PlanetLab nodes are located on all over the world and we have randomly selected those nodes. The number of terminals is 320. The terminal configuration is combination from Pentium 3 (1.2GHz) to Pentium 4 (3.4GHz), and the memory sizes vary from 512MB to 3.6GB. They use Linux OS version 2.6.12-1.1398_FC4.5.planetlab and JRE1.6 (Java version).

In order to determine unstable PlanetLab nodes, we have compared physical network delay and application level network delay. "ping" command can provide measures only for physical network delay. However, application level packet forwarding includes not only physical network delay but also delay caused by node state, packet generation, queuing and so on. Here, we define application level RTT(round trip time)/operating system level RTT of each PlanetLab node as their "instability". The distribution of instability of each PlanetLab node is shown in Fig. 5. It shows some nodes have much higher instability than the others in real environments. We consider the nodes with more than 1.5 instability are unstable.

2) Scenario of experiments: Jitter and bandwidth were selected as the streaming evaluation metrics. Two scenarios were used to compare the performance of our protocol and the greedy algorithms. We have evaluated the average jitter and bandwidth for (1) topology size and (2) the ratio of unstable nodes, respectively.

The scenario of the experiment (1) is as follows. First, 20 nodes joined the application forming the initial topology. Then streaming

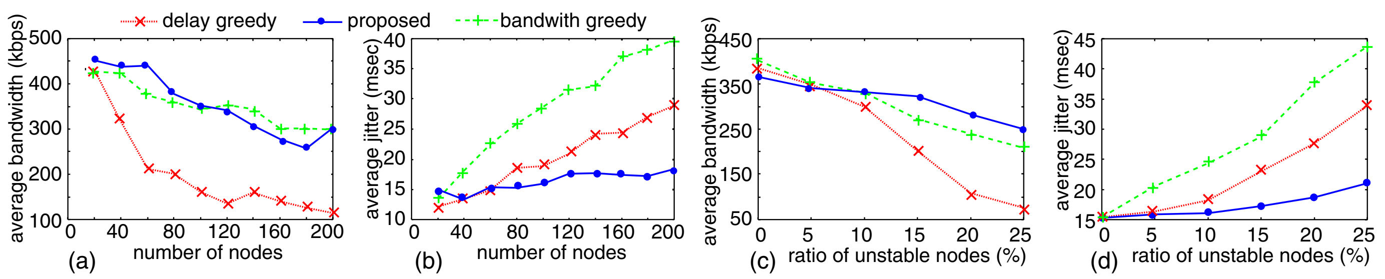

Fig. 6. PlanetLab experiment results (a):average bandwidth vs number of nodes, (b):average jitter vs number of nodes, (c):average bandwidth vs ratio of instable nodes, (d):average jitter vs ratio of instable nodes

was started after assigning a "source node" as the streaming-source. Here, the node joined first was considered as the source node and the streaming-rate was set to 500kbps, where we assume a video streaming application. Note that we selected the same node as the source node in each protocol. Next, another 20 nodes were added and the same experiment was carried out, and this was repeated until the number of nodes reached 200.

In the experiment (2), 200 nodes were selected and joined the session. The ratio of unstable nodes is changed from 0% to 25% at 5% intervals. The topology is constructed according to each protocol and the streaming-rate is also set to 500kbps. The quality of the streaming received at PlanetLab nodes can be confirmed on our web site [18].

3) Experimental results: Fig. 6-(a) and Fig. 6-(b) show the average bandwidth and jitter of each protocol over the number of nodes, respectively. The average bandwidth utilization is decreased with the increasing number of nodes on the delay greedy algorithm, which causes lower streaming qualities. The proposed protocol achieves bandwidth utilization close to that of bandwidth greedy algorithm, despite the latter is greedily optimized for bandwidth utilization. The average jitter of the receiving stream is lower on the proposed protocol compared to the other greedy algorithms. In particular, the bandwidth greedy algorithm that considers neither the delay nor the instability of nodes has higher jitter. On the bandwidth greedy scheme, jitter is increased by the unstable nodes reside close to the source node and this lowers the bandwidth utilization. In contrast, such unstable nodes are located as lower as possible in the tree and the quality of the streaming is maintained higher. Fig. 6-(c) and Fig. 6-(d) also show that the quality of the streaming is highly influenced by the unstable nodes in the greedy algorithms.

These real environmental experiments show that our protocol provides a better quality of streaming and scalability than delay based and bandwidth utilization based greedy algorithms by taking the instability of nodes into account.

VI. Concluding Remarks

In this paper, we have proposed a decentralized overlay multicast protocol for interactive multimedia applications including media streaming. The protocol constructs a spanning tree where the receive path stability of the entire tree is maximized while satisfying the delay-from-source constraint and degree constraint for each node. This can minimize the negative impact of end-hosts' unexpected leaves. The protocol is designed to cope with *multiple* senders. Simulation experiments show that it can improve the total receive path stability under given delay and degree constraints, and that the more delay constraints are relaxed, the more the total receive path stability is improved. Our PlanetLab experiments show that the proposed protocol have outperformed greedy algorithms in terms of bit-rate and jitter. The media files received in both cases are available from our web site http://www-higashi.ist.osaka-u.ac.jp/software/stable_multicast.html.

References

[1] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proc. of the 4th ACM SIGCOMM conference on Internet measurement*, 2004, pp. 41–54.
[2] S. Shi and J. Turner, "Routing in overlay multicast networks," in *Proc. of IEEE INFOCOM 2002*, 2002.
[3] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," in *Proc. of IEEE INFOCOM 2003*, 2003.
[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proc. of 3rd Usenix Symp. on Internet Technologies and Systems*, 2001.
[5] V. Roca and A. El-Sayed, "A host-based multicast (HBM) solution for group communications," in *Proc. of 1st IEEE Int. Conf. on Networking (ICN'01)*, 2001.
[6] P. Francis, "Yoid: Extending the internet multicast architecture," 2002, http://www.isi.edu/div7/yoid/.
[7] M. Yang and Z. Fei, "A proactive approach to reconstructing overlay multicast trees," in *Proc. of IEEE INFOCOM 2004*, 2004, pp. 2743–2753.
[8] H. Yamaguchi, A. Hiromori, T. Higashino, and K. Taniguchi, "An autonomous and decentralized protocol for delay sensitive overlay multicast tree," in *Proc. of 24th IEEE Int. Conf. on Distributed Computing Systems (ICDCS2004)*, 2004, pp. 662–669.
[9] T. M. Baduge, A. Hiromori, H. Yamaguchi, and T. Higashino, "Design and implemetation of overlay multicast protocol for multimedia streaming," in *Proc. of 34th IEEE Int. Conf. on Parallel Processing (ICPP2005)*, 2005, pp. 41–48.
[10] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. of ACM SIGCOMM 2002*, 2002, pp. 205–217.
[11] Y. H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Proc. of ACM SIGMETRICS*, 2000, tools are provided: http://www-2.cs.cmu.edu/ streaming/index.html.
[12] Y. Nakamura, H. Yamaguchi, A. Hiromori, K. Yasumoto, T. Higashino, and K. Taniguchi, "On designing end-user multicast for multiple video sources," in *Proc. of 2003 IEEE Int. Conf. on Multimedia and Expo (ICME2003)*, 2003, pp. III497–500.
[13] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. of ACM SIGCOMM 2004*, 2004, pp. 107–120.
[14] M. Bishop, S. Rao, and K. Sripanidkulchai, "Considering priority in overlay multicast protocols under heterogeneous environments," in *Proc. of IEEE INFOCOM 2006*, 2006, pp. 1–13.
[15] G. Tan and S. A. Jarvis, "Improving the fault resilience of overlay multicast for media streaming," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 721–734, Jun. 2007.
[16] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*, 3rd ed. W.H. Freeman & Company, 1979.
[17] K. Ikeda, T. M. Baduge, T. Umedu, H. Yamaguchi, and T. Higashino, "A middleware for implementation and evaluation of application layer multicast protocols in real environments," in *Proc. of the 17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV 2007)*, 2007, pp. 125–130, tools are provided: http://www-higashi.ist.osaka-u.ac.jp/software/ALM/middlewareAPI/.
[18] T. M. Baduge, K. Ikeda, H. Yamaguchi, and T. Higashino, "Our web site," http://www-higashi.ist.osaka-u.ac.jp/software/stable_multicast.html.