# An Autonomous and Decentralized Protocol for Delay Sensitive Overlay Multicast Tree

Hirozumi Yamaguchi Akihito Hiromori Teruo Higashino Kenichi Taniguchi
Graduate School of Information Science and Technology
Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, JAPAN
{h-yamagu, hiromori, higashino, taniguchi}@ist.osaka-u.ac.jp

## Abstract

*In this paper, we present a protocol for dynamically maintaining a degree-bounded delay sensitive spanning tree in a decentralized way on overlay networks. The protocol aims at repairing the spanning tree autonomously even if multiple nodes' leave operations or failures (disappearances) occur simultaneously or continuously in a specified period. It also aims at maintaining the diameter (maximum delay) of the tree as small as possible. The simulation results using ns-2 have shown that the protocol could keep reasonable diameters compared with the existing centralized static algorithm even if many nodes' participations and disappearances occur frequently.*

## 1 Introduction

Recent innovation of the Internet has brought us several *interactive group applications* such as multi-user video-chatting/conference systems. Since those applications require broadcast communication from every participant to the others, some multicast infrastructure among those participants is desired. For such a purpose, peer-to-peer multicast (referred to as overlay multicast) is a reasonable solution where a spanning tree involving all active participants (*i.e.* peers, referred to as *nodes* in this paper) of the application is constructed by connecting those nodes via unicast channels.

Such an interactive group application is generally delay-sensitive. Therefore, minimizing the *diameter* (maximum delay) on an overlay spanning tree is an important issue. Also in overlay multicast, it is important to consider bandwidth constraints around nodes. Since overlay links through a node actually use the same network interface of the node, the traffic amount of the node depends on its *degree* (the number of links of the node on the tree). So the degree should not exceed the capability limitation of the node.

In this paper, we present a protocol called MODE (Minimum-delay Overlay tree construction by DEcentralized operation). MODE constructs a *Degree-Bounded Minimum Diameter Tree (DBMDT)*. The construction problem of DBMDT is known as NP-hard [1]. Therefore, Ref. [1] proposes a heuristic algorithm called Compact Tree (CT) algorithm which is similar with Prim's minimum spanning tree algorithm [2]. Since CT algorithm focuses on static and centralized construction of a DBMDT, it is not suitable to directly adopt it to the DBMDT problem with nodes' join or leave operations (or failures) during a session. When some nodes leave a session, such an algorithm may require many modifications of the existing tree as well as much computation time in order to obtain a new tree with a smaller diameter. This may largely affect continuity of the session.

On the other hand, MODE aims at repairing the existing spanning tree in a simple, fast and decentralized way when at most $k$ nodes' simultaneous or continuous disappearances occur ($k$ is a protocol parameter). It also aims at shortening the diameter of the repaired tree. Our experimental results using ns-2 have shown that MODE could achieve similar diameters with CT algorithm in small computation time and small amount of control traffic even though MODE is autonomous and decentralized one.

This paper is organized as follows. Section 2 formulates the DBMDT problem. In Section 3 and Section 4, our MODE protocol is presented. Section 6 shows the experimental results and Section 7 summarizes the related work. Section 8 concludes the paper.

## 2 Preliminaries

### 2.1 DBMDT Problem

Hereafter, we call the participant nodes of an overlay tree simply *nodes*.

The definition of a Degree-Bounded Minimum Diameter Tree (DBMDT) is given below. Let $G = (V, E)$ denote a given undirected complete graph where $V$ denotes a set of nodes and $E$ denotes a set of potential overlay links which

are unicast connections between nodes. Also let $d_{max}(v)$ denote a degree bound (the maximum number of overlay links) of each node $v \in V$, and let $c(i,j)$ denote the cost (delay in this paper) of each overlay link $(i,j) \in E$. DBMDT is a spanning tree $T$ of $G$ where the diameter of $T$ (the maximum cost of the paths on $T$) is minimum and the degree of each node $v \in V$ (denoted as $d(v)$) does not exceed $d_{max}(v)$. Hereafter, $d_{max}$ is used to represent the maximum degree bound of all the nodes (*i.e.* $\max_{v \in V}\{d_{max}(v)\}$).

### 2.2 MODE Protocol Overview

MODE provides a decentralized heuristic algorithm that constructs a DBMDT, under nodes' participations and disappearances. In MODE, two logical phases called a *collection phase* and a *normal phase* are repeated alternately. The period of the collection phase is very short (*e.g.* less than two seconds in our experiments in Section 6) while that of the normal phase is relatively long (*e.g.* one minute).

In a normal phase, MODE copes with nodes' two kinds of operations, (i) join and (ii) leave/failure (referred to as *disappearance)*, in a decentralized way. We consider nodes' disappearances as a good occasion to shorten the diameter of the current tree. Therefore, whenever a disappearance happens, the "*center nodes*" of the isolated sub-trees are re-connected. Here, the center node of a tree is one which is the closest to the center of the diameter path. Also, the repair procedure should be done quickly enough to prevent data delivery from being suspended for a long time. For such a purpose, each node $u$ collects in the precedent collection phase the information of sub-trees (such as their center nodes) which will be isolated if a neighboring node $v$ disappears. This collection is done in a recursive (incremental) way. The collected sub-tree information is used for quick and efficient execution of repair procedures in the normal phase. Also, we have made MODE tolerant to at most $k$ nodes' disappearances in a normal phase.

### 2.3 Generic Assumptions

Nodes may disappear from a tree at any timing. In order to discuss the consistency of our protocol, we give the following assumptions concerned with the disappearances of nodes.

G1. Any node's disappearance does not affect the physical (underlying) network, and each node can immediately detect its neighboring node's disappearance.

G2. The initial node never disappears and each new node which wants to join a tree knows the network address (*e.g.* IP address) of this node.

This does not mean that the initial node plays a role of a centralized node. It only works as a well-known node required for new nodes' participations.

G3. A node's disappearance never occurs that loses any control message.

## 3 Collection Phase Protocol

For simplicity of discussion, we explain our collection phase protocol without assuming any disappearance during collection phases. To validate the protocol under some disappearances, see Section 5.1.

### 3.1 Initiating Collection Phase

As we mentioned in the previous section, we assume that the initial node never disappears (see assumption G2). This node is called the *root node*. The root node starts a collection phase for every (regular) interval by broadcasting *synchronization messages* on the current tree.

Recall that $d_{max}$ denotes the maximum degree bound of all the nodes. When the root node (say node $a$) sends synchronization messages to its neighboring nodes, it assigns a node ID 0 to itself and also assigns node IDs 1,...,$d(a)$ to those neighboring nodes. 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. 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 deliver control messages on trees in normal phases.

### 3.2 Sub-tree Information

For a pair of two adjacent nodes $u$ and $v$, let $T_{u,v}$ denote the sub-tree rooted at node $v$ and isolated by node $u$'s disappearance.

Each neighboring node $v$ of node $u$ collects the information of all the sub-trees which will be isolated by node $u$'s future disappearance. The sub-tree information of $T_{u,v}$ includes the followings.

- $v$'s network address (*e.g.* IP address) and node ID
- $dia(T_{u,v})$ : the diameter of $T_{u,v}$
- $diaNode(T_{u,v})$ : a set of $(k+1)$ nodes which are the candidates of the center node[1]. For each node $z \in diaNode(T_{u,v})$, its residual degree $d_{res}(z) = d_{max}(z) - d(z)$, network address and node ID are also included.

### 3.3 Collecting Sub-tree Information

A node enters a collection phase if it has received a synchronization message from its parent and sent synchronization messages to all its children.

A leaf node does not send synchronization messages. Instead, when it receives a synchronization message, it enters

[1]Having $k+1$ candidates in each sub-tree means that there exists at least one node in $diaNode(T_{u,v})$ which is alive even if $k$ nodes disappear.

a collection phase and sends a *collection message* to its parent. Each non-leaf node in a collection phase acts as follows. Whenever it receives collection messages from all the neighboring nodes except one neighboring node (say $x$), it sends a collection message to node $x$. Each node $u$ goes into a normal phase if, for each its neighboring node $v$, $u$ has received a collection message from $v$ and has sent a collection message to $v$. This means that in a collection phase, $2(n-1)$ collection messages are exchanged on the tree. We will explain how information collection is done by those collection messages.

Each node $v$ is responsible for calculating the sub-tree information of $T_{u,v}$ and includes the information into the collection message from node $v$ to node $u$. Due to the recursive definition of sub-tree information presented in this section, node $v$ can calculate the sub-tree information of $T_{u,v}$ if it receives collection messages from all the neighboring nodes (say $w_1, ..., w_{d(v)-1}$) except $u$. For the recursive definition, we introduce the following auxiliary parameters for each sub-tree $T_{u,v}$.

- $depth(T_{u,v})$ : the maximum delay of $T_{u,v}$ from $v$.
- $depthNode(T_{u,v})$ : the node list of the maximum delay path of $T_{u,v}$ from $v$. For each node $z \in depthNode(T_{u,v})$, its residual degree $d_{res}(z) = d_{max} - d(z)$, network address and node ID are also included.

We first consider to define them in a recursive way. Here "@" denotes the concatenation of node lists.

$$depth(T_{u,v}) = \max_{1 \le j \le d(v)-1} \{depth(T_{v,w_j}) + c(v, w_j)\}$$

For $w_j$ which maximizes the above,

$$depthNode(T_{u,v}) = [v]@depthNode(T_{v,w_j})$$

$dia(T_{u,v})$ can also be defined in a recursive way.

$$dia(T_{u,v}) = \max_{1 \le j \le d(v)-1} \{dia(T_{v,w_j}), jointdepth\}$$

where

$$\begin{aligned} jointdepth \quad = \quad & \max_{1 \le x,y \le d(v)-1} \{depth(T_{v,w_x}) + c(v, w_x) \\ & + depth(T_{v,w_y}) + c(v, w_y)\} \end{aligned}$$

Here, $w_x$ and $w_y$ denote two different nodes in $w_1, ..., w_{d(v)-1}$. The diameter of $T_{u,v}$ is the maximum value of (i) the diameters of its sub-trees and (ii) the sum of the two longest depths from node $v$.

Finally, $diaNode(T_{u,v})$ can be defined as follows.

$$diaNode(T_{u,v}) = \begin{cases} diaNode(T_{v,w_j}) \ \ (\text{if } dia(T_{u,v}) = dia(T_{v,w_j})) \\ (k+1) \text{ nodes near center node on "} jointpathlist\text{"} \\ \qquad (\text{if } dia(T_{v,w}) = jointdepth) \end{cases}$$

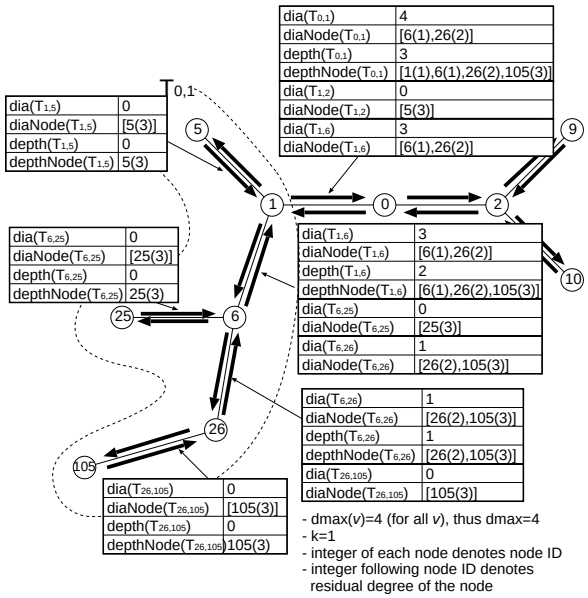


**Figure 1. Sub-tree Information Collection**

where

$$\begin{aligned} & jointpathlist \\ & = \text{reverse}(depthNode(T_{v,w_x}))@[v]@depthNode(T_{v,w_y}) \end{aligned}$$

and "reverse" is the reverse function of a list[2]. If the diameter of $T_{u,v}$ is the same as that of a sub-tree $T_{v,w_j}$, the corresponding $diaNode(T_{u,v})$ is the same as that for $T_{v,w_j}$. On the other hand, if the concatenation of two maximum delay paths from $v$ becomes $T_{u,v}$'s diameter path, we select $(k+1)$ candidate nodes from the nodes on the new diameter path.

The collection message sent from $v$ to $u$ includes:

- the sub-tree information of $T_{u,v}$,
- $depth(T_{u,v})$ and $depthNode(T_{u,v})$ and
- the sub-tree information of $T_{v,w}$ for each $w$ except $u$.

Therefore, after node $u$ receives/sends collection messages from/to all its neighboring nodes, node $u$ knows each sub-tree information of $T_{v,w}$ where $v$ is a neighbor of node $u$ and $w$ is a neighboring of node $v$. Then it enters a normal phase.

### 3.4 Example

Fig. 1 shows how collection messages are exchanged. Initially, nodes 5, 25 and 105 send the information (and auxiliary parameters) of the sub-trees $T_{1,5}$, $T_{6,25}$ and $T_{26,105}$ respectively, since they are the leaf nodes (here we omit the case for leaf nodes 9 and 10). Node 26 receives the collection message from node 105, calculates the sub-tree information (and auxiliary parameters) of $T_{6,26}$ and sends it to node 6, together with

[2]If $diaNode(T_{u,v})$ has less than $k+1$ nodes, node $v$ selects the rest of nodes from $diaNode(T_{v,w_z})$ as many as possible until $diaNode(T_{u,v})$ has $k+1$ nodes or no node is left in any $diaNode(T_{v,w_z})$.

the sub-tree information of $T_{26,105}$. If node 6 receives collection messages from nodes 25 and 26, it calculates the sub-tree information (and auxiliary parameters) of $T_{1,6}$ and sends it to node 1 together with $T_{6,25}$ and $T_{6,26}$'s sub-tree information. If node 1 receives the collection messages from nodes 5 and 6, it calculates the sub-tree information (and auxiliary parameters) of $T_{0,1}$ and send it to node 0.

Similarly, node 1 receives the collection message from node 0. If it receives the message, it calculates the sub-tree information of $T_{6,1}$ and $T_{5,1}$ and sends them to nodes 6 and 5, respectively. Since node 1 knows all the three sub-tree information of $T_{6,25}$, $T_{6,26}$ and $T_{6,1}$, it knows the required information for the repair procedure of node 6's disappearance. Note that by exchanging those information, nodes 25 and 26 can also know the same information.

# 4 Normal Phase Protocol

This section presents DBMDT construction protocol which consists of two procedures to process join and disappearance of nodes in normal phases. We guarantee that a spanning tree is maintained under at most $k$ nodes' disappearances. We make the following two assumptions concerned with normal phases to make discussion simple. Later we try to relax these two assumptions (see Section 5.2).

N1. For each (non-leaf) node $v$ that disappears, there exists at least one neighboring node which does not disappear during the repair procedure of $v$'s disappearance.

N2. Once a node is selected as a repair master or a candidate node to connect to the repair master, it does not disappear until it completes its task.

## 4.1 Join Procedure

The join procedure is simple. A new node which wants to join the current tree $T$ first asks the root node the possibility to connect to it[3]. If it is possible, the new node connects to the root node. If there is no residual degree or if the delay between those nodes is greater than a certain threshold, the root node lets the new node know the network address of a (randomly selected) neighboring node. Then the new node asks the neighboring node the possibility by the same procedure. By iterating this procedure, this node can join the current tree.

The reason to take this approach is that we would like to keep the procedure as simple as possible for fast processing of join requests. Later we show that this join process does not enlarge the diameters of trees, and is processed quickly enough.

[3]This is because we assume that a new node only knows the root node as a well-known node (see assumption G2). To avoid access concentration, several well-known nodes may be assumed rather than a single node.

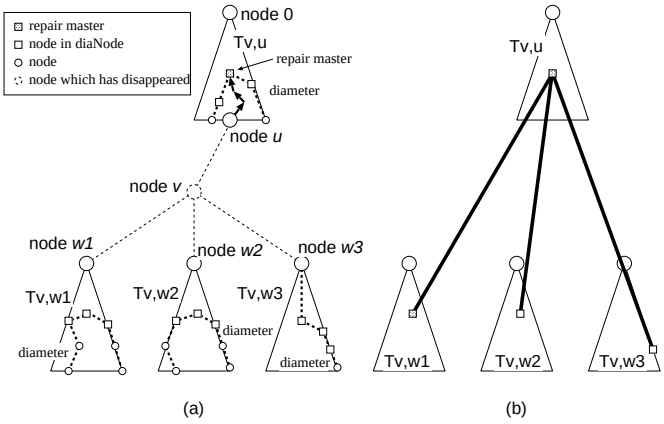


**Figure 2. Repair Procedure (single disappearance)**

## 4.2 Repair Procedure for Single Disappearance

Let $v$ denote a node which has disappeared, $u$ denote $v$'s parent and $w_j$ denote $v$'s $j$-th child ($1 \leq j \leq d(v) - 1$) where $d(v)$ is the current degree of node $v$. Also, let $R(v)$ denote the repair procedure for node $v$'s disappearance. We first explain $R(v)$ without considering other disappearances, and then consider them in the next section.

We illustrate the behavior of the repair procedure for a single (non-leaf) disappearance in Fig. 2. For node $v$'s disappearance, its parent (node $u$) activates the repair procedure and selects a node from $diaNode(T_{v,u})$ which is the closest to the center node and has at least one residual degree. This node is called the *repair master*. Node $u$ sends the information of sub-trees $T_{v,u}$ and $T_{v,w_j}$ ($1 \leq j \leq d(v) - 1$) to the repair master (see Fig. 2(a)). Using the node ID of the repair master, this delivery is done on the tree according to the algorithmic routing in Ref. [9]. For each sub-tree $T_{v,w_j}$ ($1 \leq j \leq d(v) - 1$), the repair master selects a node from $diaNode(T_{v,w_j})$ which also resides near the center node and which has at least one residual degree. Then the repair master establishes an overlay link to the node (Fig. 2(b)). Note that if the residual degree of the repair master is not enough to accommodate all the sub-trees, the repair master delegates the repair of some sub-trees to its neighboring nodes.

By this procedure, nodes near the "center" of the sub-trees are connected, and the diameter of the repaired tree is expected to be equal or smaller than before.

## 4.3 Repair Procedure for Multiple Disappearances

### 4.3.1 Additional Disappearance during Repair Procedure

We first consider how another node $v'$'s disappearance and its corresponding repair procedure $R(v')$ affect $R(v)$. Node $v'$ can be either one of the followings; (1) $v$'s parent, (2) a node on the route from the parent $u$ to repair master, including re-

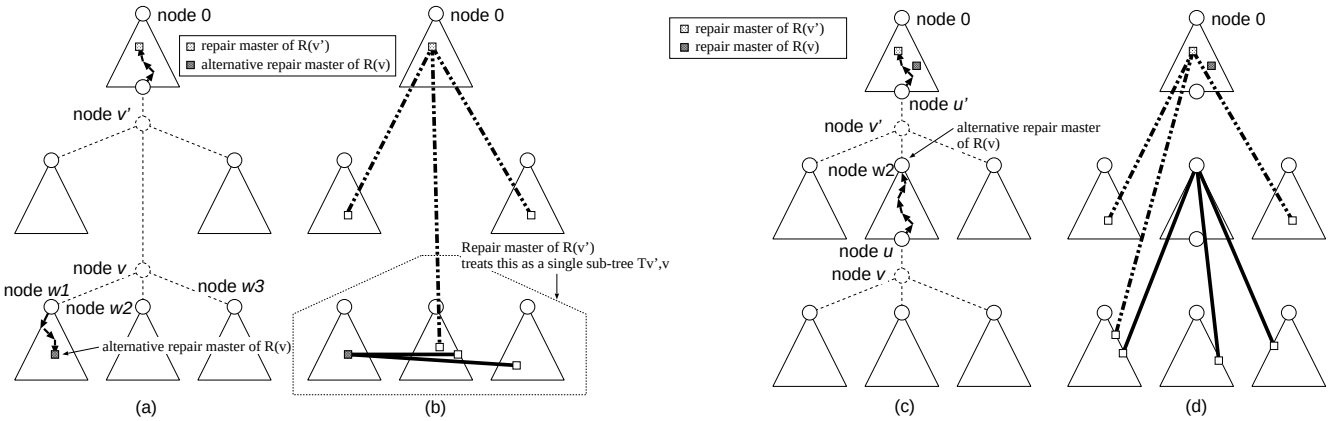


**Figure 3. Disappearance of Node v' during Repair Procedure R(v)**

pair master, (3) a center node of a sub-tree, and (4) none of the above nodes. For each case, we extend $R(v)$ to make it tolerant to the disappearance of $v'$. Note that if $v'$ disappears after $v'$ completes its role in $R(v)$ in cases (1), (2) and (3), or if $v'$ matches case (4), $R(v)$ and $R(v')$ can be regarded as two independent procedures. See Section 4.3.2 for those cases.

**[Case 1: node $v'$ is the parent of node $v$.]**

We make $R(v)$ tolerant to this case as follows. Assumption N1 (given at the beginning of this section) ensures that at least one child is being alive at that time. So whenever node $v$ disappears, we let each child of $v$ check whether the parent $u$ of $v$ is alive or not and also whether it is the eldest child (*i.e.* with the smallest ID) of the children of $v$ which are alive. Then the eldest child $w_x$ selects the *alternative repair master* from $diaNode(T_{v,w_x})$ (Fig. 3(a)). The alternative repair master works as a repair master and connects $T_{v,w_j}$ $(1 \leq j \leq d(v)-1)$ but leaves $T_{v,v'}$ unconnected (Fig. 3(b)). Since $R(v')$ and $R(v)$ are not concerned with each other, $R(v')$ considers that the sub-tree $T_{v',v}$ exists as it is. Therefore, $R(v)$'s alternative repair master is required to connect only $T_{v,w_j}$ $(1 \leq j \leq d(v)-1)$. □

The above extension also makes $R(v)$ tolerant to a sequence of disappearances of nodes $v_1$, $v_2$, ... and $v_h$ occurs where $v_{i-1}$ is the parent of $v_i$. For the disappearance of $v_i$, a node in the generation of $v_{i+1}$ is alive (assumption N1) and activates $R(v_i)$ on behalf of $v_{i-1}$. This means that the sub-trees rooted at the children of $v_i$ are connected. Since assumption G2 ensures that there finally exists the root node which is always alive, the parent of $v_1$ always exists. Consequently the sequence of disappearances can be repaired.

**[Case 2: node $v'$ is a node on the route from the parent $u$ of $v$ to the repair master, including the repair master.]**

In this case, we let the node just before $v'$ be the alternative repair master. For example, in Fig. 3(c), node $w'_2$, which is the node just before node $v'$, becomes the alternative repair master and the tree is repaired (Fig. 3(d)). □

**[Case 3: $v'$ is the center node of a sub-tree.]**

This is an easy case. Any $T_{v,w}$ has $k+1$ nodes in $diaNode(T_{v,w})$. Otherwise $T_{v,w}$ has only less than $k+1$ nodes. Therefore, we let the repair master of $R(v)$ select another node from $diaNode(T_{v,w})$. □

#### 4.3.2 Additional Disappearance after Repair Procedure

Then we consider the case that $R(v')$ occurs after $R(v)$.

After $R(v)$ is completed, the isolated sub-trees are connected using new overlay links. Thus $R(v')$ can repair the tree as in the case of a single disappearance. What we should consider here is that part of sub-tree information and node IDs does not match the current tree topology after $R(v)$. Note that for each sub-tree information which $R(v')$ may use is candidates of center nodes where at least one node is still alive under less than $k$ disappearances. Even though they may not be the "center" nodes now, there are still the candidates to recover the connectivity. On the other hand, inconsistent node IDs may lead to wrong routing in delivering a message to a repair master and wrong decision of initiator of $R(v')$, which affects the consistency of the tree.

To avoid such a problem, We let $R(v')$ never use new overlay links generated by $R(v)$ (*i.e*, we let $R(v')$ only use the original links which have existed since the beginning of the normal phase). Two nodes connected by an original link still keep node ID consistency, and if $R(v')$ faces missing node, this can be regarded as the same cases as in the previous section.

## 5 General Cases

In this section, we consider the cases where disappearances occur in collection phases and where disappearances in normal phases do not follow the assumptions given at the beginning of Section 4.

### 5.1 Disappearance in Collection Phase

In Section 3, we have assumed that no disappearance occurs in any collection phase. However, even though a collection phase takes relatively short time, a few nodes may disappear during the phase.

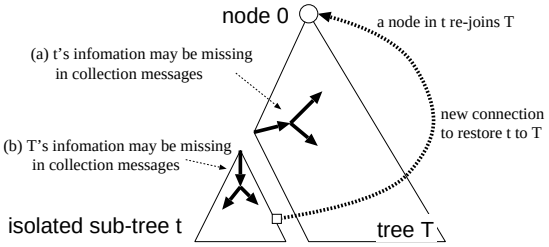


**Figure 4. Disappearance in Collection Phase**

In essential, any node's disappearance results in isolated sub-trees. We can consider two cases where (i) the nodes in an isolated sub-tree (say $t$) have already received synchronization messages and their node IDs have been refreshed and (ii) they have not received the synchronization messages yet. In both cases, we let exactly one of the nodes in $t$ re-join the tree $T$ (see Fig. 4) in the same way as we will explain in Section 4.1 in order to restore $t$ to the current tree $T$. Also we have to make sure that this isolation is in the collection phase since any isolation in a normal phase will be dissolved by the repair procedure.

In case (i), the nodes around the disappearances can determine that the disappearances were in the collection phase only if they had not completed all the collection messages' exchange yet at that time. Then the root node of $t$ (determined by the refreshed node IDs) re-joins the tree. On the other hand, in case (ii), the nodes in $t$ cannot know whether this isolation occurred in the collection phase or in the previous normal phase. Remember that the root node sends synchronization messages at regular intervals. We assume that the others know the intervals. If the isolation is dissolved by the repair procedure, the synchronization messages arrive before the regular interval has past. If not, each node in $t$ understands that it has been isolated in the collection phase. Then the nodes in $t$ execute a certain leader election in $t$ (it is really simple because $t$ is a spanning tree) and the selected node re-joins the tree $T$.

Note that for nodes' disappearances, we let new leaf nodes initiate collection messages if they have not sent it yet so that every node can know the end of the collection phase. However, some information may be missing from collection messages. Two cases are possible (see Fig. 4). In both cases, the nodes in $T$ and $t$ may not be able to recognize each other due to the missing information. However, this situation is the same as that in Section 4.3.2.

### 5.2 Relaxing Assumptions in Normal Phases

If a disappearance $v'$ does not follow either one of assumptions N1 and N2, some isolated sub-trees may exist in $R(v)$ since some nodes that should play a certain role in $R(v)$ have disappeared. In the collection phase, by using the synchronization message timeout as we discussed in Section 5.1, such isolation can be detected by the nodes in the sub-trees. However, in the normal phase, such isolation may be dissolved after the next collection phase and such a long isolation may be intolerable.

One possibility to avoid such a long isolation is that the root (initial) node sends probe messages in every short period. Then, the root nodes of the isolated sub-trees can recognize their isolation, and they can re-join the current tree by asking the root node. Some other solutions may be possible.

## 6 Performance Evaluation

### 6.1 Simulation Setup

We have implemented our MODE protocol on ns-2. In our experiments, networks with about 400 physical nodes have been generated and used as underlying networks. We have selected 200 nodes as overlay participant nodes. The end-to-end delay ranges from 80ms to 120ms (the average is 100ms).

Considering practical situations, we have prepared the following scenario that simulates a real-time session in collaborative applications such as a video-based meeting or groupware. Note that we set the interval between collection phases to 60 seconds. Also $k$ (the maximum number of nodes which are allowed to disappear in each normal phase) was set to 5 (2.5% of the entire nodes). The degree bound was set to 5 for all the nodes. The scenario is as follows. (i) The session period is 180 seconds. (ii) Each of 200 nodes joins the session only once and eventually leaves the session. (iii) Within the first 30 seconds, about 60 nodes join the session. (iv) From 30 seconds to 140 seconds, additional joins are processed. Also some existing nodes leave the session. There are two collection phases at 60 seconds and 120 seconds. (v) After 140 seconds, no node joins and about 40 nodes leave the session (consequently the number of disappearances in each normal phase greatly exceeds $k$).

In Fig. 5 and Fig. 6, we have shown the number of nodes on the tree at every second together with the numbers of join/leave operations to make it facilitate to see the dynamics of the metrics according to the scenario. Note that even though the number of disappearances in each normal phase greatly exceeds $k$ in the scenario, no island was created in MODE protocol in the experiments.

### 6.2 Implementation of Compact Tree Algorithm

As comparison, we have also implemented the centralized algorithm presented in Ref. [1] (called *CT algorithm*) on ns-2. The algorithm starts with the minimum delay (overlay) link as the initial tree, and adds the rest of links one by one so that we can minimize the diameter of the resulting tree using the entire tree knowledge.

Since CT algorithm can construct a spanning tree with a near optimal diameter value, we have used it as a benchmark to see the optimality of diameters in our MODE protocol. We have also used it to confirm the efficiency of MODE in term of the repair costs. For such a purpose, we adopt the following

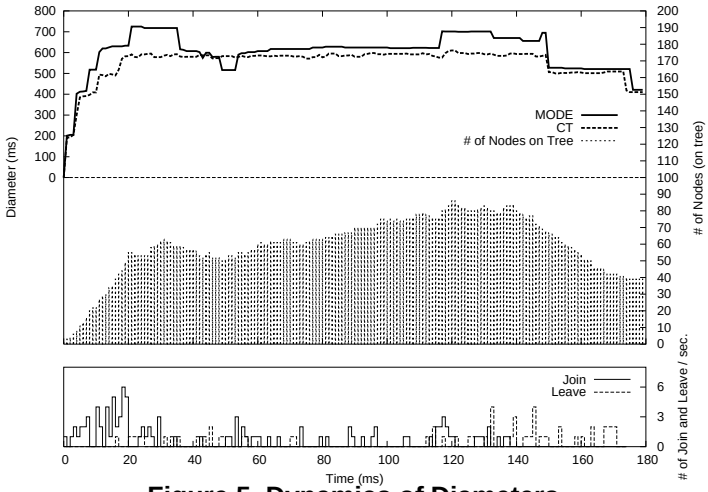


**Figure 5. Dynamics of Diameters**

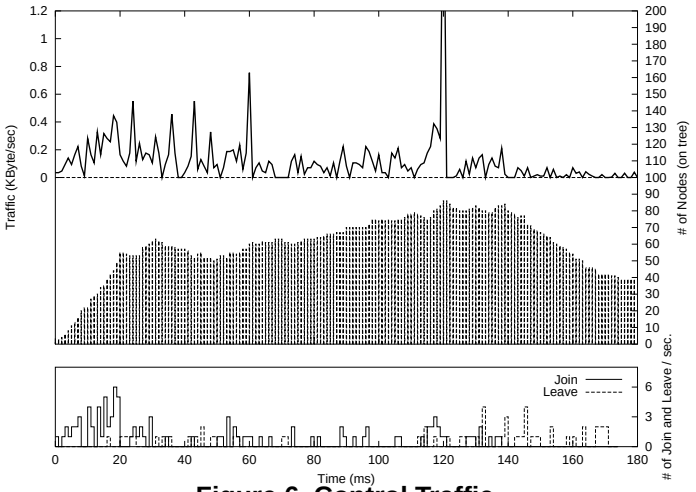


**Figure 6. Control Traffic**

implementation policy of CT algorithm to make it work with nodes' join/leave operations. (i) For a new node's join, the optimal point (node) that minimizes the diameter of the resulting tree is calculated using the entire tree knowledge. (ii) In case of a node's disappearance, the entire tree is scraped and built up again by CT algorithm so that we can obtain a near optimal solution.

### 6.3 Experimental Results

**[Diameter]** We have measured the diameters of the trees at every one second. The result is shown in Fig. 5.

We can see that MODE could archive reasonable diameters. The average diameter is only 1.09 times and in case of maximum difference the diameter is still 1.25 times as large as those in CT algorithm. In particular, throughout the session, MODE could follow the dynamics of diameters in CT algorithm. Considering the fact that CT algorithm optimized the diameter for every node's disappearance, we can say that MODE could archive almost the optimal diameters even though it is designed in an autonomous and decentralized protocol. Also we can see that as the number of node decreases, the diameter became small at the end of the session.

Note that if we use not a total delay but a hop count as a diameter, MODE could archive almost the same values as CT algorithm. The diameter difference between MODE and CT algorithm comes from the optimality of delays of links which constitute diameter paths. This means that the diameter difference may not become so large even in large-scale networks.

**[Repair Cost]** The repair cost in MODE for a repair procedure is the number of overlay links which are established and disconnected during the procedure. On the other hand, the cost in CT algorithm is the differential of two trees' overlay links before and after the disappearance. Since the repair procedure in CT algorithm is "scrap-and-build", the cost becomes large. Therefore it may seem unfair to compare it with MODE. However, we would like to see good balance of reasonable diameters and low repair costs (thus quickly repair is possible).

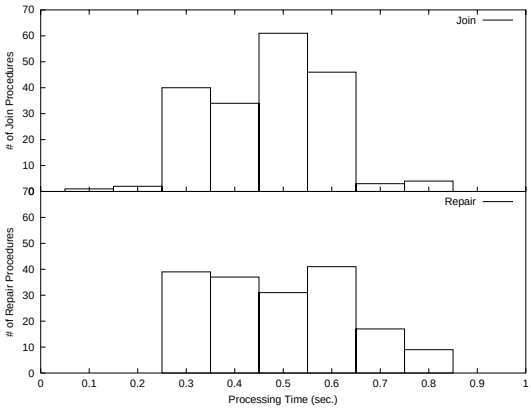


**Figure 7. Join/Repair Procedures' Processing Time Distribution**

For such a purpose, we have compared the following items.

| | MODE | CT |
|---|---|---|
| total repair cost (through the scenario) | 434 | 7283 |
| # of repair procedures applied | 149 | 149 |
| average repair cost | 2.9 | 48.9 |

Obviously, our protocol could archive low repair costs.

**[Traffic]** We have measured the control traffic amounts on the tree. The result is shown in Fig. 6.

Due to small message complexity, MODE only required about 1Kbytes/sec (8Kbps) in each collection phase (we can see the two peaks around 60sec. and 120sec.). Even for a series of join procedures, MODE needed small traffic amount (see between 0sec. to 60sec.). Totally the traffic amount in MODE protocol is very reasonable.

Note that in the underlying network, the traffic amount is expected to be at most $d_{max}$ (=5 in the experiments) times as large as that on overlay trees, because the link stress (the number of packet duplications on a physical link) around the end hosts is $d_{max}$ due to the nature of the overlay network. However, in our case, such a value is still reasonable enough.

**[Time for Procedure Execution]** Finally, we have measured the time required to execute the join/repair procedures. Their distributions are shown in Fig. 7.

We can see that almost all join/repair procedures only required at most 0.7 or 0.8 seconds. The expected average values of the join and repair procedures are 0.47 and 0.49 seconds, respectively. From the results, we can say that the procedures were processed quickly enough.

## 7 Related Work

Most researches of application layer multicast pursue the stability (*i.e.* fault-tolerance) and efficiency of overlay multicast trees under the constraints of bandwidth and delay. For example, HBM [3] mainly considers how to make backup links in a centralized way for the failure of a node. ALMI [4] proposes a centralized algorithm for constructing minimum spanning trees. Yoid [5] is similar to ALMI, however, Yoid together uses shared tree connection and mesh-like connection for robustness. Narada [6] considers mesh-like overlay networks where a shortest path tree per source is constructed. NICE [7] considers hierarchical topology where leaders organize their logical sub-domains.

Our MODE protocol is different from the above approaches in the following points. (1) *Autonomous and decentralized organization of trees.* Most protocols take centralized approaches, and decentralized approaches are sometimes considered to be ineffective because they increase protocol complexity. However, in our case, we show that decentralized information collection is very effective for our DBMDT problem with dynamic join/leave activities, because it requires very small amount of control traffic and simple operations. (2) *Tree optimization.* Most approaches mainly focus on their protocol frameworks and do not consider the optimization of trees. ALMI considers minimum spanning trees, but they are computed in a centralized way using the entire knowledge. (3) *Tolerance to multiple node failures.* We consider multiple nodes' disappearances in a distributed environment and validate the soundness.

Ref. [1] treats the DBMDT problem, however, as we stated in Section 1, only a centralized heuristic is presented. To our best knowledge, a recent paper, Ref. [8], only presents a distributed protocol called OMNI to construct DBMDT. However, the approach of OMNI is very different from our MODE protocol. OMNI iterates localized refinement operations based on SA, while our refinement is global and based on the collected diameter information. Due to the nature of design, OMNI seems to need less traffic than MODE but need more local operations. It is interesting to see difference between OMNI and MODE, and it is part of our ongoing work.

## 8 Concluding Remarks

In this paper, we have proposed an autonomous and decentralized protocol called MODE for dynamically constructing a degree-bounded delay sensitive multicast tree on overlay networks where simultaneous (continuous) join/leave activities of nodes is considered.

We have not mentioned yet the complexity of MODE. In a collection phase, $(n-1)$ synchronization messages and $2(n-1)$ collection messages are delivered on the tree. Then we look into the size of a collection message. The number of sub-trees whose information is contained in the message is at most $d_{max}$. The message also contains the depth information, whose size depends on $\log_{d_{max}} n$ in average and $n$ in the worst (but rare) case where $n$ denotes the number of nodes. Therefore, the message size complexity is $O(n + k * d_{max})$ in the worst case. We have shown that control traffic was about 8kbps in each collection phase under a hundred of nodes. Regarding time of a collection phase, broadcast of synchronization messages takes the time of the diameter and exchange of collection messages on a tree does also in the worst case, the total time required for a collection phase is at most twice of the time of the diameter. In our experiments, the diameter was less than 1 second and therefore the collection phase period was less than 2 seconds.

Implementing MODE in a real environment is part of our future work.

## References


[1] S. Shi, J. Turner and M. Waldvogel, "Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks," *Proc. of NOSSDAV'01*, pp. 83–91, 2001.

[2] R. L. Graham and P. Hell, "On the History of the Minimum Spanning Tree Problem," *IEEE Annals of the History of Computing*, Vol. 7, No. 1, pp. 43–57, 1985.

[3] V. Roca and A. El-Sayed, "A Host-Based Multicast (HBM) Solution for Group Communications," *Proc. of IEEE ICN'01)*, 2001.

[4] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," *Proc. of 3rd USENIX Symp. on Internet Technologies and Systems (USITS)*, pp. 49–60, 2001.

[5] P. Francis, "Yoid: Extending the Internet Multicast Architecture," *Unrefereed Report*, 2002. `http://www.isi.edu/div7/yoid/`

[6] Y. -H. Chu, S. G. Rao and H. Zhang, "A Case for End System Multicast," *Proc. of ACM SIGMETRICS*, pp. 1–12, 2000.

[7] S. Banerjee, B. Bhattacharjee and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. of ACM SIGCOMM 2002*, pp. 205–217, 2002.

[8] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications," *Proc. of IEEE INFOCOM 2003*, 2003.

[9] P. Huang and J. Heidemann, "Minimizing Routing State for Light-Weight Network Simulation," *Proc. of Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2001.