Decentralized Application Layer Multicast Protocols for Interactive Applications

January 2008

Thilmee Malinda BADUGE

Decentralized Application Layer Multicast Protocols for Interactive Applications

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2008

Thilmee Malinda BADUGE

List of Publications

Journal Papers

1. Thilmee M. Baduge, Akihito Hiromori, Takaaki Umedu, Hirozumi Yamaguchi and Teruo Higashino : "A Decentralized Protocol MODE for Minimum Delay Spanning Trees on Overlay Networks", *Journal of Information Processing Society of Japan (IPSJ)*, Vol. 46, No. 2, pp. 482-492 (February 2005) (in Japanese).

2. Thilmee M. Baduge, Akihito Hiromori, Hirozumi Yamaguchi and Teruo Higashino : "A Distributed Algorithm for Constructing Minimum Delay Spanning Trees Under Bandwidth Constraints on Overlay Networks", *Transactions on the Institute of Electronics, Information and Communication Engineers (IEICE)*, Vol. J88-D-I, No. 11 (November 2005), pp. 1648-1658 (in Japanese).

3. Thilmee M. Baduge, Hirozumi Yamaguchi and Teruo Higashino : "An Efficient Overlay Multicast Protocol for Heterogeneous Users", *Journal of Information Processing Society of Japan (IPSJ)*, Vol. 46, No. 11, pp. 2614-2622 (November 2005).

Conference Papers

1. Thilmee M. Baduge, Akihito Hiromori, Hirozumi Yamaguchi and Teruo Higashino : "Design and Implemetation of Overlay Multicast Protocol for Multimedia Streaming", *Proceedings of the 34th International Conference on Parallel Processing (ICPP 2005)*, pp. 41-48, Oslo, Norway (June 2005).

2. Thilmee Malinda Baduge, Hirozumi Yamaguchi and Teruo Higashino : "MODE for Mobile - An Efficient Overlay Multicast Protocol for Heterogeneous Users

-", *Proceedings of the 2nd International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2005)*, pp.96-101, Osaka, Japan (April 2005).

Other Related Conference Papers

1. Kazushi Ikeda, Thilmee M. Baduge, Takaaki Umedu, Hirozumi Yamaguchi and Teruo Higashino : "A Middleware for Implementation and Evaluation of Application Layer Multicast Protocols in Real Environments", *Proceedings of the 17th International workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV 2007)*, pp. 125–130, Illinois, USA (June 2007).

Abstract

Recent innovation of the Internet has brought us several *interactive group applications*. Especially, recent applications may require multimedia-based group communication such as multimedia streaming or video conferencing. It is a common consensus that we need a multicast solution for the group communication and IP multicast is not suitable for such a purpose because of its deployment limitations. Instead, *application layer multicast* (*ALM* in short) solutions have had a lot of attentions where application nodes (end-hosts) are connected by unicast channels and consequently form tree-like virtual networks (overlay networks) among them.

Although, all these group applications are similar in the sense of multicasting, they have different *QoS* (*Quality of Service*) requirements. By looking at the real world group applications, we can say that most of them involve interactivity, use some kind of multimedia data and have different community scales. Therefore, it is very important for an underlying ALM scheme to be (a) latency sensitive, (b) bandwidth sensitive and (c) scalable. In this dissertation, first, we address these issues by proposing an ALM scheme called *MODE*, which constructs a *Degree Bounded Minimum Diameter Tree* (*DBMDT*) in a de-centralized manner. The overlay tree constructed by this scheme satisfies the following conditions; (i) the maximum delay between any pair of nodes is minimized, (ii) the number of overlay links connected to each node (degree) is restricted by its bandwidth availability and (iii) scalable with its de-centralized design.

In addition to the above major QoS requirements, some applications require the consideration of the existence of multiple sources. For example, in a video-conference or a panel discussion, the pictures of some primary persons should be continuously delivered to the other audience. Although these sources are some times subject to change, but are not changed so frequently. For this kind of purpose, we propose a method for the de-centralized construction of spanning trees as the second contribution of this dissertation, where the maximum delay *from those senders* is minimized. This scheme,

called *STS*, constructs a *sender-dependant Degree Bounded Minimum Diameter Tree* (*s-DBMDT*). Though some basic ideas have been taken over from the previous work MODE, which basically minimizes the maximum delay between any pair of nodes, STS differs from MODE from the following aspects: (i) formulation of a new problem well suits for interactive mulimedia streaming applications which associates multiple sender nodes and (ii) design and implementation of an adaptation mechanism needed for mulitimedia streaming.

The end-user heterogeneity is also an important issue to be addressed. With the recent rapid deployment of high-spec mobile devices, the end-users are shifting from the conventional desktop PCs to various hand-held devices, such as mobile phones or PDAs. So the ALM schemes we design should be ready to adapt these mobile terminals as well. In this case the following problems come across; the mobile hosts may not be stable due to the limitation of batteries, computing and communication capabilities and so on, and thus not stable to relay data packets frequently. Moreover, they may use wireless links and thus delay of the overlay links connected to those nodes may not be stable also. We address this issue as our third contribution by introducing *MODE-m* extending from our initial work MODE, where the ALM tree is constructed carefully and dynamically to prevent those hosts from staying at critical positions that affect the end-to-end latency and the stability of the overlay multicast.

The utilizability of end-users as one-to-many data relay agents is one major fact that ALM has attracted a great deal of attention. However, the *reliability* problems of end-users induce the one of major drawbacks in the sense of spreading ALM into the real world applications. One major issue that comes up here is the end-users' desire-heterogeneity to the ALM session in which it is involved; since less-desired users leave the session sooner and often without any prior notification or cause delay and harmful jitter when they stay in the session, the participants in the downstream suffer considerable degradation of quality. This dissertation addresses this reliability issue by recognizing it as the *user-lifetime* and proposes a stability oriented overlay multicast scheme as its fourth contribution, which covers an aspect different from the previous MODE, STS or MODE-m.

The simulation experiment results, together with some experiments on real networks and Planetlab has shown that all of the above ALM schemes perform well in achieving its target.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid growth of communication technologies and the wide spread of high-bandwidth end-users such as DSL, cable and optical fiber subscribers, the Internet has shifted to a more complex, multi-purpose communication media from the conventional web and email platform. Some typical applications of this new communication paradigm are the Internet based multimedia streaming, video conferences, distance learning, multi-user games and file sharing. This kind of multi-user applications are commonly referred to as *group applications*. The multi-tier nature of these applications has resulted a demand for a new communication technology to send the same data to multiple users simultaneously, which stands in contrast to the conventional TCP/IP *point-to-point* (or *one-to-one*) communication method referred to as *unicast*. And this *one-to-many* or *many-to-many* communication method is commonly referred to as *multicast*.

IP multicast, which handles many-to-many communication in network protocol level by replicating packets at network routers, has been proposed in early 80's for this need. However, even after 20 years of its introduction, it is still not available as an open Internet service due to lack of infrastructure (eg: replacement of IP multicast supporting network routers) and existing Internet service (eg: inter career communication) support. Having the fact that open multicast services generally have not been available, an alternative called *Application Layer Multicast* (*ALM* in short) or *Overlay Multicast* in other words, has been proposed. ALM is a technology, which is based on the idea of using end-users to provide similar capabilities of network level IP multicast routers by acting as message replicating and relaying agents. Since the application running on each end-user covers the replication part, the actual message passing across the In-

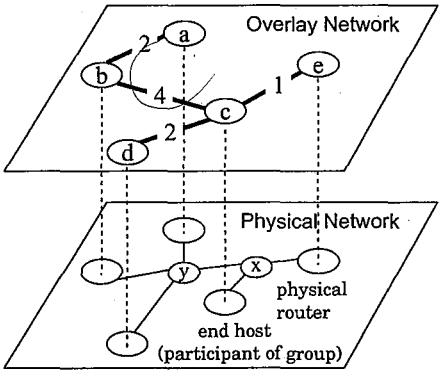


Figure 1.1: The Concept of Application Layer Multicast

ternet or the underlying network can be done using the currently available unicast as illustrated in Fig.1.1. The good about ALM is not only it can use the current network infrastructure, but also the utilizability of resource-rich end-user computing devices (commonly referred to as *peers* or *nodes*) for the message relaying, by which an innumerable variations of applications can be supported. Therefore, ALM has become the technology with the highest potential to realize the diversity of these group applications, despite it is not as efficient as IP multicast because of its delay and bandwidth penalties and less stability.

The ALM schemes (or protocols) can be classified into two main categories depending on how the end-hosts are organized. They are referred to as *unstructured* and *structured* (or *DHT*) overlays. Unstructured overlays use *floods* (flooding the query over the entire overlay network) or *random walks* (go through the overlay links randomly) for the data discovery after organizing the nodes into a random graph topology. Since there are no constraints on how the data is distributed on overlay nodes, unstructured overlays provide a high resilience to the node transiency. Structured overlays, on the other hand, use well structured overlay topologies and have constraints on the data placement. Here, data objects and nodes are assigned unique keys (identifiers) and queries are routed based on these keys, by which it can largely reduce the cost of data search. However, structured overlays are less resilient to the node transiency as the data objects are not found if the nodes responsible for them have failed or disappeared. Structured overlays are mainly used for peer-to-peer (or P2P) network applications such as file sharing or messaging. On the other hand, unstructured overlays are used

for both P2P applications and group communication applications such as multimedia streaming or video conferences.

The ALM schemes for such group communication applications (simply referred by group applications for the rest) are classified into the following three categories based on their policies to generate overlay topologies: (i) *mesh-first* approaches like [1, 2], (ii) *tree-first* approaches like [3, 4, 5, 6, 7, 8] and (iii) other approaches like [9, 10]. The approaches of (i) first build mesh-like overlay networks, and then build trees on top of the overlay networks. The approaches of (ii) directly build trees as overlay networks. The approaches of (iii) include implicit composition of tree-like forms, for example, overlay nodes are hierarchically structured in Ref. [9]. The mesh-first approaches are more resilient to the node transiency as it manages redundant overlay links, while tree-first approaches are less network resource consuming as it utilizes a single tree. In this dissertation we mainly focus on this tree-based ALM schemes.

Although, all these group applications are similar in the sense of multicasting, they demand the support from the underlying ALM service in many dimensions, or have different *QoS* (*Quality of Service*) requirements in other words. For instance, a multimedia streaming application requires a higher bandwidth, while an interactive application like a video conference rather relies on a lower end-to-end (source to destination) delay. Table 1.1 summarizes these characteristics of some typical group applications. By looking at real world group applications, we can say that most of them involve interactivity, use some kind of multimedia data and have different community scales. Therefore, it is very important for an underlying ALM scheme to be (a) latency sensitive, (b) bandwidth sensitive and (c) scalable. In this dissertation, first, we propose an ALM scheme called *MODE* to meet these three major demands, which constructs a *Degree Bounded Minimum Diameter Tree* (*DBMDT*) in a de-centralized manner. The overlay tree, which is a spanning tree, constructed by this scheme satisfies the following conditions: (i) the maximum delay between any pair of nodes is minimized (referred by "minimum diameter"), (ii) the number of overlay links connected to each node (degree) is restricted by its bandwidth availability (referred by "degree bounded") and (iii) scalable with its de-centralized design. In addition to the above features, MODE is resilient for node failures (un-announced departures) by providing quick restoring for broken trees. For this, MODE proactively calculates a *node-to-connect* for each node using some local information and this node is used in case of isolation from the main tree. The experimental results using ns-2 have shown that MODE could achieve sim-

Table 1.1: Characteristics of ALM Applications, s: small, m: medium, l: large

Application	Scale	Restrictions		Number of Sources
		Bandwidth	Latency	
Multimedia Streaming	l	l	s	single
Video Conferences	s	l	l	multiple
Distance Learning	l	l	m	single
Multi-user Games	l	m	l	multiple

ilar diameters with CT[11] algorithm, a centralized scheme that greedily constructs a DBMDT, in a small computation time and small amount of control traffic even though MODE is autonomous and decentralized.

In addition to the above major QoS requirements, some applications require the consideration of the existence of multiple sources. For example, in a video-conference or a panel discussion, the pictures of some primary persons should be continuously delivered to the other audience. These sources are some times subject to change, but are not changed so frequently. For this kind of purpose, we propose a method for the de-centralized construction of spanning trees where the maximum delay *from those senders* is minimized", rather than DBMDT[6, 7, 12], where the maximum delay between any pair of nodes is minimized. This scheme, called *STS*, constructs a *sender-dependant Degree Bounded Minimum Diameter Tree* (*s-DBMDT*), where some basic ideas have been taken over from our previous work MODE. A java middleware based on STS protocol has also been designed and implemented. This middleware consists of an adaptation mechanism to identify and relocate the nodes of low streaming performance, and this helps to achieve a better multimedia streaming quality. The performance evaluation, which has been done based on experiments in both simulated networks and real networks, strongly indicates the efficiency and usefulness of our protocol.

Again, the end-user heterogeneity is also an important issue to be addressed. With the recent rapid deployment of high-spec mobile devices, the end-users are shifting from the conventional desktop PCs to various hand-held devices, such as mobile phones or PDAs. So the ALM schemes we design should be ready to adapt these mobile terminals as well. In this case the following problems come across; the mobile hosts may not be stable due to the limitation of batteries, computing and communication capabilities and so on, and thus not stable to relay data packets frequently. Moreover, they may use wireless links and thus delay of the overlay links connected to those nodes may not

be stable also. As a solution for this we propose *MODE-m* extending from our initial work MODE, where the ALM tree is constructed carefully and dynamically to prevent those hosts from staying at critical positions that affect the end-to-end latency and the stability of the ALM tree. The experimental results have shown that this dynamic feature is very effective to keep the diameter as small as possible under the existence of mobile nodes.

As stated at the beginning of this chapter, the utilizability of end-users as one-to-many data relay agents, is one major fact that ALM has gained a lot of attention as a research topic. However, the *reliability* problems of end-users induce the one of major drawbacks in the sense of spreading ALM through real world applications. One major issue that comes up here is the desire-heterogeneity of end-hosts to the ALM session in which it is involved; since less-desired users leave the session sooner and often without any prior notification or cause delay and harmful jitter when they stay in the session, the participants in the downstream suffer considerable degradation of quality. We address this reliability issue by recognizing it as the *user-lifetime* and propose a stability oriented overlay multicast scheme, which covers an aspect different from the previous MODE, STS or MODE-m. The extensive simulations and experiments conducted in PlanetLab have shown the usability of this protocol.

The contributions of the ALM protocols discussed in this dissertation can be summarized as follows; (1) decentralized construction of minimum delay spanning trees under bandwidth constraints, (2) formulation of a new problem associating multiple-sources in minimum delay spanning trees, (3) design and implementation of an adaptation mechanism that well supports media streaming, (4) consideration of heterogeneous end-users for interactive overlay multicast, and (5) decentralized construction of stability oriented spanning trees in multiple source context.

The rest of the dissertation is organized as follows. Chapter 2 discusses the related work and show the novelty of the ALM schemes discussed in this dissertation. Chapter 3 describes the protocol MODE, while Chapter 4 explains STS and MODE-m, which are the protocols based on some basic ideas of MODE. Chapter 5 presents the fourth contribution stated above. Finally, Chapter 6 concludes this dissertation.

Chapter 2

Related work

2.1 Latency Oriented Approaches

A number of investigations have been dedicated to multicast protocols which consider the failure of nodes. From the point of view of network architecture, these researches can be classified into the following categories; (i) multicast on wireless ad-hoc networks, (ii) native multicast (*e.g.* IP multicast) and (iii) application layer multicast (or overlay multicast).

Researches in the first category such as MAODV [13] focus on reorganization of groups according to the radio ranges of nodes. The main issue is to keep connectivity among nodes, therefore the tree optimization is another issue. Some researches in the second category consider dynamic change of tree forms for fault tolerance or improvement of performance (for example, dynamic core migration is considered in Ref. [14]) and others mainly focus on enhancing reliability by means of re-transmission and/or redundancy such as RMTP [15], rather than coping with hardware failure.

Different from the above categories, multicast researches in the third category 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 [5] proposes a centralized algorithm for constructing minimum spanning trees. Yoid [4] is similar to ALMI, however, Yoid together uses shared tree connection and mesh-like connection for robustness. Narada [1, 2] considers mesh-like overlay networks where a shortest path tree per source is constructed. NICE [9] considers hierarchical topology where leaders organize their logical sub-domains. Refs. [9, 16] also consider hierarchical

overlay multicast trees for scalability reasons. For the deployment of IP multicast, HTMP [17] focuses on the connectivity between IP multicast islands.

Our protocol is different from the above approaches in the following points.

- *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 DB-MDT problem with dynamic join/leave activities, because it requires very small amount of control traffic and simple operations.

- *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.

- *Tolerance to multiple node failures.* We consider multiple nodes' disappearances in a distributed environment and validate the soundness.

To our best knowledge, no existing method considers the above issues.

And also STS and MODE-m, differ from all the above proposals including MODE, from the aspect of the consideration for multiple sender nodes and heterogeneous (mobile) nodes. Furthermore, STS considers the practical aspect of multimedia applications by proposing a adaptation mechanism to identify and replace incapable nodes in its implementation. As for ALM protocol implementations, we can say that several overlay multicast researches have implemented their protocols. In Ref. [5], an application level multicast communication library called ALMI has been implemented in Java. ALMI supports both reliable communication and datagram communication, and provides basic operations for constructing and maintaining shared trees among end-hosts. On the other hand, in Yoid (Your Own Internet Distribution) Project [4] which promotes unification of unicast/multicast communication and protocol stability, wrapper scripts are provided as Yoid Software for Mbone tools such as vic[18]. The research group in Carnegie Mellon University has developed ESM, a native code toolset based on End System Multicast methodology[1]. This tool was used for distributing live video in SIGCOMM2002 conference. HyperCast is the Java implementation based on two methodologies, HyperCast[19] and Delaunay Triangulation[20] and provides socket-like Java APIs. RelayCast[21] is a middleware to aim at adapting to various

applications that require different metrics (bandwidth, delay or both), by component-based design and implementation.

Our objective to design and implement STS middleware is different from any of the above. First, we would like to realize DBMDTs in an efficient way, for real applications. As stated before, we consider that DBMDT is suitable for interactive multimedia applications. Our objective is to provide a middleware for such an application including media streaming. In our STS protocol the nodes can reside in appropriate positions of the multicast tree. Secondly, none of the above does experiments which we intend to do, for example, measuring node-to-node delays in video streaming.

Note that focusing on protocols designed for content distribution such as near on-demand applications, there are other well-designed protocols. Ref. [22] has investigated several related projects.

2.2 Stability Oriented Approaches

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 [11, 7, 5] 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 ([3, 4]), or restore schemes where the overlay topology is re-established using a previously or immediately calculated restore mission ([23, 12, 24]) are often used to realize this. The third one corresponds to the ALM schemes like [9, 1, 25], 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, [26] 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, [27] 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 [27], [28] 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.

Chapter 3

Decentralized Construction of Minimum Delay Spanning Trees Under Bandwidth Constraints

3.1 Introduction

Whenever we design an interactive application on top of tree-based overlay multicast protocols, we have to consider the maximum delay between nodes on overlay trees (referred to as *diameter*), because the diameter can be the delay of the interactive session (*e.g.* group conversation including two nodes at the both ends of the diameter path). Also in order to handle multimedia streaming (especially video streaming) on overlay trees, 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 chapter, we present a protocol called MODE (Minimum-delay Overlay tree construction by DEcentralized operations), which constructs a *degree-bounded* delay sensitive spanning tree as an overlay multicast tree. The construction problem of such a tree is known as the *degree-bounded minimum diameter tree (DBMDT)* problem, and it is NP-hard [6]. Therefore, Ref. [6] proposes a heuristic algorithm called Compact Tree (CT) algorithm which is similar with Prim's minimum spanning tree algorithm [29]. Since CT algorithm focuses on static and centralized construction of a degree-bounded minimum diameter tree, 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, our protocol aims at repairing the existing spanning tree in a simple, fast and decentralized way when multiple nodes' simultaneous or continuous disappearances occur. It also aims at shortening the diameter of the repaired tree.

In our protocol, a *collection phase* and a *normal phase* are alternately repeated. In each collection phase, the neighbor nodes of a node v collect the information about the sub-trees which will appear by v's possible disappearance. The information of a sub-tree includes the center node of the sub-tree's longest path. In the subsequent normal phase, when node v disappears, the neighbor nodes have already known the information of the isolated sub-trees. Therefore one of them can start a repair procedure and it can be executed quickly. Note that the procedure connects the isolated sub-trees through the center points of their longest paths to obtain a tree with a small diameter. Moreover, this procedure is tolerant of multiple nodes' simultaneous (or continuous) disappearances in a normal phase. The consistency of the protocol for such multiple nodes' disappearances is discussed under some assumptions.

Our experimental results using ns-2 have shown that our algorithm could achieve similar diameters with CT algorithm in small computation time and small amount of control traffic even though our protocol is autonomous and decentralized one.

This chapter is organized as follows. Section 3.2 formulates the degree-bounded minimum diameter tree problem. In Section 3.3, the way to collect the sub-tree information for repairing broken trees is given. Section 3.4 presents the protocol to process nodes' participations and disappearances. Section 3.5 shows the experimental results and Section 3.6 concludes the chapter.

3.2 Protocol Overview and Assumptions

3.2.1 DBMDT Problem

Hereafter, we call the participant nodes of an overlay multicast tree simply *nodes*. Our goal is to provide an autonomous and decentralized protocol that constructs Degree-Bounded Minimum Diameter Tree (DBMDT) as an overlay network, under nodes' participations and disappearances.

The definition of 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 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) 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.* $\max_{v \in V}\{d_{max}(v)\}$).

3.2.2 Overview of MODE

In our protocol, 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 3.5) while that of the normal phase is relatively long (*e.g.* one minute).

In a normal phase, our protocol copes with nodes' two kinds of operations, (i) join and (ii) leave/failure (referred to as *disappearance)*, in an autonomous and decentralized way. We consider nodes' disappearances as a good occasion to shorten the diameter of the current tree. Therefore, every time a disappearance happens, the tree is partially re-constructed (*i.e.* repaired) so that it can eventually converge to a DBMDT. 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 collects the sub-tree information in the precedent collection phase. This collection is done in a recursive (incremental) way. The collected information is used for quick and efficient execution of repair procedures in the normal phase.

When a node disappearances from the current tree T, the generated sub-trees are connected to each other and the new tree T' is constructed in such a way that the new diameter becomes smaller. For that, MODE utilizes a policy of connecting the "*center nodes*" together. The center node of a sub-tree is the node on the *diameter path* (longest delay path) of that sub-tree where the difference between the delays from the both ends of the diameter path to the node is the smallest.

For example, if node u on the diameter path $x_1 - y_1 - y_2 - x_2$ (shown with the thick line in Fig. 3.1(a)) disappears, MODE connects the center nodes (shown with squares)to each other and constructs the new tree T' as shown in Fig. 3.1(b). By

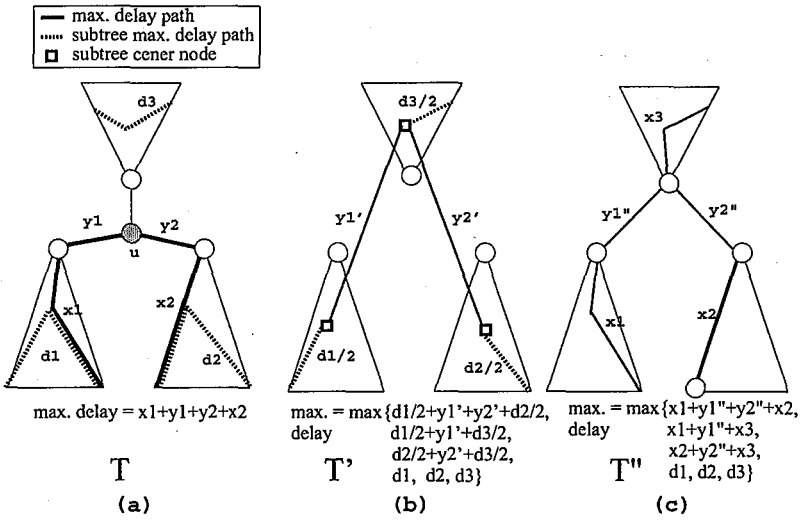


Figure 3.1: Different constructions by connecting sub-trees result in different maximum delay trees

connecting through the center nodes like this, the possible maximum delay from the connecting node in each sub-tree (the maximum delay from the connecting node to any other node in that sub-tree) becomes smaller (approximately $1/2$ of the maximum delay of that sub-tree) and that results a smaller diameter for T' in a higher possibility. On the other hand, if the neighboring nodes of the disappeared node are connected to each other as shown in Fig. 3.1(c), the maximum delay from the connecting node in each sub-tree, x_1 and x_2, which has composed the diameter of T as well, are still the maximum delays of those sub-trees. Hence, in this case, the diameter of the repaired tree T" might not be much smaller than T. Therefore, MODE utilizes the idea of connecting through the center nodes.

3.2.3 Assumptions

Nodes may disappear from the current 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 neighbor node's disappearance.

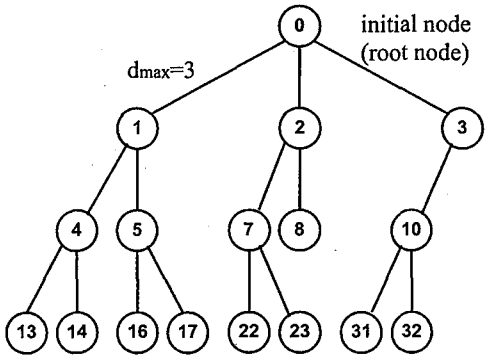


Figure 3.2: Node ID Assignment

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 causes a lose of any control message.

3.3 Information Collection on Tree

We explain MODE's collection phase in this section.

3.3.1 Node ID Assignment

In our protocol, when a node disappears from a tree, one of its neighbor nodes has the responsibility for detecting the disappearance and initiates the repair procedure. To let the neighbor nodes know who should be the initiator, a unique node ID is assigned to each node of the tree. It is fully refreshed at the beginning of every collection phase. For this purpose, we use the *algorithmic routing* in Ref. [30] where the assignment of unique node ID's is also presented[1]. Fig. 3.2 shows an example of the node ID assignment.

As we mentioned in the previous section, we assume that the initial node never disappears (see assumption G2). Node ID 0 is assigned to this node and it is called the

[1]Algorithm routing is a routing strategy on a spanning tree which relies on the original node ID assignment rules. We later use this routing strategy in Section 3.4.

root node. Here, let d_{max} denote the maximum degree bound of all the nodes. The root node starts each collection phase at regular intervals by broadcasting *synchronization messages* on the tree. When the root node (say node a) sends the messages to its neighbor nodes, it assigns node ID's 1,...,$d(a)$ to those nodes. Similarly, if a node v receives a synchronization message from a neighbor node and if it knows that node ID n is assigned to itself, it assigns node ID's starting from $n \times d_{max}+1$ up to $n \times d_{max}+d(v)-1$ to the rest of its neighbor nodes when it sends messages to them. Finally all the nodes in the tree have unique node ID's.

For each node v, the neighbor node with a smaller node ID than that of v is called the *parent* node (or simply *parent*) of v. The rest of the neighbor nodes with larger node ID's are called the *child* nodes (or simply *children*) of v.

3.3.2 Sub-tree Information Collection

For a pair of two adjacent nodes u and v, let T_uv denote the sub-tree rooted at node v generated by node u's disappearance.

After the broadcast of synchronization messages, for each node (say u), its neighbor nodes collect the information of the sub-trees which will appear by u's possible disappearance. This information is called *sub-tree information*. Using the information, one of the neighbors can start the repair procedure which connects the isolated sub-trees if node u disappears. As stated in the previous section, in order to shorten the diameter of the repaired tree, our idea is to connect the isolated sub-trees through their center nodes. For example, in Fig. 3.3, the diameter path of T_uv is [node a, node b, node c node d, node e], and node c is the center node of T_uv (we assume the delays of links are the same).

For such a purpose, each neighbor node of node u collects the information of all the sub-trees connected to node u. For example, in Fig. 3.3, node v keeps the sub-tree information of T_uv, T_uv' and T_uv'' to provide for node u's disappearance. It also keeps the sub-tree information of T_cv, T_cb and T_cd to provide for node c's disappearance. The sub-tree information of T_uv includes the followings.

- network addresses (*e.g.* IP address) and node IDs of v and its neighbors other than u. Hereafter we denote these neighbors of v as $neighbor(v)$.
- $dia(T_uv)$: the diameter of T_uv
- $center(T_uv)$: network address and node ID of the center node of T_uv

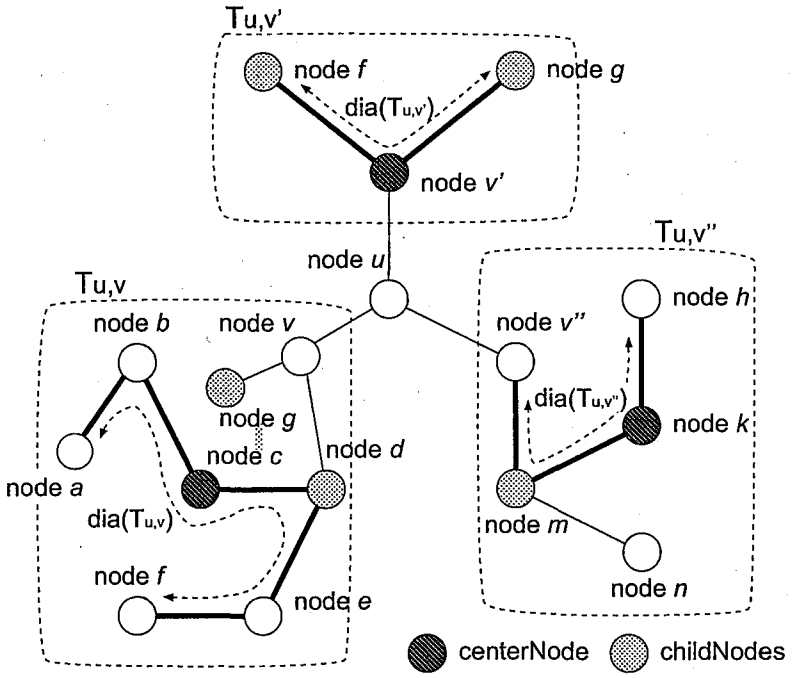


Figure 3.3: Example of Sub-trees

For example, $dia(T_u v)$=4 and $center(T_u v)$=c. where we assume that the maximum degree $d_{max}(x)$ is 4 for all the nodes x, and the delays of all the links are the same.

When a leaf node (say z) receives a synchronization message, z sends a *heartbeat* message to its parent y. Node y sends a heartbeat message to *each* neighbor node x whenever it receives heartbeat messages from all the neighbor nodes except x.

Each node v is responsible for calculating the sub-tree information of $T_u v$. The calculated information must be included into the heartbeat message from node v to node u. When a node v receives heartbeat messages from all the neighbor nodes (say $w_1, ..., w_{d(v)-1}$) except u, the sub-tree information of $T_u v$ can be calculated at node v. For such recursive calculation of sub-tree information, 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.
- $H(T_u v)$: the node list of the maximum delay path of $T_u v$ from v. For each node

$z \in H(T_u v)$, the network address and the node ID are included.

They can be defined in a recursive way as follows. Here "@" denotes the concatenation of node lists.

$$depthT_u v = \max_{1 \le j \le d(v)-1} \{depthT_v w_j + c(v, w_j)\}$$

For w_j which maximizes the above,

$$H(T_u v) = [v] @ H(T_v w_j)$$

Next, we can also define $dia(T_u v)$ in a recursive way as follows.

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

where

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

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, $center(T_u v)$ can be defined as follows.

$$center(T_u v) = \begin{cases} center(T_v w_j) & (\text{if } dia(T_u v) = dia(T_v w_j)) \\ \text{center node on "}jointpathlist\text{"} & (\text{if } dia(T_v w) = jointdepth) \end{cases}$$

where

$$jointpathlist = \text{reverse}(H(T_v w_x)) @ [v] @ H(T_v w_y)$$

and "reverse" is the reverse function of a list. If the diameter of $T_u v$ is the same as that of a sub-tree $T_v w_j$, the corresponding $center(T_u v)$ 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 the center node on the new diameter path.

The heartbeat message sent from v to u includes:

- the sub-tree information of $T_u v$,
- $depthT_u v$ and $H(T_u v)$ and

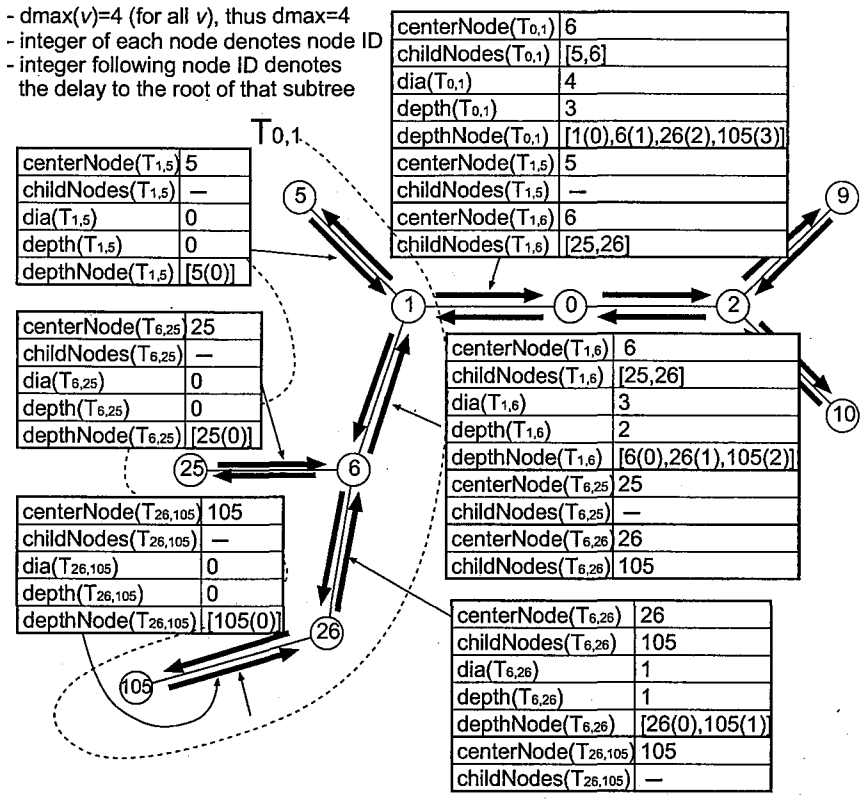


Figure 3.4: Sub-tree Information Collection: Example

- the sub-tree information of $T_v w$ for each w except u.

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

3.3.3 Example

Fig. 3.4 shows how heartbeat 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 heartbeat message from node 105, calculates the sub-tree information (and auxiliary parameters) of $T_6 26$ and sends it to node 6, together

with the sub-tree information of $T_{26}105$. If node 6 receives heartbeat messages from nodes 25 and 26, it calculates the sub-tree information (and auxiliary parameters) of T_16 and sends it to node 1 together with T_625 and T_626's sub-tree information. If node 1 receives the heartbeat messages from nodes 5 and 6, it calculates the sub-tree information (and auxiliary parameters) of T_01 and send it to node 0.

Similarly, node 1 receives the heartbeat message from node 0. If it receives the message, it calculates the sub-tree information of T_61 and T_51 and sends them to nodes 6 and 5, respectively. Since node 1 knows all the three sub-tree information of T_625, T_626 and T_61, 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.

3.4 DBMDT Construction Protocol

This section presents DBMDT construction protocol which consists of two procedures to process join and disappearance of nodes in normal phases. We make the following two assumptions concerned with normal phases to make the discussion simple. Later we try to relax these two assumptions (see Section 3.4.5).

N1. Node disappearances never occur in collection phases.

N2. 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.

N3. 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.

3.4.1 Join Procedure

The join procedure is simple. Here we take the approach of connecting the newly joining node to be closest to the center node of T. A new node which wants to join the current tree T, first sends a query message to the root node [2] asking the address of the center node. In MODE, any node can calculate the tree's center node using the sub-tree information received from each neighbor. So the root can send the center node's address to the queried node once it has entered to the normal phase. In case of the

[2]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.

impossibility of center node calculation (*i.e.* before the first collection phase has been completed) or the center node's early disappearance, the root node sends the address of itself.

Once the joining node receives the reply from the root node, it sends a *connection request message* to the center node. The center node, which receives the connection request messages, sends a *connection acceptable message* to the joining node if it has the potential of accepting a new child. And, at the same time it broadcasts the connection request message to its child nodes[3] . The child nodes also treat the connection request message in the same way. Note that these nodes send no message to the joining node if they have no potential of accepting a new child. Here, the delay from the center node is also added to the connection request message before it is broadcast to the child nodes. So every node which sends a connection acceptable message to the joining node includes the delay from the center also. Then the joining node uses these values and the delay to each responded node (this can be measured using *ping* for instance) to calculate the *candidate connecting node* which has located itself at the position closest to the center node. And the newly connected nodes do not involve in repair procedures of other node disappearances before they finish their first collection phase.

Here, the potential of accepting a new child is discussed in terms of the residual degree. A node is said to have the potential of accepting a new child if its residual degree is greater than a certain threshold k. Our simulation results have shown that constructing the initial tree with some residual degree in each node makes more diameter reductions in later repair procedures, and we have set $k = 0.8d_{max}$.

3.4.2 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. 3.5. For node v's disappearance, its parent (node u, this is called the *repair initiator*) activates the repair procedure and sends the information of $T_v w_j$ ($1 \leq j \leq d(v) - 1$) to the center node of $T_v u$ (this is called the *repair master*). Us-

[3]Here we can set a suitable forwarding count limit to prevent the joining node from receiving a huge number of connection accepting messages.

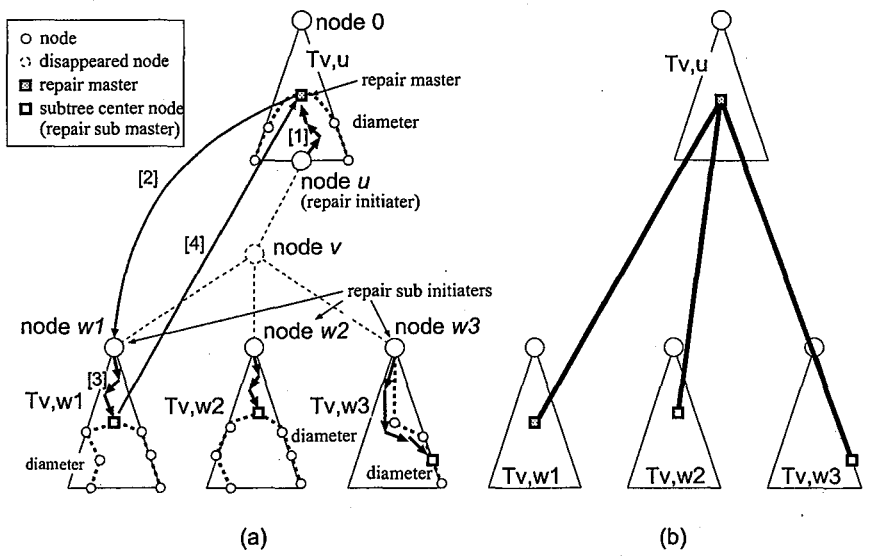


Figure 3.5: Repair Procedure (single node disappearance)

ing the node ID of the repair master, this delivery is done on the tree according to the algorithmic routing introduced in Ref. [30]. This step is named phase1 (refer Fig. 3.6). The repair master then sends its address to each w_j $(1 \leq j \leq d(v) - 1)$ (these are called *repair sub-initiators*) and handovers the repair process to them (phase2). Then Each w_j sends the repair masters address to the center nodes (called *repair sub-masters*) of the sub-trees where they are the root nodes (see Fig. 3.5(a)) (phase3). This is also done according to the algorithmic routing. Now each repair sub-master of $T_v w_j$ $(1 \leq j \leq d(v) - 1)$ knows the address of repair master, so they connect themselves to the repair master in the same way discussed in Section 3.4.1 (phase4). Note that if the repair sub-master has no residual degree, its closest node with a residual degree is handed over the connection process.

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.

Once this repair is done, the repair master and the repair sub-master exchanges the address and the IDs of their neighbor nodes over the new link.

3.4.3 Repair Procedure for Multiple Disappearances

Simply applying the above procedure to multiple nodes' disappearances occur simultaneously or consecutively, might result in loops or isolated sub-trees. Therefore, we discuss the required extensions to the above repair procedure to handle these cases in this section.

E1. If the repair initiator (the parent of v) has also disappeared, $R(v)$ is not initiated. In this case, the protocol is extended such that every neighbor node of v probes for other neighbors existence when it detects v's disappearance, and the node with the smallest ID (say w) becomes the repair master of $R(v)$, while the rest become the repair sub-masters. Note that the assumption N2 guaranties that at least one neighbor node of v is alive, so the repair initiator of $R(v)$ is always found. And this time the center node of $T_v w$ becomes the repair master and connects the repair sub-masters in the sub-trees except $T_v u$.

In the same way, if any of the repair sub-initiators, w_j $(1 \leq j \leq d(v) - 1)$, has disappeared, the repair master cannot send its address to the corresponding repair sub-master. In this case, the repair master selects w_j's child node (say w_k) located on the path from w_j to the center node of $T_v w_j$, or another node randomly if w_k does not exist.

E2. If a node on the path from $R(v)$'s initiator to repair master has disappeared, the sub-tree information of other sub-trees cannot be delivered to the repair master. Therefore, in this case the node just before the disappeared node becomes the repair master instead. Similarly, if a node between a repair sub-initiator and a repair sub-master has disappeared, the node just before the disappeared node is selected.

E3. If an edge (v, w), which has been created by a previous repair procedure is broken by node v's disappearance, the repair procedure cannot be initiated as $T_v w$'s sub-tree information is not known to v's neighbors. Therefore, in this case, $R(v)$ is not applied for $T_v w$, and a local repair procedure which connects v's neighbor with the smallest ID to $T_v w$ at w (or one of w's neighbors) is applied instead.

3.4.4 Repair procedure consistency

In this section, we show the consistency of MODE by proving the consistency of $R(v)$ after the extensions E1, E2 and E3 are applied. Here the protocol is said to be consistent if no loops and isolated sub-trees are created.

Hereafter, a node disappeared at the same time or later of v's disappearance is denoted by v'. The timing of v''s disappearance can be categorized to one of the phases (phase0 - phase6) of $R(v)$ (see Fig. 3.6). Note that phase0 denotes the time between v's disappearance and $R(v)$'s initiation, and phase6 is the time after phase5. Here, v''s role in $R(v)$ is one of the following.

1. the initiator or the sub-initiator of $R(v)$
2. a node located on the path from $R(v)$'s initiator to the repair master (including the repair master) or a node located on the path from $R(v)$'s sub-initiator to the repair sub-master (including the repair sub-master)
3. some other node

We show $R(v)$'s consistency for any combination of $v's$'s role and $R(v)$'s phase following the above categorization. Again, we show the consistency of two simultaneous repair procedures by discussing the consistency of $R(v')$ in $R(v)$. This can be easily extended to show the consistency of repairing n node disappearances.

First, we discuss the sub-case of (1) where v' is the initiator of $R(v)$ (see Fig. 3.7(a)). According to Fig. 3.6 if v''s disappearance occurs in or after phase1, $R(v)$ is not affected by v''s disappearance, so $R(v)$ and $R(v')$ can be considered as two independent repair procedures (this case is discussed later in this section). So here we discuss v''s disappearance in phase0. In this case, since assumption N2 assures the existence of at least one child node, v's one child becomes the initiator of $R(v)$ according to extension E1 (see Fig. 3.7(a)) and connects the sub-trees other than $T_v v'$ (see Fig. 3.7(b)). On the other hand v is the sub-initiator of $R(v')$. However, the discussion on this can be omitted considering the consistency of the case where v' is the sub-initiator of $R(v)$, which is discussed below. Furthermore, $R(v')$ treats $T_{v'} v$ as a single sub-tree, so after applying $R(v)$ and $R(v')$ the tree is repaired without loops and isolated sub-trees (see Fig. 3.7(b)).

Similarly, in the sub-case of (1) where v' is the sub-initiator of $R(v)$, consideration of only in or before phase2 is required according to Fig. 3.6. In this case, sub-tree

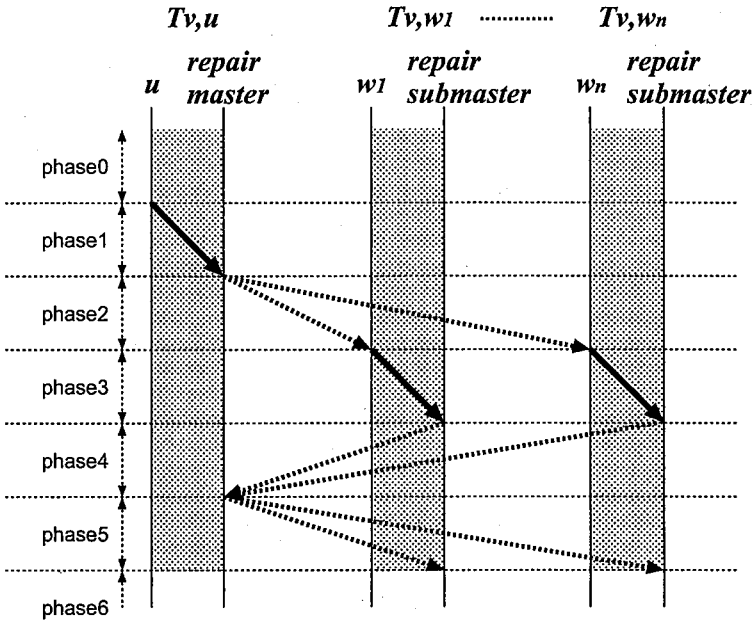


Figure 3.6: Timing Chart of $R(v)$

information delivered to the repair master contains the information about v''s neighbor nodes too. So the repair master can use one of these children on behalf of v' and can connect $T_v v'$. Though v is the initiator of $R(v')$, this is already discussed in the previous discussion. Therefore, $R(v')$ connects $T_{v'} w_j$ $(1 \leq j \leq d(v') - 1)$ to each other and makes $T_v v'$ a single tree. Finally, $R(v)$ connects $T_v v'$ to $T_{v'} v$ to make a single spanning tree.

In the sub-case of (2) where v' is located on the path from $R(v)$'s initiator to the repair master (see Fig. 3.8(a)), only the time in or before phase2 is required to be discussed (see Fig. 3.6). That is because, a disappearance of v' which takes place in or after v does not affect $R(v)$, so $R(v)$ and $R(v')$ run independently. In this case, the node just before v' (say w') can be found by $R(v)$ in its algorithmic routing process towards the repair master. So w' becomes the repair master according to assumption E2 and repairs $T_{v'} w'$ to a single spanning tree (see Fig. 3.8(b)). At the same time, $R(v')$ treats $T_{v'} w'$ as a single sub-tree and connects it to other isolated sub-trees created with v''s disappearance (see Fig. 3.8(b)).

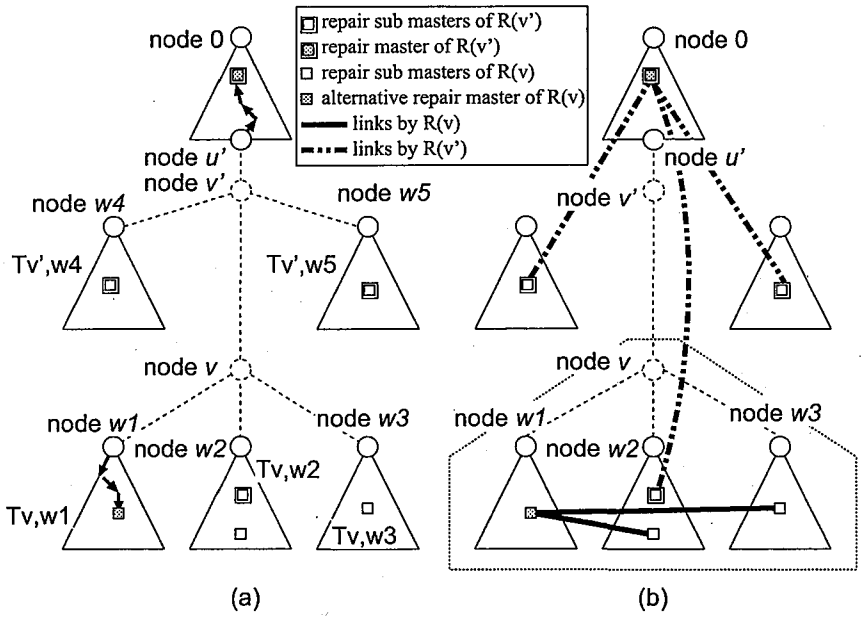


Figure 3.7: Repair Procedure : v' is the Initiator of $R(v)$

Similarly, in the sub-case of (2) where v' is located on the path from $R(v)$'s sub-initiator to the repair sub-master, consideration of v''s disappearance in or before phase3 is adequate, and a alternative repair sub-master is found in the same way. Note that in case of that v' is the repair master (repair sub-master), disappearance of v' in phase2, 3 and 4 (phase4, 5 and 6) does not take place according to assumption N3. Then, if v' disappears in or after phase5 (phase6), $R(v)$ and $R(v')$ can be considered as separate independent processes, and this is discussed later.

In case of (3) v''s disappearance occur in any phase is independent of $R(v)$ and $R(v)$ and $R(v')$ are independent.

Finally we discuss the cases where $R(v)$ and $R(v')$ are independent. In these cases, it is obvious that v' does not affect $R(v)$. However, $R(v')$ is executed in a situation such that some neighbor nodes have already disappeared and some new nodes have joined the tree. Here the repair consistency for the disappeared nodes can be shown by replacing v' with v, in the above proof of $R(v)$'s consistency. And for the newly joined nodes, a local repair (simply connecting the neighbor nodes) is applied instead of $R(v')$ according to the extension E3. With this the consistency of $R(v)$ with v''s

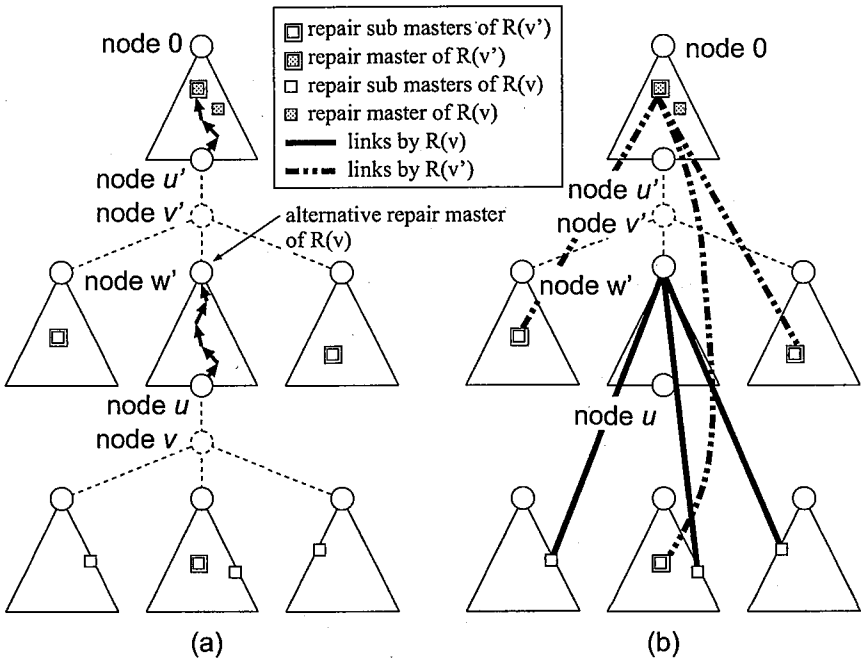


Figure 3.8: Repair Procedure : v' is on the Path from the Initiator to the Repair Master of $R(v)$

disappearance in any timing of $R(v)$ has been proved.

3.4.5 Unexpected disappearances

In this section, the repair procedure for the node disappearances occur when the assumptions N1, N2 and N3 described earlier are not followed or before the first collection phase, are discussed.

First we take the case where assumption N1 is not followed. To make the discussion easier we assume that the synchronization message reaches all nodes in the tree[4].

If a node disappears in the collection phase its all neighbors are in the collection phase. This is because they are either nodes which are waiting to send the collection message, or nodes which are waiting to receive the collection message. In this case, the repair is done in the following procedure. First, all neighbor nodes of v shift to the

[4]If not, there might be some nodes which cannot receive the synchronization message. In this case, a time out can be set to make them join the tree following the join procedure after deciding that they are isolated

normal phase neglecting v and the sub-tree rooted at it. Then each child w of v joins to T's root node with the sub-trees rooted at them.

When the assumptions N2 and N3 are not followed, isolated sub-trees might be generated as some nodes which play a important role in the repair procedure might not exist. In this case, nodes in these isolated sub-trees can find there isolation by using the synchronization message time out for instance. Then the node with smallest ID in each isolated sub-tree can connect itself to the root node of T to recover the isolation. However, in this case the isolation time depends on the synchronization message interval and such a long isolation may be intolerable. One possibility of overcoming such a long isolation is to make the root node send a probe message in every short period and make the isolated nodes find there isolation sooner.

3.5 Performance Evaluation

In this section, we discuss the performance evaluation done by simulation experiments.

3.5.1 Simulation Setup

We have implemented our MODE protocol on ns-2. In our experiments, networks with 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 of them ranges from 10ms to 200ms.

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 a groupware. Note that we have set the interval between collection phases to 60 seconds. The degree bound has been set to 5 for all the nodes. The scenario is as follows. (i) The session period is 300 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 270 seconds, additional joins are processed. Also some existing nodes leave the session. The collection phases starts at 30, 90, 150, 210 and 270 seconds successively. (v) After 270 seconds, no node joins take place and about 40 nodes leave the session.

We have shown the number of nodes on the tree at every second together with the numbers of join/leave operations in Fig. 3.9 and Fig. 3.10 to make it facilitate to see the dynamics of the metrics according to the scenario.

3.5.2 Implementation of Compact Tree Algorithm

As a comparison, we have also implemented the centralized algorithm presented in Ref. [6] (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 the diameter of the resulting tree can be minimized 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 two implementation policies of CT algorithm to make it work with nodes' join/leave operations. (i) For each join or leave of a node the existing spanning tree is completely destroyed and reconstructed according to the CT algorithm. Hereafter this implementation is denoted as CT. Though CT can maintain a near optimal diameter, the number of disconnecting and establishing links is held high as the tree is reconstructed each time. (ii) For each join node the connection-end that makes the best diameter is calculated using CT algorithm. And when node disappearances occur the best connection-end is calculated in the same way for each isolated node. In the both cases above, the tree destroying before the reconstruction is not done, but the un-isolated part of the tree is kept as it is. Hereafter this implementation is denoted as partial-CT. partial-CT reduces the number of disconnecting and establishing links comparing to CT, but the diameter becomes more larger. In the both of above implementations the entire tree knowledge is used.

3.5.3 Experimental Results

[Diameter variance against the number of nodes] The diameter variance through the session is shown in Fig. 3.9. We can see that the diameter has started to decline from its initial value of 225[ms] of initial tree construction phase (0[s]-60[s]). And has reached to a stability around 60[s] where joins/leaves are repeated. Through out the session, MODE's diameter is quite close to that of partial-CT, though it is somewhat faraway from CT's diameter and we can say MODE provides a good distributed reconstructing method.

[Diameter and repair cost] We define the number of overlay links which are established and disconnected during the repair procedure as the *repair cost*. The average, worst values of the diameter and the average repair cost is calculated by averaging 10 sessions executed in different networks. And these values are shown in Table 3.1 di-

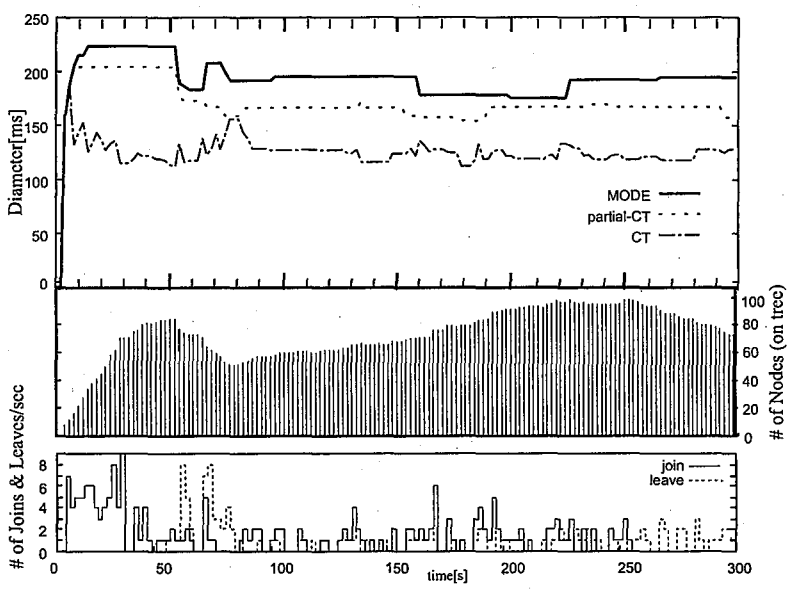


Figure 3.9: Dynamics of Diameters

viding the session in to 3 sub periods of initial, stable and last. Here the periods of 0[s]-50[s], 50[s]-240[s] and 240[s]-300[s] are named as initial, stable and last period successively. According to Table 3.1, MODE holds a diameter that is 1.47 times of CT and 1.14 times of partial-CT in stable period, while that is 1.52 times of CT and 1.13 times of partial-CT in initial period. And in the last period that is 1.39 times of CT and 1.14 times of partial-CT. And for the worst values of diameter, MODE has shown a value that is 1.19 times of CT and 1.09 times of partial-CT which can be considered to be reasonable. Obviously, MODE's repair cost is quite small that is 0.05 times of CT and 0.38 times of partial-CT. Taking the above results into account, we can say MODE gives a reasonable diameter with smaller repair cost comparing to the existing centralized algorithms, though it copes with node joins and leaves in a decentralized way.

[Control Traffic] We have measured the control traffic amounts on the tree. The result is shown in Fig. 3.10.

Due to small message complexity, MODE has required the maximum of about 200Kbps around 30[s] collection phase (we can see the peaks around 30[s], 90[s],

Table 3.1: Comparison of Diameter and Repair Cost

	diameter				repair
	average			worst	cost
	initial	stable	last		avg.
CT	136	137	132	187	58.3
Partial-CT	184	164	161	204	7.5
MODE	208	188	184	224	2.9

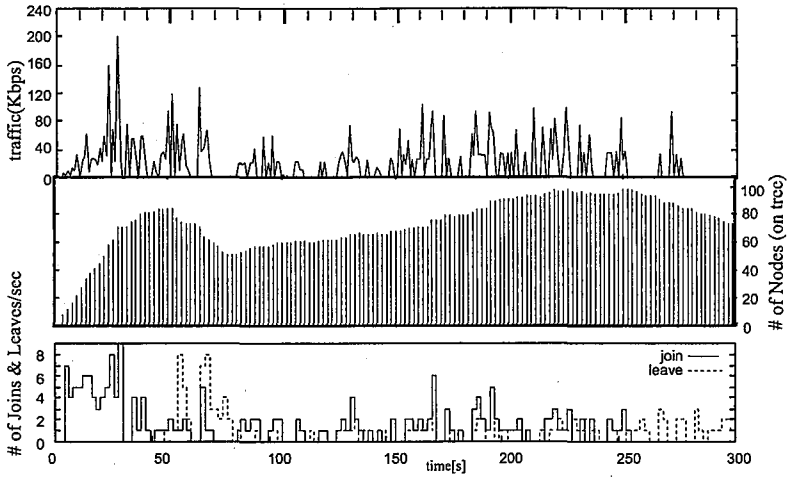


Figure 3.10: Control Traffic

150[s], 210[s] and 270[s]). This is the traffic for the whole network and the average traffic for a single overlay link has remained 3Kbps. And considering the maximum degree is 5 for each node, the traffic of the physical network link is held under 15Kbps. We can say this amount is small enough considering the bandwidths of DSL connectivity which is getting spread drastically.

[Time for Procedure Execution] Finally, we have measured the time required to execute the join/repair procedures. Their distributions are shown in Fig. 3.11.

We can see that the all the join procedures have only required at most 0.5 seconds, while the all the repair procedures have only required at most 0.9 seconds. The expected average values of the join and repair procedures are 0.26 and 0.70 seconds, respectively. From the results, we can say that the procedures were processed quickly enough.

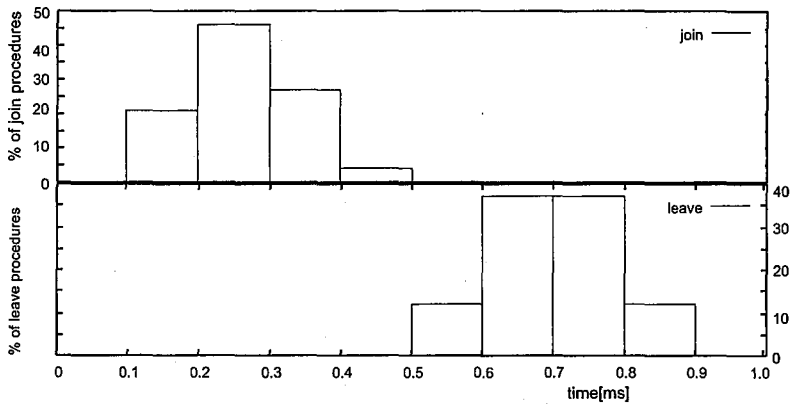


Figure 3.11: Join/Repair Procedures' Processing Time Distribution

3.6 Concluding Remarks

In this chapter, we have proposed an autonomous and decentralized protocol for dynamically constructing a degree-bounded delay sensitive multicast tree on overlay networks where multiple nodes' simultaneous (continuous) join/leave activities in a specified period is considered.

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

Chapter 4

Coping With Multiple Sources and Heterogeneous Users

4.1 Association of Multiple Sender Nodes

In the previous chapter we have presented a protocol called MODE (Minimum-delay Overlay tree construction by DEcentralized operation). MODE aims at minimizing maximum delay (referred to as *diameter*) between any pair of nodes under the degree-constraints given by the nodes, assuming interactive applications where every node can be a potential sender. Through our investigations, we have learned that tree-based approaches are simpler, since managing many trees makes operations more complex and may increase overhead. In interactive applications, multiple nodes may often be potential senders, therefore, we think that a shared tree approach is preferable.

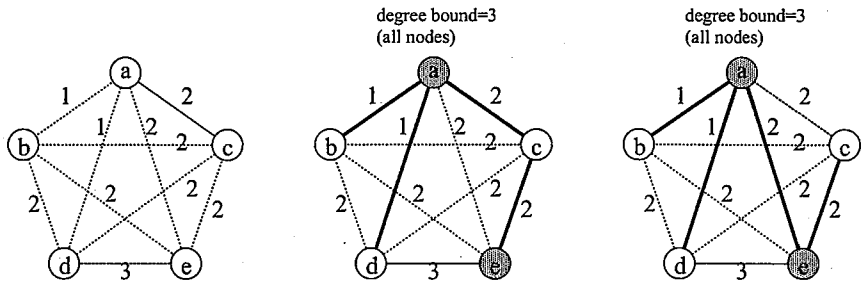
Whenever we design such a shared tree approach, we should consider the following points. (i) The maximum delay of overlay trees should be minimized, because the diameter can be the delay of the interactive session (*e.g.* group conversation including two nodes at the both ends of the diameter path). (ii) Bandwidth constraints around nodes should be taken into account. 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 of each node should not exceed the capability limitation of the node. Considering these facts, some techniques for constructing Degree-Bounded Minimum Delay (or Diameter) Trees (DBMDTs) have been proposed [6, 7, 12]. However, unfortunately none of those methods has considered the following several important features of interactive *multimedia* applications

such as video conferencing.

First, such an application may have several sources. These sources are subject to change, but are not changed so frequently. For example, in video-conferencing, pictures of some primary persons should be continuously delivered to the other audience. In such an application, we would like to efficiently build a tree where the maximum delay *from the current senders* is minimized satisfying the degree bounds of nodes, rather than DBMDT[6, 7, 12] where the maximum delay between any pair of nodes is minimized. Such a tree may have shorter delay than DBMDT if we only focus on the delay from those sources. Hereafter, for a given set of senders, such a tree is called *sender-dependent DBMDT* and denoted as *s-DBMDT*. Fig. 4.1 shows an example that explains the difference between DBMDT and s-DBMDT. For a given complete graph that represents an overlay network in Fig. 4.1(a), DBMDT is a spanning tree which involves all the nodes of the graph and has a minimum diameter, as shown in Fig. 4.1(b). In this case, the diameter path is *b*-*a*-*c*-*e* (or *d*-*a*-*c*-*e*) of delay 5. Here, let us suppose that currently only nodes *a* and *e* are senders, which are shown by meshed circles in the figures. In this case, considering the fact that only these nodes send data and others are receivers, s-DBMDT in Fig. 4.1(c) achieves smaller maximum delay 4 (*a*-*e*-*c*) from those senders, while 5 (*e*-*c*-*a*-*b* or *e*-*c*-*a*-*d*) in DBMDT of Fig. 4.1(b).

Secondly, in a practical aspect of interactive multimedia applications, we need to identify incapable hosts and prevent them from staying in the center of s-DBMDT, since those hosts may delay or drop packets due to limitation of network bandwidth or processing power to forward packets, or instability of hosts. Similarly, we should provide a reasonable way to allow each host to determine an appropriate degree bound, since the capability overflow of such a host may also cause packet delay or dropping at the host.

In this chapter, we propose a new protocol called *Shared Tree Streaming (or STS in short)* protocol that constructs, in a decentralized manner, s-DBMDT adaptively. We also design and implement a Java middleware based on STS protocol. STS is a distributed heuristic algorithm to build a s-DBMDT as an overlay network. The features of the proposed STS protocol have not been considered in the existing literatures, including our previous work. Compared with those work, our contribution can be summarized to the following two points. First, we define a new problem that is well-suited to multimedia interactive applications and design a new decentralized protocol for the problem. Secondly, we have designed and implemented an adaptation mechanism that



(a) An Overlay Network (b) DBMDT (dia.=5,s-dia.=5) (c) s-DBMDT (dia.=5,s-dia.=4)

Figure 4.1: DBMDT and s-DBMDT on Overlay Network

is needed for multimedia streaming on overlay shared trees. Our performance evaluation is based on experiments in both simulated networks and real networks that strongly shows the efficiency and usefulness of our protocol.

This chapter is organized as follows. Section 4.2 gives the overview of our STS protocol. The detailed design is shown in Section 4.3. The design and implementation of STS middleware in Java, denoted by STS/J, are stated in Section 4.4. The experimental results on a simulated network and a real network are shown in Sections 4.5 and 4.6, respectively. Finally, Section **??** concludes the chapter.

4.2 Shared Tree Streaming (STS) Protocol

4.2.1 Definition of DBMDT and s-DBMDT

First, we revise the definition of a Degree-Bounded Minimum Diameter Tree (DBMDT), which has also been discussed in the previous chapter. 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 of each node $v \in V$ (the maximum number of overlay links attached to v), and let $h(i, j)$ denote the delay of each overlay link $(i, j) \in E$. DBMDT is a spanning tree T of G where the *diameter* of T (the maximum delay on T) is minimum and the degree of each node $v \in V$ (denoted as $d(v)$) does not exceed $d_{max}(v)$.

Based on the above, we define a sender-dependent DBMDT (s-DBMDT) introduced in this chapter as follows. For a given $G = (V, E)$ and a given set $S \subseteq V$

of senders, s-DBMDT is a spanning tree T of G where the maximum delay from the senders in S is minimum and $d(v) \leq d_{max}(v)$ where $v \in V$. The maximum delay from the senders is called *sender-dependent diameter* and denoted by *s-diameter*.

The DBMDT construction problem has been proved to be NP-complete [6]. The DBMDT construction problem is a special case of our s-DBMDT construction problem where $S = V$. Therefore, we need an efficient heuristic algorithm for the problem.

4.2.2 Key Idea for Minimizing the Maximum Delay from Senders

Our goal is to build s-DBMDT efficiently in such an environment where nodes join and leave the tree dynamically and senders in the tree vary from time to time. In order to minimize the maximum delay from those senders, a new node should be connected to an adequate position in the current tree, and disconnected sub-trees by a node's leaving should be repaired by connecting adequate nodes in the sub-trees. For such a purpose, we design a decentralized heuristic protocol called Shared *Tree Streaming (STS) protocol* that consists of two procedures, join and repair. The join procedure makes a new joining node connect to a node which positions the joining node "closest to the senders" of the current tree in order to prevent the new node from making the s-diameter longer. The repair procedure is activated when a node on the current tree leaves (or suddenly disappears) and it connects appropriate intermediate nodes in the isolated sub-trees to make a new tree with a shorter s-diameter. To execute the above procedures in decentralized manner, each node in our STS protocol autonomously collects the information about the current sender nodes and diameter paths of the sub-trees that will appear by neighboring node's leaving. This information collection is executed periodically to keep up with the status changes of the tree (e.g. location of sender nodes and diameter paths of the sub-trees). This will be explained in Section 4.3.2. In this sub-section, we explain how the two procedures keep the s-diameter as small as possible, satisfying given degree bounds of nodes.

Join Procedure Outline For a new joining node u, the join procedure never changes the current form of the tree, but lets the new node u connect to such a node (say v) where the maximum delay from the senders to the new joining node u (*i.e.* a candidate for the s-diameter of the consequent tree) is minimum. Note that node v must be such a node that has at least one residual degree.

Fig. 4.2 shows an example where senders are denoted by meshed circles. We

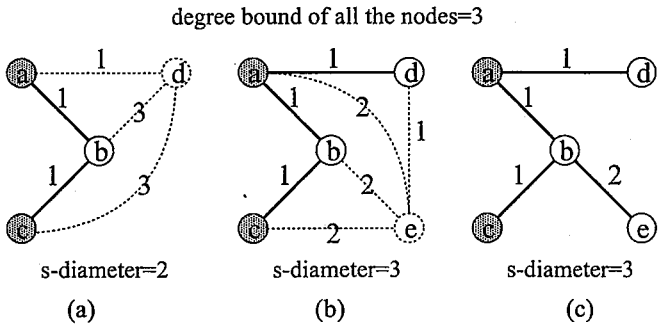
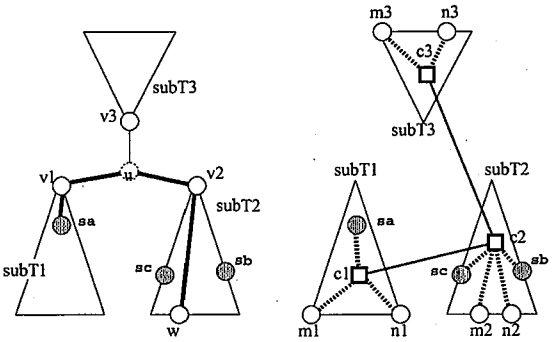


Figure 4.2: Join Procedure

assume that the degree bounds of all the nodes are 3. In Fig. 4.2(a), we have a tree involving three nodes a, b and c where a and c are senders. Also the s-diameter of the tree is 2. Let us assume that a node d wants to join the tree, and the dotted lines represent the measured delay between the new node d and the existing nodes on the tree. Consequently, node d is connected to node a since the maximum delay from senders a and c becomes 3 (Fig. 4.2(b)) and it is the minimum in all the possible connecting positions. Let us also assume that after the join of node d, node e wants to join to the tree, and it is connected to node b since the maximum delay from the senders to the new node e is minimum (=3) (Fig. 4.2(c)). This keeps the s-diameter the same as before.

Later we will explain how this operation is realized in a distributed environment.

Repair Procedure Outline Whenever a node's disappearance occurs, the disconnected sub-trees are repaired by the repair procedure. At that time, we try to shorten the s-diameter of the repaired tree, by connecting the *core nodes* of the isolated (disconnected) sub-trees. Here, let m and n be the both end nodes of the diameter path (*not s-diameter path*) of a tree t. The *core node* of t is a node whose maximum delay from the nodes m and n is minimum in all the nodes of t. This means that the core node is located on the "center" of the diameter path. Here, we will explain why such a procedure can make the s-diameter of the repaired tree shorter. An example is shown in Fig. 4.3 where the senders are s_a, s_b and s_c represented by meshed circles. In tree T of Fig. 4.3(a), let us assume that its s-diameter path is s_a-v_1-u-v_2-w and a new node u leaves the tree. By the leave of node u, the tree T is partitioned into three sub-trees



(a) T (b) T'

	path	type
1.	longest path from s_a in $subT_1$	(i)
2.	s_a-c_1-c_2-m_2	(ii)
3.	s_a-c_1-c_2-c_3-m_3	(ii)
4.	longest path from s_b in $subT_2$	(i)
5.	s_b-c_2-c_1-m_1	(ii)
6.	s_b-c_2-c_3-m_3	(ii)
7.	longest path from s_c in $subT_2$	(i)
8.	s_c-c_2-c_1-m_1	(ii)
9.	s_c-c_2-c_3-m_3	(ii)

(c) Candidates for s-diameter path of T'

Figure 4.3: Repair Procedure

$subT_1$, $subT_2$ and $subT_3$, and they are connected through their core nodes c_1, c_2 and c_3 (denoted by squares) and reorganized into a new tree T' as shown in Fig. 4.3(b) (there are some possibilities to connect among the core nodes and this figure shows one of them). In this case, the s-diameter of T' is either of (i) the maximum delay from a sender to a node *within the same sub-tree*, or (ii) the maximum delay from a sender to a node on a *different sub-tree*. We enumerate all the candidates for the new diameter path in Fig. 4.3(c) along with their classification of the above type (i) or (ii). We note that m_i and n_i are both ends of the diameter path of $subT_i$ and without loss of generality we assume that delay of path c_i-m_i on the tree, denoted by $L(c_i, m_i)$, is always equal to or larger than $L(c_i, n_i)$.

Obviously, if a path of type (i) (either path 1, 4 or 7 in Fig. 4.3(c)) becomes the s-diameter path of T', the s-diameter is equal to or smaller than that of T. Otherwise, one of the paths of type (ii) becomes the s-diameter path of T'. Here we can say that for any sub-tree $subT_i$, the following inequality always holds;

$$L(s_i, c_i) \leq L(c_i, n_i) \leq L(c_i, m_i) \tag{4.1}$$

where L is the delay of the path between two nodes on the tree[1]. The above inequality

[1]This is obvious since $L(c_i, n_i) + L(c_i, m_i)$ is the diameter (the maximum delay) of $subT_i$.

suggests that for any path of type (ii), the delay from a sender to the core node on the same sub-tree is not larger than the half of the diameter of the sub-tree. Also on another sub-tree, the delay from its core node to an end of the diameter path on the sub-tree is the half of the diameter of the sub-tree. Consequently, the diameter of T' may be equal to or less than the sum of halves of the diameters of the two sub-trees plus the delay between the core nodes. Here, the half of the diameter of a sub-tree of T is always smaller than the half of the diameter of the original tree T. Therefore, this may shorten the s-diameter compared with T in many cases even though it depends on the delay between the core nodes. To make it facilitate to understand, let us compare our strategy with the simple one where we simply connect the neighboring nodes of u (*i.e.* v_1, v_2 and v_3). As the result of this procedure, the new s-diameter is almost equal to that of T with a high possibility, since the maximum delay paths from the neighboring nodes v_1, v_2 and v_3 remain as they are, and they may again be a part of the s-diameter path of the repaired tree.

We note that the above description is for the case that the leaving node u is on the s-diameter path of T. For the case that node u is not on the s-diameter path of T, the s-diameter of T' is not smaller than that of T. This is because the s-diameter path of T is preserved in an isolated sub-tree as it is, and thus it remains in T'. However, we think that the s-diameter of T' is not changed from T in most cases.

As a whole, we can expect the s-diameter to be smaller when nodes leave.

4.3 Design of STS Protocol

4.3.1 Join/Repair Procedure Design

As stated before, we have used the basic ideas of our previous work MODE for the join and repair procedure designs. Since the core node of STS have the similar responsibility as the center node of MODE, we can use the same repair procedure of MODE by replacing its center node with the core node of STS. Therefore, we omit the discussion of the repair procedure design here. However, the join procedure of STS is somewhat different from that of MODE, because of its consideration to the existence of sender nodes. Hence, we discuss the design of join procedure in detail here.

As explained in the previous section, we take a policy to connect a joining new node to an adequate node (called *accepter node*) in the current tree so that the maximum delay from the senders to the new node is minimum. A new node which wants

to join the current tree first sends a *query message* to a well-known node on the tree to ask the address of the *center node* of the tree. The center node of a tree is a node whose maximum delay from the senders is the minimum. Intuitively, the accepter node that makes the maximum delay from the senders to the new node minimum seems to be located near the center node of the current tree. Therefore, we start searching the accepter node from the center node. To do this, in our STS protocol, we assume that each node can know the center node of the current tree (thus any node can be a well-known node). We also assume that each node knows the maximum delays from all the senders. The way of this information collection will be explained later in Section 4.3.2. Once the joining node receives the reply from the well-known node, it sends a *connection request message* to the center node. Then, the center node sends a *connection permission message* to the joining node only if its residual degree is not zero. At the same time, it broadcasts the connection request message to its neighboring nodes of the tree. In response to the reception of the connection request message, each neighboring node acts in the same way as the center node. Therefore the connection request message is delivered on the tree to all the nodes. We note that if there are many nodes on the tree, it is not a good idea for the new joining node to receive a lot of connection permission messages. In such a case, we can give a maximum hop count from the center node, and make only nodes reachable from the center node with at most the maximum hop count reply the connection permission messages. Even if we give such a maximum hop count, the possibility to find the best position (accepter node) is rather high, since usually the accepter node can be chosen from nodes near the center node.

We assume that the connection permission message from each node contains the information about the maximum delay from all the senders to that node. Therefore, the new joining node can know that each node which has sent the connection permission message having at least one residual degree. It can also know the maximum delay from the senders to the node. Then the joining node can measure the delay to the node (this can be measured using *ping* for instance) and choose the accepter node to be connected to, which minimizes the maximum delay from the senders to the new node.

For example, suppose that in Fig. 4.2(b) a new node e wants to join the current tree. The center node of the current tree is either node a or node b. Suppose that node a is chosen as the center node. Then, the node e sends a connection request message to node a and the node a replies the connection permission message which shows that the maximum delay from the senders to node a is 2. Then, node e measures the delay (=2)

from node a and recognizes that the maximum delay from all the sender nodes to node e is 4. At the same time, node a broadcasts the connection request message to nodes b and d along the current tree. Node b replies with a connection permission message which shows that the maximum delay from the senders is 1, node e measures the delay (=2) from node b and node b recognizes that the maximum delay from all the sender nodes to node e is 3. By repeating this message exchange, node e can recognize that it should connect itself to node b (that is, node b is selected as the accepter node).

4.3.2 Information Collection

In STS protocol, all the nodes periodically collects those information required to execute join and repair procedures by message exchange along the current tree. Although this have some similarities to that of MODE, here we discuss it because of some additional parameters caused by the sender nodes.

Hereafter we will explain how the information required by join and repair procedures are collected by collection messages. Here, we show the information that each node x must hold.

- The ID and address of the center node of the current tree.
  The center node is the node whose maximum delay from the senders is minimum.
- The maximum delay from the senders to node x.
  This information is used by the join procedure.
- The sub-tree information of T_yz for every pair of y and z where y is a neighboring node of x and z is a neighboring node of y.
  This information is used by the repair procedure. Note that the sub-tree information of T_yz contains the IDs and network addresses of z, z's neighboring nodes and the core node of T_yz ($core(T_yz)$).

Here, we introduce the details of the sub-tree information of T_xy.

- $dia(T_xy)$: the diameter of T_xy.
- $depthT_xy$: the maximum delay of T_xy from y.
- $H(T_xy)$: the maximum delay path of T_xy from y. The delay of each link on the path is also included.

- $ST(s, T_x y)$: the path from a sender s in $T_x y$ to y. The delay of each link on the path is also included.

We let each node (say v) be responsible for calculating the sub-tree information of $T_u v$ for every its neighbor u. For this purpose, we let the collection message sent from v to u have the following information.

- $dia(T_u v)$, $depthT_u v$, $H(T_u v)$ and $ST(s, T_u v)$ (for each sender s in $T_u v$)
- the sub-tree information of $T_u v$
- the sub-tree information of $T_v w$ for each w (except u) of the neighboring node of v

Now we show that node v can calculate the above information if it receives the collection messages from all the neighboring nodes except u (let W denote the set of neighboring nodes of v except u). First, regarding the parameters $dia(T_u v)$, $depthT_u v$, $H(T_u v)$ and $ST(s, T_u v)$ (for each sender s in $T_u v$), we can define them as follows.

$$
\begin{aligned}
dia(T_u v) &= \max_{w,x,y \in W} \{dia(T_v w),\ depthT_v x + h(v,x) + depthT_v y + h(v,y)\} \\
depthT_u v &= \max_{w \in W} \{depthT_v w + h(v,w)\} \\
H(T_u v) &= [v]@H(T_v w_o) \quad (w_o \in W \text{ where } w_o \text{ maximizes } depthT_u v) \\
ST(s, T_u v) &= [v]@ST(s, T_v w) \quad (\forall w \in W \text{ where } ST(s, T_v w) \text{ is not empty})
\end{aligned}
$$

Note that $h(x, y)$ is the link delay on the tree. The equation for $dia(T_u v)$ comes from the definition of the diameter. noun 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. The others are straightforward.

Secondly, regarding the sub-tree information of $T_u v$, the IDs and network addresses of v and v's neighbors are known by v. Therefore, $core(T_u v)$ can be defined as follows.

$$
core(T_u v) = \begin{cases} core(T_v w) \\ \quad (\text{if } dia(T_u v) = dia(T_v w) \text{ where } w \in W) \\ \\ \text{center of } rev(H(T_v x))@[v]@H(T_v y) \\ \quad (\text{if } dia(T_u v) = depthT_v x + h(v,x) + depthT_v y + h(v,y) \\ \quad \text{where } x, y \in W) \end{cases}
$$

where "rev" is the reverse function of a given path. If the diameter of $T_u v$ is the same as that of a sub-tree $T_v w$, the corresponding $core(T_u v)$ is the same as that of $T_v w$. On the

other hand, if the concatenation of two maximum delay paths from v becomes $T_u v$'s diameter path, we must select a node in the center of the diameter path as $core(T_u v)$. Therefore, the sub-tree information of $T_u v$ can also be known.

Thirdly, the sub-tree information of $T_v w$ is included in the collection message from w.

From the above, we have proved that each node v can calculate the content of the collection message to be sent to node u. Then, after receiving the collection messages from all the neighbors, node u can know the information included in the collection messages. Assuming those information, we show that any node, say x, can calculate the information that node x must hold, listed previously in this section. The center node and maximum delay from the senders can be calculated by all the $ST(s, (T_x*))$ included in the received collection messages. Also the sub-tree information of $T_y z$ is directly included in the collection message from y to x. Consequently, we have proved that every node can obtain the required information after the collection phase.

4.4 Implementation of STS Protocol as a Java Middleware

We have implemented our protocol as a Java based middleware called STS/J. This middleware constructs and maintains an overlay multicast tree based on STS protocol.

4.4.1 Overall Architecture

Our middleware is composed of two layers as shown in Fig. 4.4. The underlying layer is Tree Controlling Layer that controls the tree structure according to STS protocol. The overlaying layer is Media Streaming Layer that provides rich functions for transmission of streaming data. The underlying layer is implemented as a class named STS that has some instance methods. To use this middleware, first, we must choose an initial degree bound d and make an instance of class STS by passing d. The degree bound of each node will be adjusted by our middleware automatically depending on the capability of the node. Then we can activate our middleware by calling method $join$ with a parameter $SocketAddress$ (the pair of an IP address and a port number) that specifies the well-known node of the tree. When a STS object is initiated with the address of its well-known node, the object will connect to the tree according to STS protocol. We can detach a node from the tree by calling *leave* method. Multicasting of

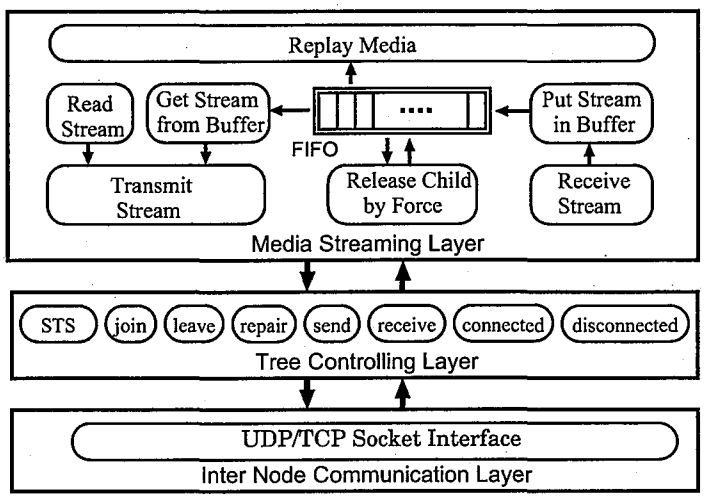
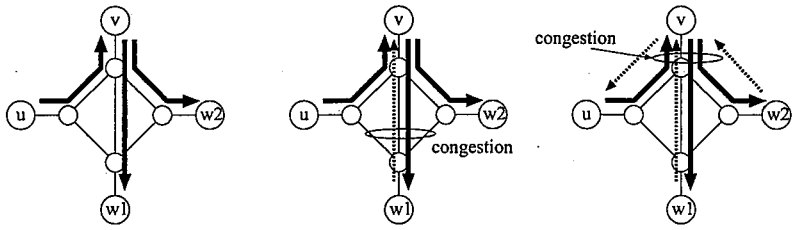


Figure 4.4: System Architecture

messages can be carried out by calling *send* method.

We provide three callback methods *receive*, *connected* and *disconnected*, and we can implement applications by overriding them. The *receive* callback is called when a message transmitted by *send* method arrives. If *receive* is not overridden, the node becomes a relay node that only forwards the messages and does not receive any message. An original transmission process for multimedia streams can be implemented in the *receive* callback by using QoS control mechanisms such as transcoding. The *connected* callback is called when a node is connected to the tree, and *disconnected* callback is called when the tree is broken. These callbacks are prepared for notification to the users. So we can use this middleware without overriding them because the necessary processes are carried out inside of the middleware automatically. If we want to implement a node that transcodes the forwarding streams based on some QoS mechanisms, it can be done by overriding $forward$ method by alternative forwarding method.

We have implemented the overlaying layer as a class $STSstream$. To use this layer, we must make a STS object, and then initiate a $STSstream$ object with the STS object. By specifying a stream which is used to transmit data by $setOutputStream$ method, we can start multicasting data through the stream to all the nodes of the tree that the STS object joins. The transmitted data can be received from a stream which



(a) Physical network and overlay tree.

(b) Congestion occurs on a remote link between nodes v and $w1$.

(c) Congestion occurs on the local link of node v.

Figure 4.5: Adaptation Mechanism

is given by $getInputStream$ method. This layer controls the underlying STS objects according to the following QoS mechanisms in order to keep an efficient tree structure for multicasting.

4.4.2 Adaptation Mechanism for Media Streaming

Theoretically, we have shown importance of s-DBMDT for interactive multimedia applications in previous sections. Here, we focus on practical aspects of multimedia streaming on a tree. Practically, it is difficult to determine an appropriate degree for each node. It is theoretically simple, since usually an end host has only one network interface, and the overlay links attached to the host uses the interfaces. Therefore, the upper bound of the degree bound $d_{max}(v)$ of node v is determined by $d_{max}(v) \leq \frac{N}{\sum_{s \in S} B_s}$ where N is the bandwidth of the network interface and B_s is the bitrate transmitted from a sender s. However, the actual bandwidth of network interface, especially wireless network interface, changes from time to time. Therefore, the degree bound should be adapted according to the network status.

Here, we adopt the following policy for each node, say v. Here we denote the set of the neighboring nodes of v by W. We also denote the neighboring node which sends a stream to v by u. Thus node v relays the stream to the nodes in $W - \{u\}$ (Fig. 4.5(a)). Our implementation uses RTP and RTCP, and if a node detects loss or jitter of packets, it sends receiver reports to its upstream. In Fig. 4.5, streams are represented as thick arrows, while RTCP reports are represented as dotted arrows. In the figure, the underlying network is shown where small circles represent physical routers and big ones represent end hosts.

- If node v receives an RTCP receiver report from only one node (or some nodes) $w \in W - \{u\}$, node v determines that network congestion happens not on the local link (*i.e.* network interface) but on a remote link on the unicast path between v and w (Fig. 4.5(b)). In this case, node v sends *compulsory leave message* to node w to let it leave and rejoin the tree.

- If node v receives an RTCP receiver report from each node in $w \in W - \{u\}$ and also node v sends an RTCP receiver report to node u, node v determines that network congestion happens on the local link (Fig. 4.5(c)). In this case, node v ignores the compulsory leave message from node u, and sends compulsory leave messages to some nodes in $w \in W - \{u\}$ to let them leave and rejoin the tree. This is done to dissolve the congestion on the local link of node v by decreasing its current degree. After that, node v sets its degree bound $d_{max}(v)$ to the adjusted degree to prevent itself from accepting other neighbors.

- If node v continues stable states for a while, it increments its degree bound.

4.5 Simulation Experiments

In this section, we describe our performance evaluation on ns-2 simulator.

4.5.1 Simulation Setup

We have implemented our STS protocol on ns-2 to evaluate the enhancement against our previous work MODE[12]. In our experiments, networks with about 400 physical nodes have been generated and used as underlying networks. We have selected 200 nodes, including both wireless and wired nodes, as overlay participant nodes. In the simulation, we have set the initial end-to-end delay (overlay link delay) to vary between 10ms to 200ms for both wired and wireless nodes, while setting the wireless nodes to change their end-to-end delay up to 300ms during the simulation.

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. The initial degree bound was set to 5 for all the nodes. The scenario is as follows. (i) The session period is 300 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 270 seconds, additional joins are processed. Also some existing nodes leave the session. The collection phases starts at 30, 90, 150, 210 and 270 seconds successively. (v) After 270 seconds, no node joins and about 40 nodes leave the session.

In Fig. 4.6, Fig. 4.7 and Fig. 4.8, 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.

4.5.2 Experimental Results

Diameter and Sender-Dependent Diameter: We have measured (a) the diameters and (b) sender-dependent diameters (s-diameters) at every one second for STS and MODE. According to the goals of those two protocols, we can expect that STS could achieve smaller s-diameters, but a bit larger diameters than MODE. Fig. 4.6 and Fig. 4.7 shows the results. We can figure out that the s-diameter of STS is smaller (Fig. 4.7). On the other hand the diameter of STS remains larger than MODE according to Fig. 4.6.

Note that the diameter is not dominant on the streaming as long as one of the predefined sources (defined at the tree construction stage) play the role of streaming source. However, the nodes other than the predefined sources may also become the streaming source. For that, STS periodically refreshes the information of the tree, and after the information is refreshed, the succeeding join/repair operations are done according to the new sources. However, for the duration before the refreshment, some validity should also be given to the diameter. The results shown in Fig. 4.6 states that the diameter of STS has not much diverged from that of MODE, which means that the value is still small enough to enable the nodes other than the predefined sources being the streaming sources as well. Also this diameter increment here can be considered reasonable enough by taking the s-diameter reduction advantage into account.

We have also measured the average diameters and s-diameters (in milliseconds) for STS and MODE after performing simulations for 10 different sessions, where each follows the above scenario. The results are shown in Table 4.1. According to those results, we can again say that STS can support multimedia applications better than MODE.

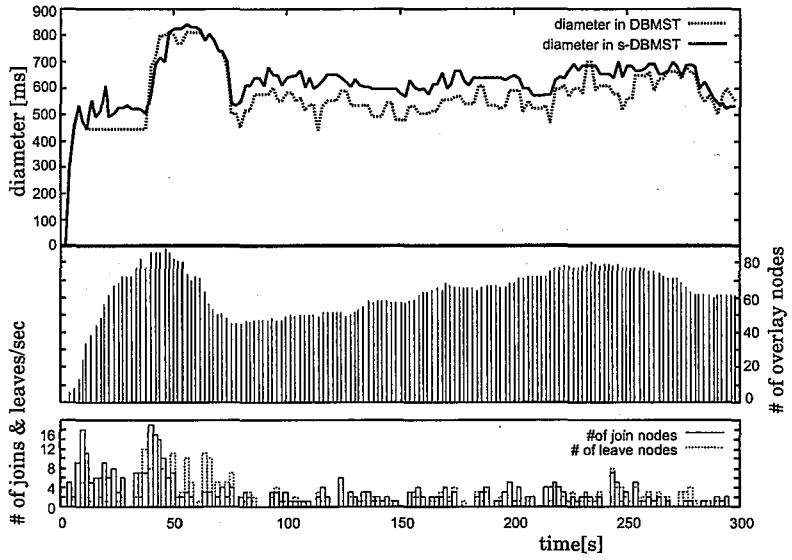


Figure 4.6: Dynamics of Diameters in Simulation Experiments

Table 4.1: Average Diameter and S-diameter of STS and MODE

protocol	diameter[ms]	s-diameter[ms]
MODE	585	480
STS	651	364

Control Traffic: We have measured the control traffic flows on the entire tree for STS to see if it is small enough compared with streaming bandwidth. The result is shown in Fig.4.8. The highest traffic amount has been generated around 30 seconds (around the first collection phase) as the number of nodes has reached to top. Even taking this peak value (350kbit) together with the number of nodes in the session at that time (90 nodes approximately), the average traffic amount on a single node can be calculated as 4kbit/sec. We can say this value (4kbps/node as maximum) is small enough for streaming applications which usually consume several hundreds of kbps.

Join/Repair Procedure Overhead: We have measured the time required for the two procedures explained in Section 4.3 to see if that is reasonable enough for multimedia streaming. Table 4.2 shows the results. According to the results, the time required

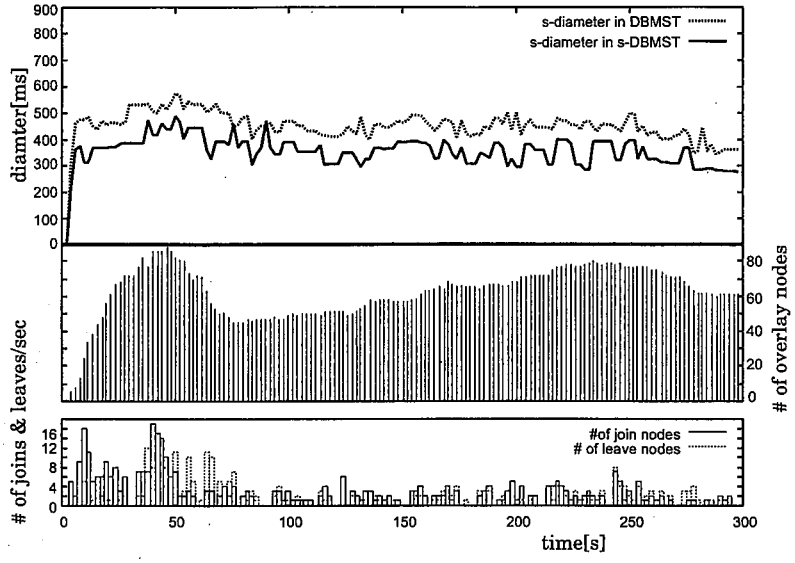


Figure 4.7: Dynamics of Sender Dependant Diameters (S-diameters) in Simulation Experiments

Table 4.2: Average Time Required for Join and Repair Procedures

Join [ms]	Repair [ms]
930	790

for restoration of isolated trees (repair time) remains less than 1 second, which can be considered small enough for multimedia streaming. The time required for the join procedure (join time) here is larger than the repair time. Here we have set the connection permission message timeout (the time each joining node waits to receive connection permission messages before it selects the best position to connect) large enough (0.6[s]) to receive as more as connection permission messages. So the join time holds a larger value (0.93[s]), but this value is considered reasonable enough as time required for bootstrapping.

4.6 Experiments on Real Networks

In this section, experiments using the implementation on real networks and those results are described.

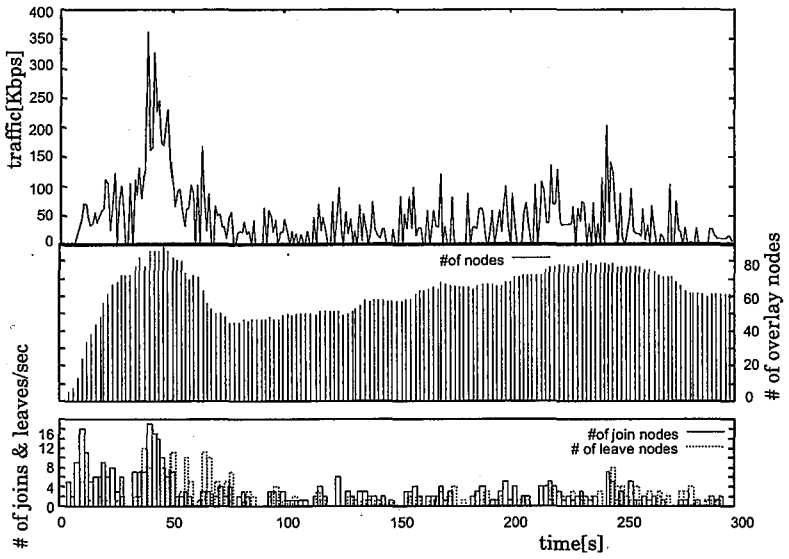


Figure 4.8: Control Traffic (Total kbits on tree per each second)

4.6.1 Experiment Settings

We have used 8 machines placed in the same LAN (100Base-TX) and run 4-6 processes on each machine to emulate 40 user nodes. Here, we made each process keep the packets for a certain time period before forwarding, if the forwarding target is located in the same machine where that process runs. In this way, we could prevent the link-delay between the nodes located in the same machine from being too small. The scenario is as follows. All nodes join the tree before streaming video so that they can be ready to receive the initial frame of the stream. It includes some information to decode and play video such as its resolution and frame rate. Throughout the session, the streaming-source changes it's position through the pre-defined source nodes. During the streaming, 3-5 nodes are set to leave in every interval between collection phases. The period is of 60 seconds. Here the link delays vary from 10[ms] to 100[ms] and the initial degree bound for each node is 4. Each node's degree bound is dynamically changed to enhance the media delivery performance on the tree as described in Section 4.4. And also, we set STS to select its predefined streaming sources increasingly with the total node count in a manner to make the sources, which includes the initial node as well, occupy 20% of the entire nodes.

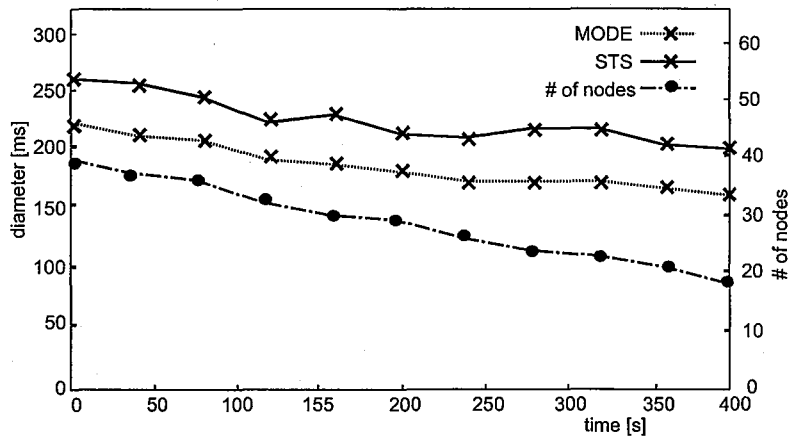


Figure 4.9: Dynamics of Diameters in Real Network Experiments

4.6.2 Experimental Results

Diameters and S-diameters: We have measured the dynamics of diameters and s-diameters. We have used the same scenario for 10 cases each of which has different locations for pre-defined source nodes, and have measured the average for every 25 seconds. The measured variance is shown in Fig. 4.9 and Fig. 4.10. We can see that the diameter of STS is held higher than that of MODE (Fig. 4.9). But s-diameter, which counts more in multimedia streaming, is held smaller in STS according to the results shown in Fig. 4.10. These results state that STS is better for multimedia streaming in applications like video conferences with a certain number of key-persons, who address the rest of the audience for the most part of the session.

Degree Adaptation: We have checked whether the degree adjustment strategy described in Section 4.4.2 works well. For that we have set the 2 scenarios shown in Fig. 4.11 which illustrates a part of the overlay nodes (machines) we used and the 802.11b wireless links. Fig. 4.11-(a) represents a case, where a congestion occurs on a unicast link between 2 nodes, and Fig. 4.11-(b) is another case, where a congestion occurs on the physical network link connected to a node. For each of these cases we have used a 256 Kbps media stream as the multicast media and a wireless node to make the congestions. And also the lower threshold of the bit-rate, below which a node detects that a network congestion has occurred, has been set to -20% (205 Kbps) of the stream's

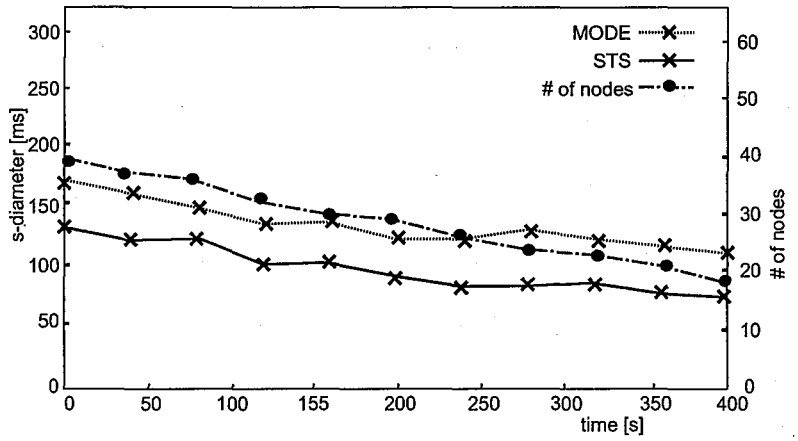


Figure 4.10: Dynamics of Sender Dependant Diameters (S-Daimeters) in Real Network Experiments

real bit-rate.

In the first case, we have generated some additional traffic across the link between $pc2$ and $pc4$. Then we could see node $pc2$ has detected the congestion on the down link to $pc4$, where it has sent a compulsory-leave message to $pc4$. Then $pc4$ successfully has left and rejoined the session. In the second case, we have generated a separate process requires some additional traffic, which is enough to make a congestion on the physical network link, on $pc2$. Here we have found that nodes $pc1$ and $pc2$ have detected that downward links are congested, after receiving RTCP reports from their child nodes. Then $pc2$ has sent a compulsory-leave message to $pc3$ and $pc4$ (these are randomly selected to occupy around 50% of the total connection count, in our experiments). Here, the forcibly disconnected node, which is subjected to follow the ordinary join procedure, has rejoined to the tree after 1.28 second average value. And the isolated sub-tree, which was located under the forcibly disconnected node, has reconnected to the tree after average 0.43 milliseconds following the ordinary repair procedure.

Time Required for Join/Repair Procedures: We have measured the time required to complete the join and repair procedures. Their average and maximum values are shown in Table 4.3. We can confirm that the time required for a repair procedure is small even in the worst case. Here, the time for join procedures remains higher. This is because we have made each joining node wait at least 1 second before making a link

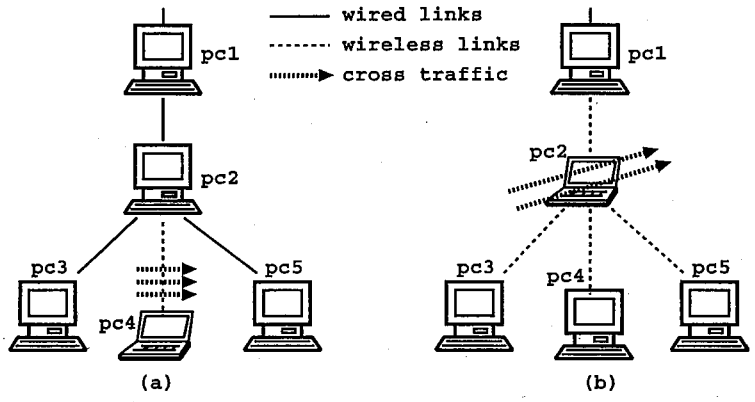


Figure 4.11: Degree Adaptation Experiment

Table 4.3: Time Required for Join and Repair Procedures

procedure	average[ms]	worst[ms]
join	1212	1415
repair	318	734

to the tree. It allows a node to receive as many permission messages as possible (see Section 4.3.1). This contributes to let the joining node connect to more closer node to the center node.

Table 4.4: Packet Loss Ratio Against a Single Repair Procedure

bit-rate [bps]	data loss(KBytes)/restoration
64[K]	5
128[K]	11
256[K]	24
512[K]	54
1024[k]	123

Also some packets may be lost in the restoration processes for nodes' disappearances. We have measured this loss using several video streams of different qualities. The average data loss per a single repair procedure is shown in Table 4.4 against the various video bit-rates.

We can see that the loss is low enough. Considering the fact that the repair procedure completes less than in one second, we do not have serious distortion in playback of received video. The packet loss ratio has increased with the bit-rate as we can predict.

Currently STS nodes convey each multimedia data packet to the corresponding neighbor nodes without using the cache (the ring buffer) to support the real-time video conferences. But, readers may understand that STS can be simply modify to adapt applications requiring a higher quality playback (*i.e.*lower jitter, less stream loss) by using the ring buffer, where the jitter can be made lower and the data loss in sub-tree restoration process can be get to zero or to a negligible value.

4.7 Consideration of Heterogeneous Users

With the recent rapid deployment of high-spec mobile devices, the end-users are shifting from the conventional desktop PCs to various hand-held devices, such as mobile phones or PDAs. Therefore, some nodes participate in a ALM application may be mobile terminals, and those hosts may not be stable due to limitation of batteries, computing and communication capabilities and so on. Hence, they may not be stable to relay data packets. Moreover, they may use wireless links and thus delay of the overlay links connected to those nodes may not be stable also. In such an environment, overlay multicast should be constructed carefully and dynamically to prevent those hosts from staying at critical positions that affect the diameter and stability of the overlay multicast.

The protocols, MODE in our previous work and OMNI[7] have presented distributed solutions for the requirements (i) and (ii). However, as far as we know, none of the existing work, including MODE and OMNI, has not dealt with the third issue, which is a very important issue to realize seamless communication between non-mobile and mobile nodes in group communication.

In this chapter, we present a protocol called MODE-for-mobile (MODE-m in short). MODE-m is extended from MODE protocol to incorporate instable and less powerful hosts such as mobile nodes. The original MODE aims at constructing a *Degree-Bounded Minimum Diameter Tree (DBMDT)* in a distributed manner. MODE-m has the same goal as MODE, however supporting mobile nodes is a new feature to handle group communication with diverse hosts. In MODE-m, if mobile nodes occupy intermediate positions of the current tree and the diameter becomes large, such nodes are set to leave and re-join the tree so that the diameter of the tree can be held short. Our experimental results have shown that this feature is very effective to keep the diameter as small as possible under the existence of mobile nodes.

4.8 Motivation

MODE-m described this chapter can adaptively determine the positions of mobile nodes to *prevent these incapable nodes from affecting the diameters*. We illustrate how MODE-m works in comparison with MODE in Fig.4.12. For simplicity, in the following figures, we only illustrate overlay networks, and physical networks are omitted. Also an integer on an overlay link represents its delay, and description of delay

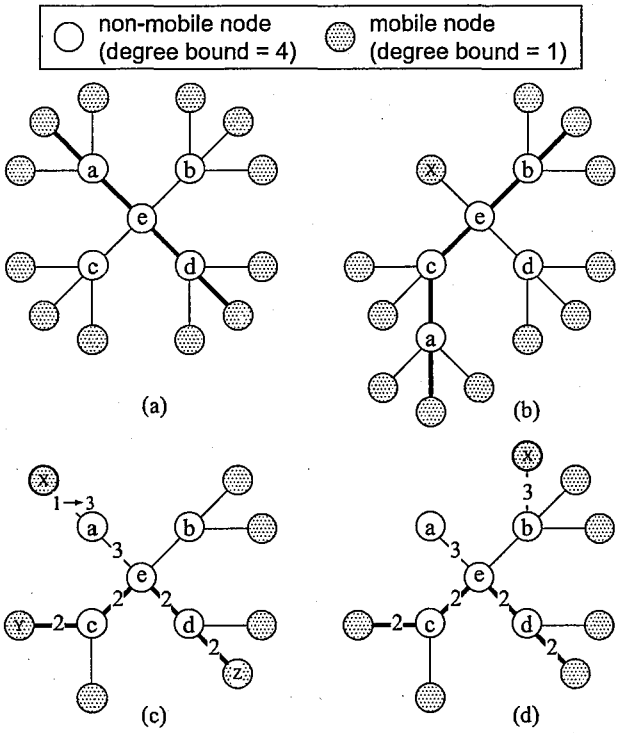


Figure 4.12: Concept of MODE-m

"1" is omitted. Finally, diameter paths are denoted by thick lines. The essential difference between MODE and MODE-m is illustrated in Fig.4.12(a) and Fig.4.12(b). In Fig.4.12(a), non-mobile nodes (white circles) with degree four are connected near the center (node e in this case) of the tree, and mobile nodes (meshed circles) with degree one (*i.e.* they never relay packets) are located in the leaf positions. On the other hand, in Fig.4.12(b), a mobile node X is connected to node e, thus node a is connected to node c and consequently they form a longer diameter path. The case of Fig.4.12(b) will happen in MODE, in case of node X's prior arrival to node a in the assumed incremental joining to the current tree, since the tree optimization is done only when a node in the tree leaves. Therefore, once a mobile node occupies a position close to the center, it continues to keep the position without providing degrees. This results in pushing the following joining nodes like node a away from the center and making the diameter longer. In MODE-m, if a non-mobile node with higher degree bound

joins the current tree, our protocol makes it preempt the position of a mobile node near the center by letting the mobile node leave and re-join the tree. Thus we can expect that a tree like Fig.4.12(a), which is well organized and has a shorter diameter, will be formed eventually. Another feature is that mobile nodes adaptively move if they know that they become to form diameter paths due to variation of overlay link delay. For example, in Fig.4.12(c), the diameter path is the path between node Y and Z. Then let us assume that the delay of overlay link X-a changes to 3 and the path X-Z becomes the diameter path. In the original MODE, each node knows the current diameter and the "height" of the tree rooted at the neighbor of the node by periodical collection of diameter information[2]. Using this information, node X can know whether the variation of delay makes the path be the diameter path or not. In MODE-m, if it is true, node X spontaneously leaves and re-joins the tree to find a better position (Fig.4.12(d)). If this variation is caused by the "last one hop" wireless link, the overlay link delay between X and the new neighbor may be the same as before. However, the diameter, and thus the session delay, will be improved as in Fig.4.12(d).

4.9 Overview of MODE-m

In this chapter, we assume that nodes are classified into two types, mobile nodes and non-mobile nodes. Also we assume that the degree bounds of the mobile nodes are all one and the delay of overlay links connected to mobile nodes may change from time to time.

4.9.1 Outline of MODE for Mobile

The main effort of our research is to present a new DBMDT constructing protocol to support *mobile nodes*. The basic idea is inspired from the protocol MODE, which is one of our prior works. Mobile nodes we discuss here are nodes with low-bandwidth and less capability of computing, which are connected via wireless links to wired networks. Therefore, we set mobile nodes' degree bounds to only one (*i.e.* they never relay packets) and treat them in a special way where we always keep them as leaf nodes of the tree.

In MODE-m, we follow MODE's concept of repeating two phases, the *collection phase* and the *normal phase*, alternately. The protocol carries out a certain informa-

[2]The information is aggregated at each node and the amount of information is very small in each collection packet. This is one of the key advantages of MODE.

tion gathering in the collection phase and this information is used to construct, refine and repair the spanning tree in the normal phase which starts right after that collection phase. The normal phase stands for the time-period between two collection phases. MODE-m has three procedures, *join procedure*, *refresh procedure* and *leave procedure*, in addition to the information collection. These are responsible for adding newly joining nodes, stabilizing the diameter fluctuation caused by mobile nodes and repairing isolated trees caused by nodes' disappearance, respectively.

As stated before we have used the basic ideas of our previous work MODE for the join, repair and information collection procedures. Since the center node of MODE-m have the similar responsibility as the center node of MODE, we can use the same repair procedure of MODE except for the following behavior. The mobile nodes never become repair masters (refer Fig. 3.5) neither repair initiators as they reside as leaves. However, they may become repair sub-initiators or repair sub-masters only for the subtrees those consist of a single mobile node. In this case, the repair procedure may not get completed due to the message loss caused by the mobile node link. As a result isolated subtrees may generate. Our protocol overcomes these isolations by making the root nodes of the isolated subtrees perform a usual join attempt after a certain timeout. Again we can use the same information collection procedure as we utilize the same information. Therefore, we omit the discussion of the repair procedure and information collection here. However, the join procedure of MODE-m is somewhat different from that of MODE, because of its consideration to the existence of mobile nodes. Hence, we discuss the design of join procedure in detail here.

4.9.2 Join Procedure

In MODE-m, the basic policy to accept a new node is to connect the new node to a existing tree's node with more than one residual degree, which places the new node closest to the center node. This basic approach provides quite reasonable diameters in an incremental way. However, as we stated in Section 4.8, this approach fails to make trees with reasonable diameters when mobile nodes' participation takes place. So we move the mobile nodes so that they become leaf nodes in the following manner. Every non-mobile node replaces one of its mobile node neighbors (say m) with the newly joining non-mobile node by forcibly disconnecting m, only if that makes the new node closest to the center node. Here the node m is selected randomly, and m is subjected to *rejoin* the tree in the same way as a newly joining mobile node. Note that newly

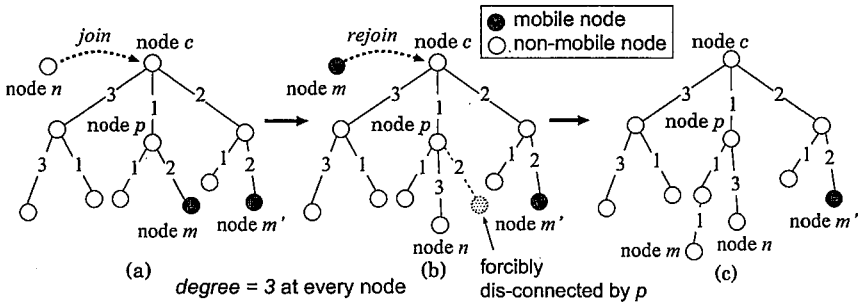


Figure 4.13: Mobile Node Rejoin in Join Procedure

joining mobile nodes cannot replace them with the existing mobile nodes. The detailed join procedure is as follows.

A new node which wants to join the current tree first sends a query message to the root node [3] to ask the address of the center node. In MODE-m, any node can calculate the center node of the tree using the sub-tree information described in the previous subsection. So the root node can send the center node's address to the new node. In case that the root node does not know the center node (*i.e.* before the first collection phase has been completed) or the center node has already disappeared, the root node sends the address of itself instead.

Once the joining node receives the reply from the root node, it sends a connection request message to the center node (Fig.4.13(a)). The center node (say c), which receives the connection request message, sends a connection permission message to the joining node if at least one of the following two conditions, (i) c has more than one residual degree or (ii) the joining node is a non-mobile node and c has at least one mobile node as a neighbor, is satisfied. At the same time, it broadcasts the connection request message to its neighbors[4]. The neighbors also treat the connection request message in the same way. Note that these nodes send no message to the joining node if they satisfy neither of the conditions (i) and (ii) stated earlier. Here the delay from the center node is also added to the connection request message before it is broadcast to the neighboring nodes. So every node which sends a connection permission message

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

[4]Here we can set a suitable forwarding count limit to prevent the joining node from receiving a huge number of connection permission messages.

to the joining node also includes the delay from the center node to the node itself. Then the joining node uses these values and the delay to each node which has sent a connection permission message (this can be measured using *ping* for instance) to select the node (say p) which can place this node closest to the center node. Then it establishes an overlay link with the node p. If this overlay link violates the degree constraint at p (Fig.4.13(b)), that is p has accepted the new node according to the condition (ii), then the node p forces one of its neighboring mobile nodes to rejoin the tree (Fig.4.13(b)). As a result of these rejoin processes, the mobile nodes shift themselves from positions close to the center node to leaves of the tree.

4.9.3 Refresh Procedure

Refresh procedure is applied only to mobile nodes, to prevent the diameter from being affected by mobile nodes' delay fluctuation. Mobile nodes become leaf nodes according to the above join procedure and that helps to construct a spanning tree with a shorter diameter. However, the diameter path's both ends may be occupied with mobile nodes in many cases. Therefore, the diameter becomes sensitive to the mobile nodes' delay fluctuation. To avoid this phenomenon we make mobile nodes rejoin the tree if its wireless characteristics result in a longer diameter. We name this process *refresh* [5]. Note that mobile nodes can detect the change of the delay to its neighbor according to the assumption G1.

A mobile node m performs a refresh procedure if the following statement is true. Here n denotes the neighboring node of m.

$$c(m,n) + depthT_mn > dia_{now} + \alpha$$

Here, $c(m,n)$ denotes the delay of the link from m to n, $depthT_mn$ denotes the longest depth of the subtree T_mn and dia_{now} denotes the current diameter of the tree. Here α regulates the refresh frequency. In other words the refresh procedure does not take place if the new diameter exceeds the previous one by not more than α. Without this the mobile nodes may oscillate in the tree. Fig.4.14 illustrates this refresh strategy.

Fig.4.14(a) shows a case that a mobile node(m) finds that the above condition is true after detecting the change of the delay to its peer ($c(m,n)$). Then m rejoins the tree as shown in the Fig.4.14(b) to prevent the diameter from increasing. Note that

[5] We assume that the link delay variation of non-mobile nodes are negligible, hence non-mobile nodes are not concerned for the refresh procedure.

m keeps checking the above condition whenever it detects a new delay, despite its previous refresh tasks. To avoid infinite refresh tasks, we have set the mobile nodes not to perform a second refresh procedure, if it got connected to the same peer after rejoining the tree.

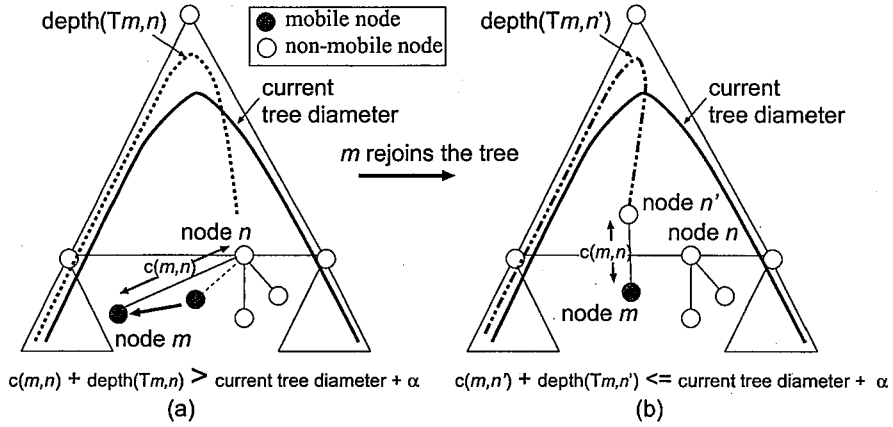


Figure 4.14: Mobile Node Refresh Strategy

4.10 Performance Evaluation

4.10.1 Simulation Setup

We have implemented our MODE-m protocol on ns-2 to evaluate the enhancement against MODE. In our experiments, networks with about 400 physical nodes have been generated and used as underlying networks. We have selected 200 nodes, including both mobile and non-mobile nodes, as overlay participant nodes. Here we define the mobile node occupancy (as a percentage of the entire nodes) as an evaluation parameter m. We have set mobile nodes change their end-to-end delay (overlay link delay) randomly from 50ms to 300ms, hence emulated random movement during the simulation. And constant delays vary from 10ms to 200ms have been set for the overlay links between non-mobile nodes.

Considering practical situations, we have prepared the following scenario that simulates a real-time session in collaborative applications such as video-based meetings or groupwares. Note that we have set the interval between collection phases to 30 seconds. The degree bound on each non-mobile node was set to a randomly selected

constant value, which ranges from 3 to 7, while that of each mobile node was set to 1. The scenario is as follows. (i) The session period is 300 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 250 seconds, additional joins are processed. Also some existing nodes leave the session. The collection phases take place at 30, 60, 90, 120, 150, 180, 210, 240, 270 and 300 seconds successively. (v) Mobile nodes are set to refresh their delay in each 5 seconds. And the value α (see Section 4.9.3) is set to 50[ms]. (vi) After 250 seconds, no join takes place and about 40 nodes leave the session.

We have evaluated MODE-m following the above scenario setting m (ratio of mobile nodes to all the nodes) to 5%, 10%, 20%, 30% and 40%. Here, we have limited the upper margin of m to 40% considering the fact that construction of DBMDT might be impossible with a larger percentage of mobile nodes where $d_{max} = 1$ [6]. In Fig.4.15 and Fig.4.16, the number of nodes on the tree at every second together with the numbers of join/leave operations are shown, to make it facilitate to see the dynamics of the metrics according to the scenario.

4.10.2 Experimental Results

[Diameter Against the # of Nodes] We have measured the diameters of the trees at every one second for MODE-m and MODE for various m's. The result for $m = 30\%$ is shown in Fig.4.15.

Obviously MODE-m could archive a shorter diameter. Especially in the period with higher mobile node occupancy (within 0[ms] to 75[ms] in Fig.4.15), MODE's performance has become poorer. Here the diameter difference of MODE and MODE-m in the time period of 0[ms] to 30[ms] in Fig.4.15, shows the validity of the improvement done to the join procedure, while the rest part of the session shows the performance gained by the mobile node specific refresh procedure. Considering both results we can say that MODE-m well supports mobile nodes.

[Average Diameter and Rejoin-cost] We have measured the average diameters (in milli seconds) for MODE-m and MODE after performing simulations for ten different sessions, where each follows the above scenario. And also we calculated the *rejoin-cost* for the MODE-m in each session and calculated the average value for the ten

[6] Readers may note that constructing of DBMDT will become possible with the existence of powerful mobile nodes (nodes where $d_{max} > 1$).

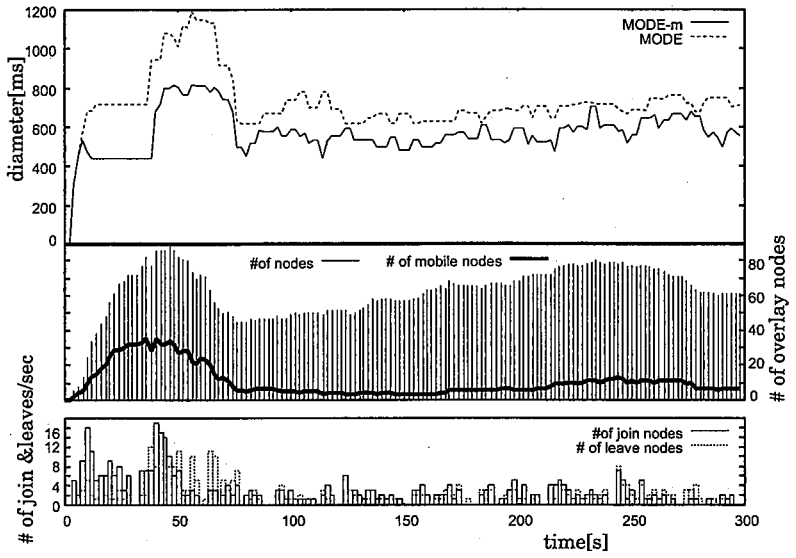


Figure 4.15: Dynamics of MODE-m Diameters

sessions. *Rejoin-cost* is the sum of the number of overlay links which are established and disconnected due to the rejoin processes during the join and refresh procedures over the session. Note that the rejoin-cost becomes two times of the number of rejoin processes applied. The results for different values of m's are shown in Table. 4.5.

Table 4.5: Comparison of Diameter [ms] and Rejoin Cost [number of attached/detached links] of Mobile Nodes

m		5%	10%	20%	30%	40%
MODE -m	diameter	467	513	565	585	646
	rejoin-cost	32	64	162	197	222
MODE	diameter	494	569	633	736	801

The result shows that the performance of MODE-m increases with the growth of the mobile node occupancy. At the same time we can note that the rejoin-cost also keeps increasing with the mobile node occupancy.

The rejoin-cost discussed here reflects the disconnecting frequency of mobile nodes. Considering that the mobile nodes always resides as leaf nodes, we can understand the fact that non-mobile nodes remain unharmed in this regard. This implies that the ses-

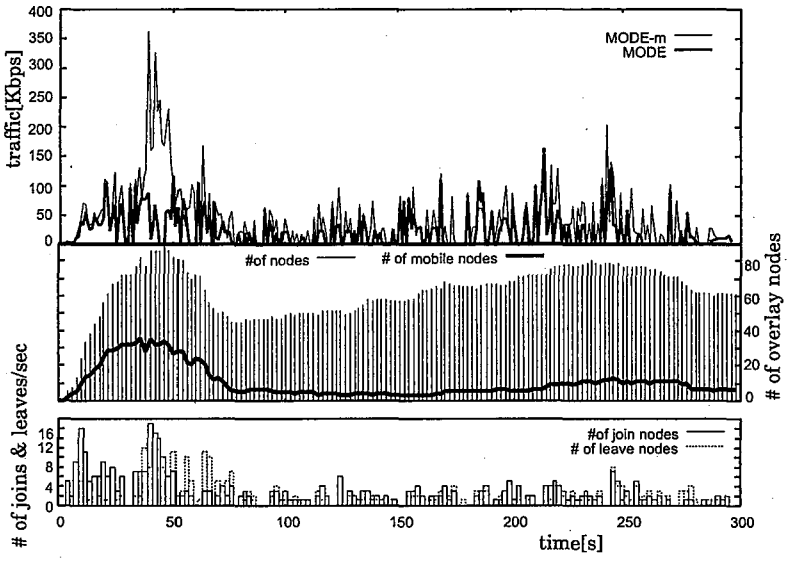


Figure 4.16: Control Traffic of MODE-m

sion remains consistent for the rest of nodes, even if the rejoin-cost is considerably large. However, we can say the rejoin-cost is reasonable enough considering the diameter reduction gained compared to MODE, by these rejoin processes. Furthermore, the rejoin-cost can be reduced by increasing the value of α (see Section 4.9.3), though this might increase the diameter, which is in a trade-off with the rejoin-cost.

[Traffic] We have measured the control traffic flows on the entire tree for the MODE-m and MODE. The result is shown in Fig.4.16. The increased traffic amount in MODE-m is mainly the traffic due to rejoin processes. And the highest traffic amount has been generated around 30[ms] as the number of mobile nodes has reached to top. Taking this peak value, 350[Kbps], together with the number of nodes in the session at that time (90 nodes approximately), the average traffic amount on a single node can be calculated as 4[Kbps]. We can say this value is small enough for real world applications.

[Time Required for Procedure Execution] Finally, we have measured *join-time* and *refresh-time*, the time required to execute join and refresh procedures successively. Note that here refresh procedure does not count for MODE, and the comparison of the time required to repair is omitted here, since repair procedure is common to MODE and MODE-m. Here, the *join-time* of a node n includes n's join time plus the time needed

Table 4.6: Comparison of the Time Required for Join and Refresh

	join-time[s]	refresh-time[s]
MODE-m	0.93	0.79
MODE	0.71	-

for the mobile node, which have been forcibly disconnected due to the join procedure of n, to rejoin the tree. The expected average values for join-time and refresh-time when $m = 30$ are shown in Table. 4.6. According to those results both the join-time and the refresh-time hold values less than 1 second, which can be considered small enough for real applications. Note that the values in Table. 4.6 remain almost same for different values of m as well, according to the nature of the protocol.

4.11 Concluding Remarks

In this chapter, first we have stated the design and implementation of an overlay multicast protocol for interactive multimedia applications including media streaming. The protocol is called Shared Tree Streaming (STS) protocol that constructs a shared tree called *s-DBMDT* (sender-dependent Degree-Bounded Minimum Diameter Tree) as an overlay network that involves all the participants of the application. We believe that this is the first approach that defines s-DBMDT construction problem and presents a distributed protocol for the purpose. Our implementation is done in Java where some adaptation mechanisms are incorporated for media streaming. Our performance evaluation is based on experiments in both simulated networks and real networks that strongly shows the efficiency an usefulness of STS protocol. Evaluating STS protocol on large-scale, real environments such as PlanetLab is part of our future work.

Second, we have proposed a new overlay multicast protocol called MODE-m that constructs a degree-bounded minimum diameter tree, supporting mobile hosts. MODE-m is considerably enhanced from MODE so that it can achieve reasonable diameter under the existence of heterogeneous hosts. The experimental results have shown the advantage of MODE-m compared with MODE.

Chapter 5

Decentralized Construction of Stability Oriented Spanning Trees in Multiple Source Context

5.1 Introduction

Application Layer Multicast (ALM) 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 distance learning, internet games, 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 chapter 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* [31]. 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 [31]. This property helps to predict nodes remaining lifetime to make the multicast topology more robust to node departures.

In this chapter, 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 chapter 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.

The rest of the chapter is structured as follows. Section 5.2 analyzes and describes the problem. Section 5.3 describes the proposed decentralized solutions. Section 5.4 describes the simulation and PlanetLab experiments. Section 5.5 concludes the chapter.

5.2 Problem analysis

5.2.1 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.

$$stab_T = \sum_{(s,r) \in S \times R} stab_T(s,r)$$

$$stab_T(s,r) = \begin{cases} 1 & (s = r) \\ sc(r') \cdot stab_T(s,r') & (s \neq r) \end{cases}$$

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 [32], can be transformed to our ms-DDBMSST decision problem in polynomial time.

Further, *Degree Bounded Maximum Stability Spanning Tree (DBMSST)* construction problem has also been addressed in few previous lifetime-aware schemes ([26], [27], [28]). 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 5.3 consists of

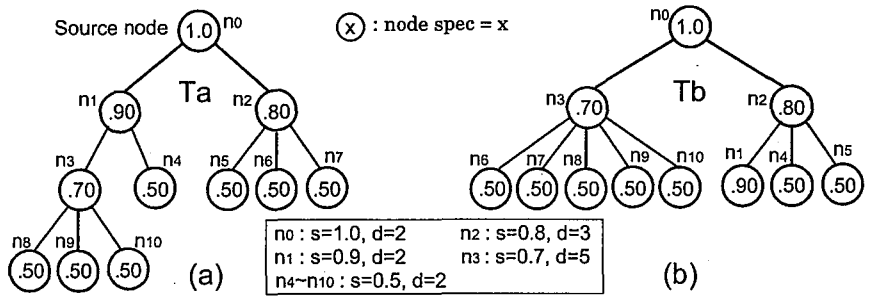


Figure 5.1: Two Simple Methods that Construct DBMSST: (a) stability-first (s-first) and (b) stability-degree product-first ($s \cdot d$-first)

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.

5.2.2 Learning from DBMSST

Existing approaches

Fig. 5.1 illustrates two simple methods of constructing DBMSST presented in [27]. The stability coefficients and degrees are given in the box of Fig. 5.1. In Fig. 5.1(a) the tree is constructed by greedily choosing the nodes with maximum stability coefficient (this is called s-first), while Fig. 5.1(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 $stab_{T_a}$=$\sum_{i=1}^{10} stab_{T_a}(n_0, n_i)$= (1.0·1) + (1.0·1) + (0.9·1.0)+(0.9·1.0)+(0.8·1.0)+(0.8·1.0)+(0.8·1.0)+(0.7·0.9·1.0)+(0.7·0.9·1.0)+ (0.7·0.9·1.0) =8.09. In the same way, $stab_{T_b}$ becomes 7.9, and that says s-first approach is better for this example.

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 ap-

proaches, which have been used in the early-stage lifetime aware schemes such as [26], [27] and [28]. 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.

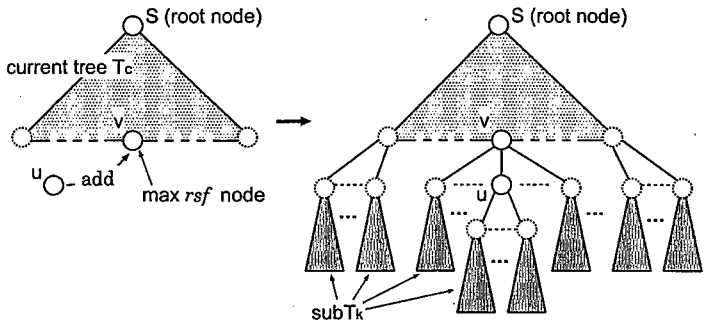
This consists of two main steps. First, it simply selects the peer node p from the current tree, where p has the maximum path stability with at least one residual degree. Second, the node for adding to the tree is selected. For this purpose, a new metric called *estimated tree stability* is introduced, 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 and for that the following auxiliary parameters are used.

- T_{cur+u} : the tree after adding node u to the current tree T_{cur}
- R_{rem} : the set of nodes that have not been included in T_{cur+u}
- $\hat{sc}$: the average stability coefficient of nodes in R_{rem}
- $\hat{d}$: the average degree of nodes in R_{rem}
- R_{res} : the set of nodes that have residual degrees on T_{cur+u}
- $outd(t)$: 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 $subT$. (see Fig. 5.2). 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 $subT$).

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



(a) The current tree T_{cur} (b) The estimated tree after adding node u

Figure 5.2: Concept of the Centralized Heuristic Algorithm

Here, $m = log_d|subT|$ 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.

5.3 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 [26] 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 [11] 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 [11] 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 joins and departures, we assume that they are handled by some existing decentralized protocols such as [24] and mainly concentrate on the initial tree construction and refining procedures. These procedures are described in detail in the following sections [2].

[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.

5.3.1 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 [11] 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.

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});

5.3.2 Tree refinement

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

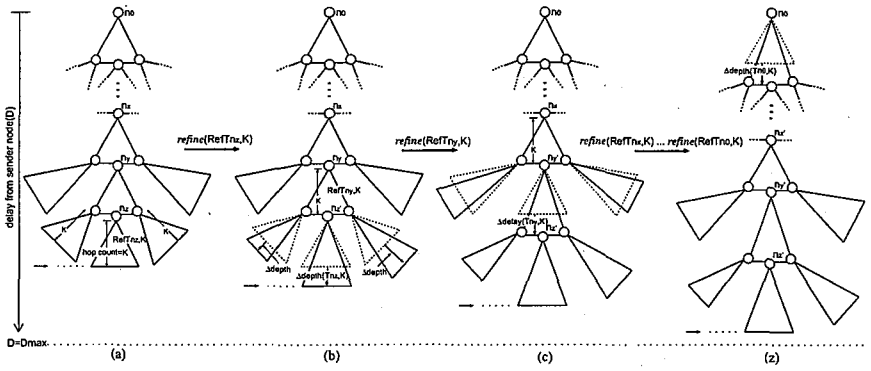


Figure 5.3: The Outline of the Pre-Session Refinement of Initial Tree

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 as outlined in Fig. 5.3.

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.

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.

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

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

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.

$$\Delta_{u,w} = \begin{cases} \sum_{n=1}^{d(u)} \Big(sc(w) - sc(u)\Big) \cdot stab_{T_{x_n}} + \\ \sum_{n=d(u)+1}^{d(w)} \Big(sc(w) - sc(u) \cdot R \cdot sc(w)\Big) \cdot stab_{T_{y_n}} \\ \qquad\qquad (\text{if } d(w) \geq d(u),\ \text{Fig.5.4(b)}) \\ \sum_{n=1}^{d(w)} \Big(sc(w) - sc(u)\Big) \cdot stab_{T_{x_n}} + \\ \sum_{n=d(w)+1}^{d(u)} \Big(sc(u) \cdot R \cdot sc(w) - sc(w)\Big) \cdot stab_{T_{x_n}} \\ \qquad\qquad (\text{if } d(w) < d(u),\ \text{Fig. 5.4(c)}) \end{cases}$$

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. 5.5(a). Then, the delay portion 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,

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

$$D(s_f, z)_{new} = \begin{cases} D(s_f, u)_{new} + D(u, z) & \text{if } d(w_j) \geq d(u) \\ \qquad\qquad (z \text{ is not moved along with } w_j) \\ D(s_f, w_j)_{new} + D(w_j, z) & \text{else} \\ \qquad\qquad (z \text{ is moved along with } w_j) \end{cases}$$

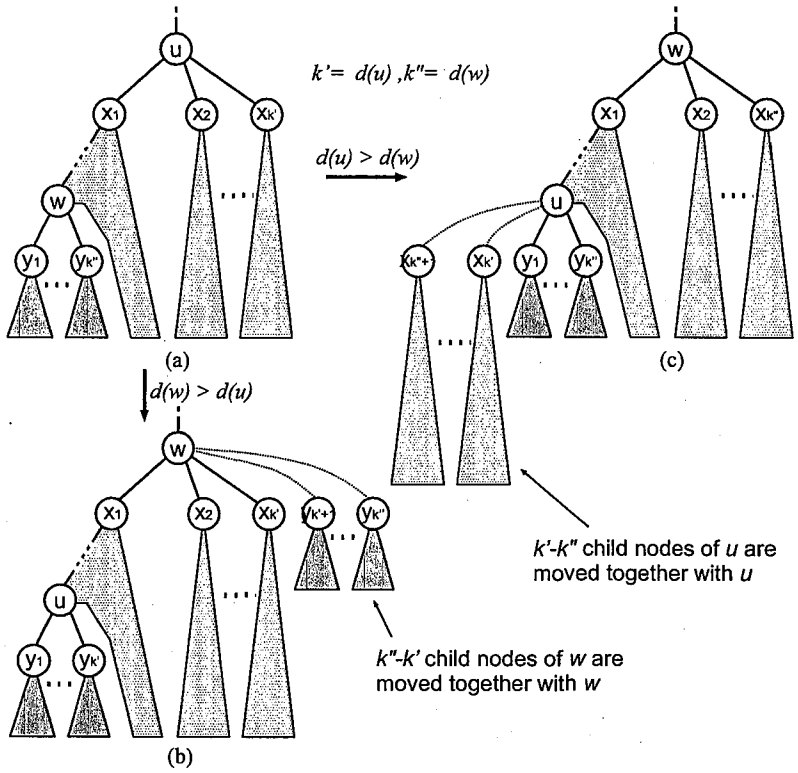


Figure 5.4: Concept of Replacing Candidate Selection: (a) the initial tree (b) the tree after swapping u and w when $d(w) > d(u)$ (c) the tree after swapping u and w when $d(w) < d(u)$

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.

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

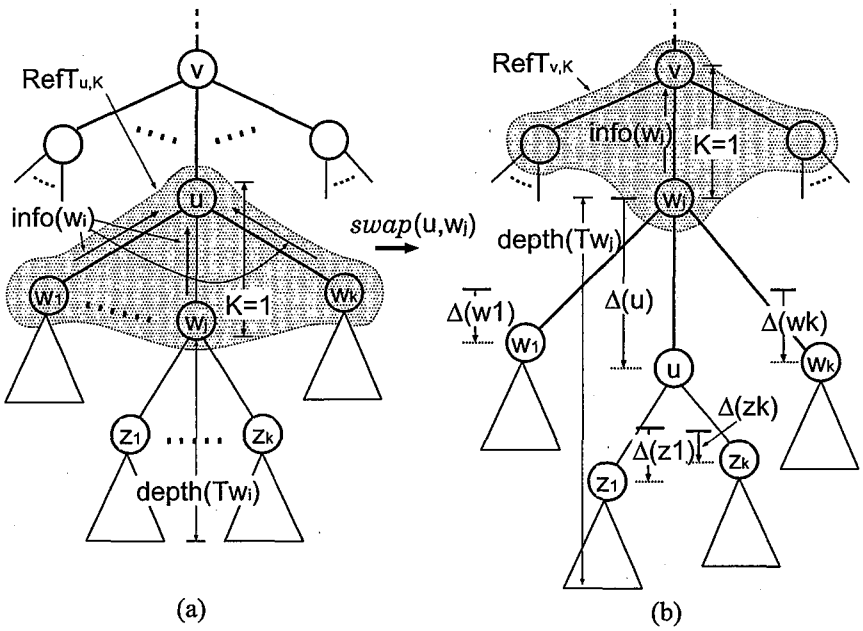


Figure 5.5: Refinement of $RefT_{u,K}$ (a) before refining (b) after refining

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*.

Here we describe the way that such nodes are treated using the example in Fig. 5.6. 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}.

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

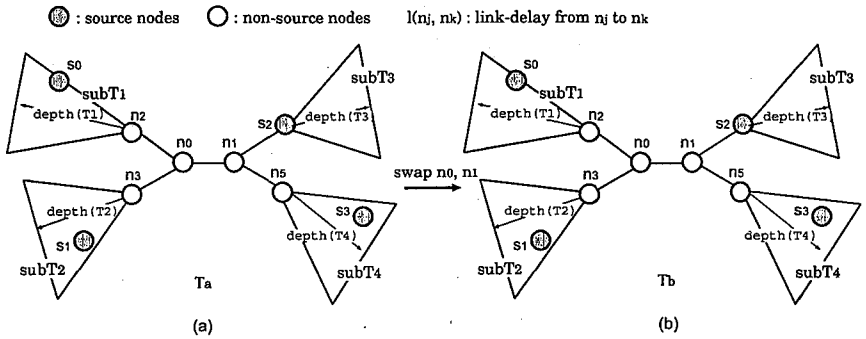


Figure 5.6: Dealing with Multiple Sources: Exchanging nodes on paths between source nodes (a) before exchange (b) after exchange

$\sum_{r \in R} stab_{T_a}(s_i, r)$ $(i = 1, 2, 3)$ 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]. 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[5].

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

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

Table 5.1: Performance of Proposed Protocol for Various Delay Bounds

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

5.4 Experiments

5.4.1 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 [33] for ALM protocols and will be open to the public soon.

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 [27] 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 $b = 1$.

[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 5.1 shows this situation. When the

Table 5.2: Swapping Policy Comparison

	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

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 $stab_T$.

[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 [27] and [28]. Table 5.2 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 5.2 compares the centralized algorithm described in Section 5.2.2, 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.

[Effect of source node count] Another interesting aspect to see is how our protocol works with different numbers of source nodes. Table 5.3 illustrates the results. 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.

Table 5.3: Effect of the Number of Source Nodes

# of sources	1	2	3	4	5
$stab_T$	0.65	0.63	0.62	0.58	0.56
$depth_T$[ms]	1427	1398	1390	1452	1421

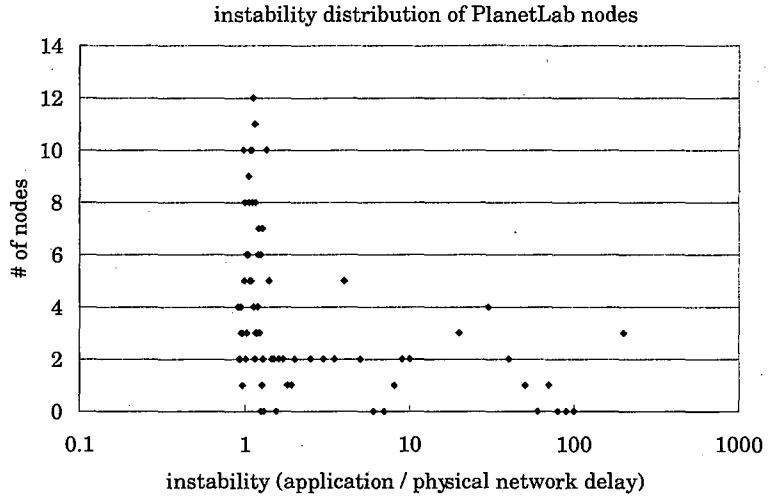


Figure 5.7: PlanetLab Instability Distribution

5.4.2 PlanetLab experiments

We have evaluated the performance of the proposed protocol through experiments on PlanetLab. The experiments were carried out using our middleware [33]. 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.

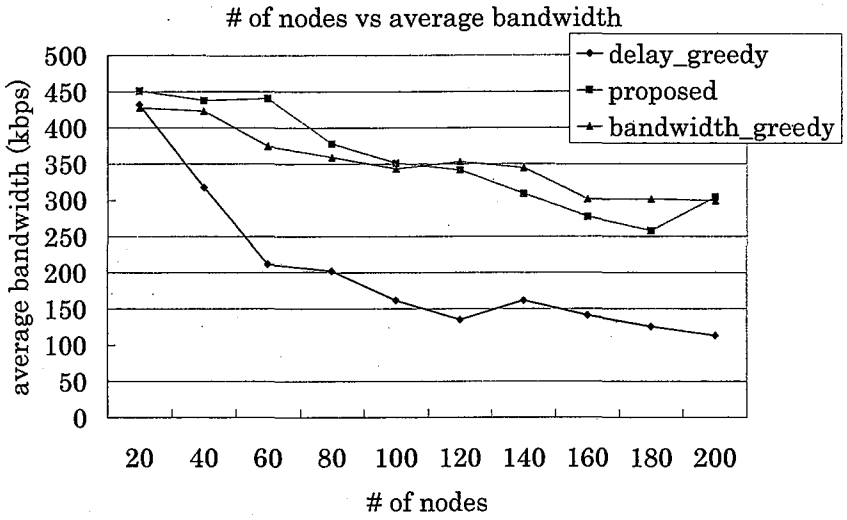


Figure 5.8: Average Bandwidth vs Number of Nodes

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. 5.7. 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.

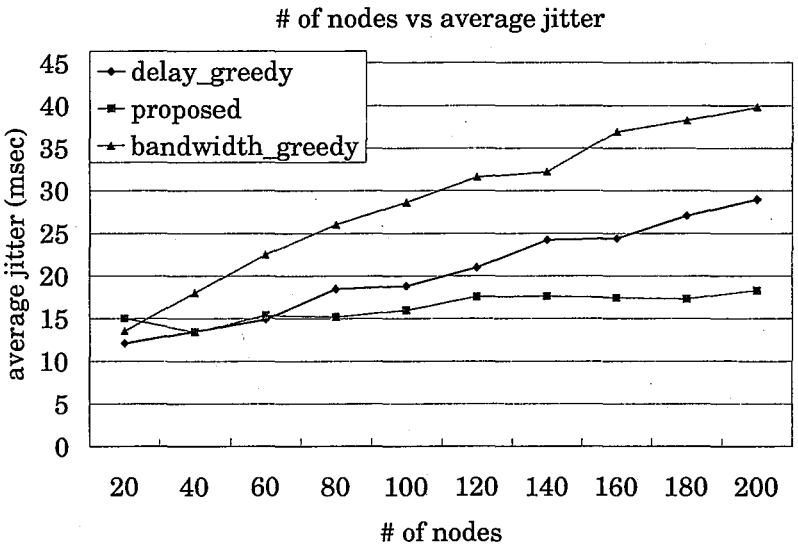


Figure 5.9: Average Jitter vs Number of Nodes

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 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 [34].

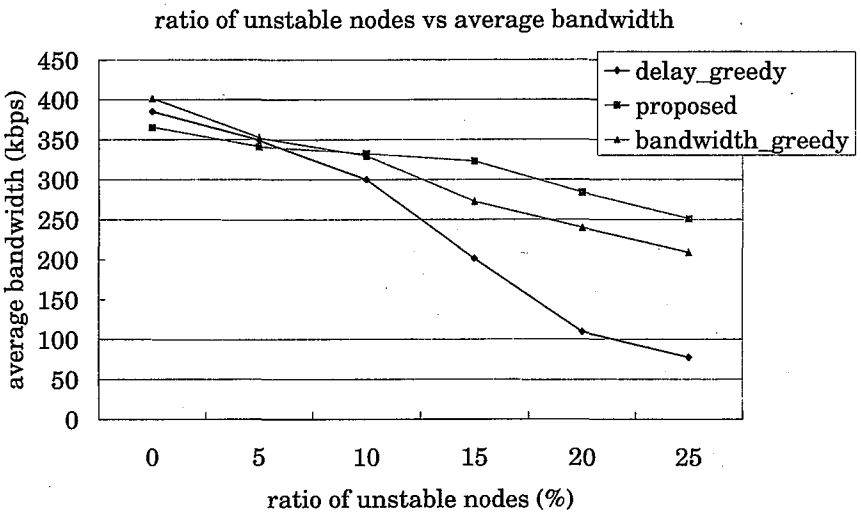


Figure 5.10: Average Bandwidth vs Ratio of Unstable Nodes

Experimental results

Fig. 5.8 and Fig. 5.9 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. 5.10 and Fig. 5.11 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.

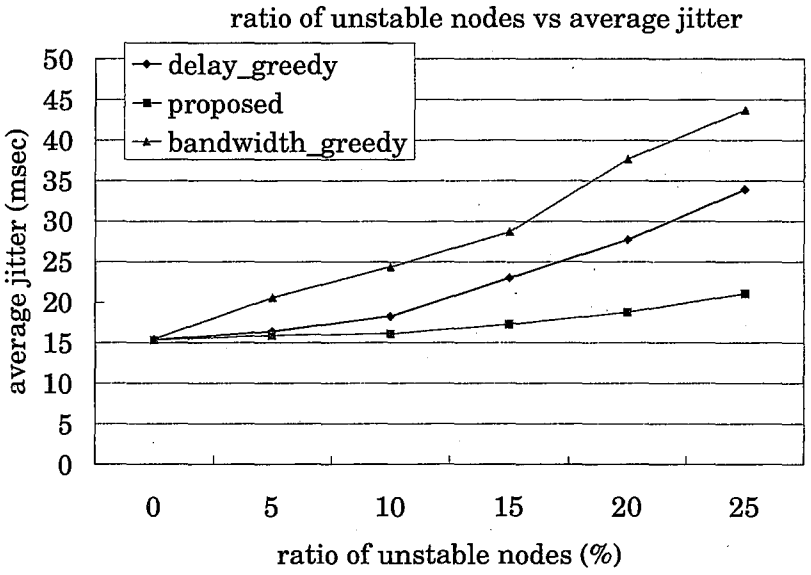


Figure 5.11: Average Jitter vs Ratio of Unstable Nodes

5.5 Concluding Remarks

In this chapter, 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.

Chapter 6

Conclusion

The major contributions of the ALM schemes discussed in this dissertation are as follows.

C1. Decentralized construction of minimum delay spanning trees under bandwidth constraints

C2. Formulation of a new problem associating multiple-sources in minimum delay spanning trees

C3. Design and implementation of an adaptation mechanism that well supports media streaming

C4. Consideration of heterogeneous end-users for interactive overlay multicast

C5. Decentralized construction of stability oriented spanning trees in multiple source context

As for C1, an ALM scheme called MODE has been proposed in order to meet the major demands of interactive group applications, which constructs a *Degree Bounded Minimum Diameter Tree* (*DBMDT*) in a de-centralized manner. The overlay tree, which is a spanning tree, constructed by this scheme satisfies the following conditions: (i) the maximum delay between any pair of nodes is minimized (referred by "minimum diameter"), (ii) the number of overlay links connected to each node (degree) is restricted by its bandwidth availability (referred by "degree bounded") and (iii) scalable with its de-centralized design. In addition to the above features, MODE is resilient for node failures (un-announced departures) by providing quick restoring for

broken trees. For this, MODE proactively calculates a *node-to-connect* for each node using some local information and this node is used in case of isolation from the main tree. The experimental results using ns-2 have shown that MODE could achieve similar diameters with CT[11] algorithm, a centralized scheme that greedily constructs a DBMDT, in a small computation time and small amount of control traffic even though MODE is autonomous and decentralized.

C2 discusses the association of multiple sender nodes in minimum diameter spanning trees and minimizes the maximum delay from those sender nodes. This constructs a *sender-dependant Degree Bounded Minimum Diameter Tree(s-DBMDT)*. A java middleware based on STS protocol has also been designed and implemented (C3). This middleware consists of an adaptation mechanism to identify and relocate the nodes of low streaming performance, and this helps to achieve a better multimedia streaming quality. The performance evaluation, which has been done based on experiments in both simulated networks and real networks, strongly indicates the efficiency and usefulness of our protocol.

C4 discusses the issues of mobile terminals in ALM schemes, and proposes a protocol called *MODE-m* extending from our initial work MODE. In MODE-m the ALM tree is constructed carefully and dynamically to prevent those hosts from staying at critical positions that affect the end-to-end latency and the stability of the ALM tree. The experimental results have shown that this dynamic feature is very effective to keep the diameter as small as possible under the existence of mobile nodes.

The *reliability* problems of end-users have been discussed in C5. For example, less-desired users leave the session sooner and often without any prior notification or cause delay and harmful jitters when they stay in the session, which makes the participants in the downstream suffer considerable degradation of the quality. C5 addresses this reliability issue by recognizing it as the *user-lifetime* and propose a stability oriented overlay multicast scheme, which covers an aspect different from the previous MODE, STS or MODE-m. The extensive simulations and experiments conducted in PlanetLab have shown the usability of this protocol.

The future work of this study includes conducting larger scale real world experiments with the proposed ALM schemes to investigate their pros and cons in various real world situations. We strongly hope those will help ALM technology to spread through the real world network applications to realize the next generation communication paradigm.

Acknowledgement

My foremost thank goes to my supervisor Professor Teruo Higashino of Osaka University. Without him, this dissertation would not have been possible. I thank him for his patience, kindness and excellent understanding that carried me on through difficult times of both research and personal life, and for his continuous support, encouragement and guidance of the work.

I am very grateful to Professor Makoto Imase, Professor Koso Murakami, Professor Masayuki Murata and Professor Hirotaka Nakano of Osaka University for their invaluable comments and helpful suggestions concerning this thesis.

I would like to express my sincere and deep gratitude to Associate Professor Hirozumi Yamaguchi of Osaka University for the continuous guidance, valuable suggestions and discussions throughout this work. His valuable feed back has contributed greatly to this dissertation. And also I would like to express my thank to Professor Akio Nakata of Hiroshima City University, Associate Professor Keiichi Yasumoto of Nara Institute of Science and Technology who have provided many valuable comments.

I am very grateful to Assistant Professor Takaaki Umedu of Osaka University for his insightful and constructive comments. And also a big thank goes to all the members of Higashino Laboratory of Osaka University for their helpful advice.

I am also grateful to Kyouritsu International Exchange Foundation, Ito Foundation For International Education Exchange and International Communications Foundation (ICF) in Japan for supporting me financially from under graduate to post graduade studies.

Again my deep gratitude goes to my parents who educated me being in infinite difficulties, and to my two bothers for their support and encouragement.

Finally, I would like to thank my wife Malee, daughter Sayumi and son Misaka for their infinite patience, encouragement and understanding.

Bibliography

[1] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS '00)*, pages 1–12, 2000. Tools are provided: http://www-2.cs.cmu.edu/ streaming/index.html.

[2] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proc. of ACM Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 55–67, 2001.

[3] V. Roca and A. El-Sayed. A host-based multicast (HBM) solution for group communications. In *Proc. of IEEE Int. Conf. on Networking (ICN'01)*, pages 610–619, 2001.

[4] P. Francis. Yoid: Extending the internet multicast architecture. http://www.isi.edu/div7/yoid/, 2002.

[5] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of USENIX Symp. on Internet Technologies and Systems (USITS'01)*, pages 49–60, 2001.

[6] S.Y. Shi, J.S. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *Proc. of ACM Int. workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV'01)*, pages 83–91, 2001.

[7] 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 Int. Conf. on Computer Communications (INFOCOM'03), pages 1521–1531, 2003.

[8] R. Cohen and G. Kaempfer. A unicast-based approach for streaming multicast. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'01)*, pages 440–448, 2001.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'02)*, pages 205–217, 2002.

[10] X. Zhang, J. Liu, B. Li, and Y.-S.P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'05)*, pages 2102–2111, 2005.

[11] S. Shi and J. Turner. Routing in overlay multicast networks. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'02)*, pages 1200–1208, 2002.

[12] H. Yamaguchi, A. Hiromori, T. Higashino, and K. Taniguchi. An autonomous and decentralized protocol for delay sensitive overlay multicast tree. In *Proc. of IEEE Int. Conf. on Distributed Computing Systems (ICDCS'04)*, pages 662–669, 2004.

[13] E. M. Royer and C. E. Perkins. Multicast ad hoc on-demand distance vector (maodv) routing. In *IETF Internet Draft, draft-ietf-manet-maodv-00.txt*, 2000.

[14] D. Thaler and C. V. Ravishankar. Distributed center-location algorithms. *IEEE Journal on Selected Areas in Communications*, 15(3):291–303, April 1997.

[15] J. C.-H. Lin S. Paul, K. K. Sabnani and S. Bhattacharyya. Reliable multicast transport protocol (rmtp). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, April 1997.

[16] Y. Chawathe, S. McCanne, and E. A. Brewer. RMX: Reliable multicast for heterogeneous networks. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'00)*, pages 795–804, 2000.

[17] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'02)*, pages 1366–1375, 2002.

[18] S. McCanne and V. Jacobson. Vic: A flexible framework for packet video. In *Proc. of ACM Int. Conf. on Multimedia (MULTIMEDIA'95)*, pages 511–522, 1995.

[19] J. Liebeherr and T. K. Beam. HyperCast: A protocol for maintaining multicast group members in a logical hypercube topology. In *Proc. of Int. Workshop on Networked Group Communication (NGC '99) (LNCS 1736)*, pages 72–89, 1999.

[20] J. Liebeherr and M. Nahas. Application-layer multicast with delaunay triangulations. In *Proc. of IEEE Global Telecommunications Conf. (Globecom'01)*, pages 1651–1655, 2001.

[21] N. Mimura, K. Nakauchi, H. Morikawa, and T. Aoyama. RelayCast: A middleware for application-level multicast services. In *Proc. of Int. Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC'03)*, pages 197–212, 2003.

[22] Y. Cui, B. Li, and K. Nahrstedt. ostream: Asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, pages 91–106, June 2004.

[23] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'04)*, pages 2743–2753, 2004.

[24] T. M. Baduge, A. Hiromori, H. Yamaguchi, and T. Higashino. Design and implemetation of overlay multicast protocol for multimedia streaming. In *Proc. of IEEE Int. Conf. on Parallel Processing (ICPP'05)*, pages 41–48, 2005.

[25] 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 (ICME'03)*, pages III 497–500, 2003.

[26] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application endpoints. In *Proc. of ACM Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'04)*, pages 107–120, 2004.

[27] M. Bishop, S. Rao, and K. Sripanidkulchai. Considering priority in overlay multicast protocols under heterogeneous environments. In *Proc. of IEEE Int. Conf. on Computer Communications (INFOCOM'06)*, pages 1–13, 2006.

[28] Guang Tan and Stephen A. Jarvis. Improving the fault resilience of overlay multicast for media streaming. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):721–734, June 2007.

[29] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *IEEE Annals of the History of Computing*, 7(1):43–57, January 1985.

[30] P. Huang and J. Heidemann. Minimizing routing state for light-weight network simulation. In *Proc. of Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 108–116, 2001.

[31] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *Proc. of ACM SIGCOMM Conf. on Internet Measurement*, pages 41–54, 2004.

[32] 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)*. W.H. Freeman & Company, third edition, 1979.

[33] 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 Int. workshop on Network and Operating Systems Support for Digital Audio & Video (NOSSDAV'07)*, pages 125–130, 2007. Tools are provided: http://www-higashi.ist.osaka-u.ac.jp/software/ALM/middlewareAPI/.

[34] 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.