uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yong Zhong

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Proxy-based Single Source Application Layer Multicast Media Streaming

TITRE DE LA THÈSE / TITLE OF THESIS

A. El Saddik

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

S. Shirmohammadi

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

M. Weiss

J. Zhao

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Proxy-Based Single Source Application Layer Multicast Media Streaming

By

Yong Zhong

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements for the degree of

Master of Applied Science

In

Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

© Yong Zhong, Ottawa, Canada, 2005

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-11478-9
Our file Notre référence
ISBN: 0-494-11478-9

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

Multicasting is the data distribution from one sender to a group of receivers. Traditionally multicasting is implemented at the network layer, in the way that routers perform membership management, maintain data delivery path, and replicate and forward data. IP Multicasting is the most efficient way for group data distribution. However, it has been shown that it is extremely difficult to deploy IP Multicasting at a large scale. There are scalability issues, deployment hurdles and marketing reasons. Therefore an alternative has been proposed to shift multicast support to the application layer. This approach expects end-hosts participating in the application to perform multicast functions. This is Application-Layer Multicasting (ALM). This thesis analyzes existing ALM protocols and proposes a proxy-based single source ALM protocol which targets media streaming applications, where latency is the overlay building metric. A single source real-time media streaming application is implemented based on this protocol. Some measurements are taken both on the intranet and on the Internet. The performance data provided by the tests show that the proposed protocol introduces acceptable control overheads. Finally, based on the experience with the implemented ALM system, recommendations for future work are provided.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor Dr. Abdulmotaleb El Saddik, for giving me an opportunity to pursue my graduate study and providing valuable support whenever I need. I greatly appreciate his trust, guidance, enthusiasm, and encouragement.

My gratitude also goes to Dr. Shervin Shirmohammadi, my thesis co-supervisor, who has taught me a lot on how to conduct research. I could not have accomplished my thesis without his help.

I would also like to thank my colleagues in the Multimedia Communication Research Laboratory, University of Ottawa, for providing me a pleasant working environment.

The financial support provided by the National Capital Institute of Telecommunications (NCIT) is also highly appreciated.

Finally, thank to all my family members, for giving me endless love, care and support.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures vii

List of Tables ix

List of Abbreviations x

Chapter 1 1

Introduction 1

1.1 Motivation 1

1.2 Research Problem 2

1.3 Research Objective 3

1.4 Research Contributions 4

1.5 Organization of the Thesis 4

Chapter 2 6

Background 6

2.1 Multicast Overview 6

2.1.1 IP Multicast Problems 8

2.1.2 MBone 9

2.2 Application-Layer Multicasting 10

2.2.1 Application Domains 10

2.2.2 Classifications 11

2.2.3 Related work 14

2.2.4 ALM Metrics and Evaluation Criteria 19

Chapter 3 21

The Proposed Protocol: ProBaSS 21

3.1 Algorithm 21

3.2 Theoretical Evaluation 24

Chapter 4 26

System Design 26

4.1 System Architecture 26

4.2 Use cases 27

4.3 Static Model 30

4.4 Dynamic Model 34

Chapter 5 42

Implementation 42

5.1 System Flow Chart 42

5.2 Proxy Package 44

5.3 Client Package 49

5.4 Proxy Side Client Package 51

5.5 Race Condition Solution 51

Chapter 6 54

Testing and Measurement 54

6.1 Text-Based Online Chatting System 54

6.1.1 System Design 54

6.1.2 Measurement 55

6.2 Single Source Video/Audio Conferencing System 59

6.2.1 System Design 61

6.2.2 Implementation 62

Chapter 7 66

Conclusions and Future Work 66

7.1 Conclusions 66

7.2 Future Work 67

References 69

List of Figures

Figure 1.1 IP Multicasting and Application-Layer Multicasting 3

Figure 2.1 Network traffic in multiple unicasting and multicasting 6

Figure 2.2 Unicast and multicast forwarding tables 8

Figure 2.3 Proxy-based and end-system based ALM architectures 12

Figure 2.4 Mesh and tree building procedure in Narada 15

Figure 2.5 Hierarchical arrangement of hosts in NICE 15

Figure 2.6 Control and data delivery paths for a two-layer hierarchy in NICE 16

Figure 2.7 Data delivery paths in NICE and ZIGZAG 16

Figure 2.8 CAN with 2D coordinate space: control and data topologies 17

Figure 2.9 Routing in the Delaunay triangulation overlay 17

Figure 2.10 Host multicast architecture 18

Figure 2.11 OMNI architecture 18

Figure 3.1 The overlay multicast architecture of ProBaSS 21

Figure 3.2 Member join algorithm 22

Figure 3.3 Member leave algorithm 23

Figure 4.1 System architecture 26

Figure 4.2 Layered system architecture 27

Figure 4.3 Client-Server architecture 27

Figure 4.4 Use case diagram 28

Figure 4.5 Use case diagram of sub-systems 30

Figure 4.6 Class diagram for proxy server sub-system 31

Figure 4.7 An example of a multicast tree and its object diagram 31

Figure 4.8 Class diagram for client sub-system 33

Figure 4.9 Package diagram 33

Figure 4.10 Packages of each sub-system 34

Figure 4.11 Activity diagram of the member join use case 35

Figure 4.12 Activity diagram of the refresh message exchange use case 37

Figure 4.13 Sequence diagram of the refresh message exchange use case (1) 38

Figure 4.14 Sequence diagram of the refresh message exchange use case (2) 38

Figure 4.15 Activity diagram of the member leave use case 40

Figure 4.16 Sequence diagram of the member leave use case 41

Figure 5.1 System flow chart 43

Figure 5.2 Race condition cases 52

Figure 6.1 User interface in simple text transmission 54

Figure 6.2 Message flow in simple text transmission 55

Figure 6.3 Round trip time of IP Multicast and ALM 56

Figure 6.4 Test on ALM processing time 56

Figure 6.5 Test on the intranet 57

Figure 6.6 Test on the Internet 58

Figure 6.7 Streaming between proxy and client 63

Figure 6.8 Receiver's screen 65

List of Tables

Table 6.1 Online chatting system deployment configuration 56

Table 6.2 ALM processing time 57

Table 6.3 Test data from the intranet 57

Table 6.4 Test data from the Internet 58

Table 6.5 Bandwidth overhead 59

Table 6.6 Video/Audio conferencing system deployment configuration 64

List of Abbreviations

ALM	Application-Layer Multicasting
MBone	Multicast Backbone
IETF	Internet Engineering Task Force
QoS	Quality of Service
IGMP	Internet Group Management Protocol
PIM	Protocol Independent Multicast
DVMRP	Distance Vector Multicast Routing Protocol
CBT	Core Based Tree
RP	Rendezvous Point
OMNI	Overlay Multicast Network Infrastructure
MSN	Multicast Service Node
HMTP	Host Multicast Tree Protocol
CAN	Content Addressable Network
VOD	Video-on-Demand
RTT	Round-trip time
UML	Unified Modeling Language
P2P	Peer-to-Peer
HTTP	HyperText Transfer Protocol
TCP	Transmission Control Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
RTP	Real-time Transport Protocol
RTCP	Real-time Control Protocol
JMF	Java Media Framework
LAN	Local Area Network
GUI	Graphic User Interface
CODEC	Compression/Decompression
NAT	Network Address Translator
ITU	International Telecommunication Union

Chapter 1

Introduction

1.1 Motivation

The traditional Internet service, known as unicasting, is based on one-to-one data communication. Unicasting is well suited when communications occurs between two specific hosts (e.g. Email, web browsers). However, many applications today require one-to-many and many-to-many communications. Among them are shared virtual spaces, multimedia distribution services, video conferencing, and bulk data transfer, etc. For this kind of applications, which involve transmitting a large amount of data to multiple recipients, the unicast service is no longer efficient. For example, considering a multimedia content provider using unicasting to distribute a 2 Mb/s streamed video, to support 1,000 viewers, the server needs a 2 Gb/s access. Therefore, the server interface capacity can be a significant bottleneck, limiting the number of unicast video streams per video server. Replicated unicast transmissions consume a lot of bandwidth within the network, which is another significant limitation [12, 34, 35].

To overcome this unnecessary consume of bandwidth, an efficient one-to-many communication scheme – multicasting – was developed. Multicasting is the most efficient way of distributing data from one sender to multiple receivers with minimal data duplication. The data are not sent individually to each recipient; instead, only one copy of data is sent, but it is replicated when the network links split. It is forwarded until it reaches a destination. In the end, all specified destinations receive a copy of data.

IP Multicasting and Application-Layer Multicasting (ALM) are two different approaches to implement multicasting. IP Multicast protocol implements multicasting at network routers, while ALM protocols implements Multicasting within applications. The purpose of this thesis is to investigate existing multicast protocols to find a way that is practical and cost-effective to deploy multicasting via the Internet.

1.2 Research Problem

Multicasting is defined as the distribution of content from one sender to a group of receivers. Basic multicast functionality involves group membership management, data delivery path maintenance, and data replication and forwarding [12]. Traditionally multicasting is implemented at the network layer, in the way that network routers define the data delivery tree. As packets flow on this tree, they are replicated by routers at different branch points of the tree. This is IP Multicast architecture. IP Multicasting is the most efficient way to perform group data delivery, for it is able to eliminate traffic redundancy and improve bandwidth utilization on the wide-area network [1].

However, after extensive researches for years, large scale deployments of IP Multicasting are still sparse. There are technical as well as non-technical reasons.

First, multicasting requires routers to maintain more information than unicasting. In unicasting, the router maintains a routing table that contains an entry for each destination address, and these addresses are aggregatable. But in multicasting, the router maintains an additional multicast routing table, each entry for a multicast group address. And these addresses are not easily aggregated. This means the multicast routing table may grow very large and may change frequently when hosts join or leave any group, which is very challenging for routers [12]. This is the key issue that prevents IP Multicasting from being widely deployed.

Second, current deployment practices of IP Multicasting require manual configuration at routers to form the MBone, which makes the MBone expensive to set up and maintain [2]. There lacks a mechanism to support auto-reconfiguration at deployment stage to adapt network changes.

Third, providing reliability, congestion control, flow control, and security is more difficult and complex in multicasting than the unicast case.

And last, billing management for multicast traffic is not well-defined. The traditional pricing model is that only downstream is charged. But in multicasting, any participant may introduce large amount of upstream. Therefore the pricing model needs to be re-defined.

1.3 Research Objective

Our research is motivated by a real-world problem: how can bandwidth-intensive content like video be delivered to a large number of Internet users? Without multicast support it is impractical because the bottleneck bandwidth between content providers and consumers is considerably less than the natural consumption rate of such media [4].

Due to the problems of IP Multicasting, an alternative has been proposed to shift multicast support from core routers to end systems. This is Application-Layer Multicasting (ALM). Unlike IP Multicasting where data packets are replicated at routers inside the network, in ALM data packets are replicated at end-hosts. The end-hosts form a multicast overlay, and the goal of ALM is to construct and maintain an efficient overlay for data transmission [1]. The two approaches of multicasting are shown in Figure 1.1.

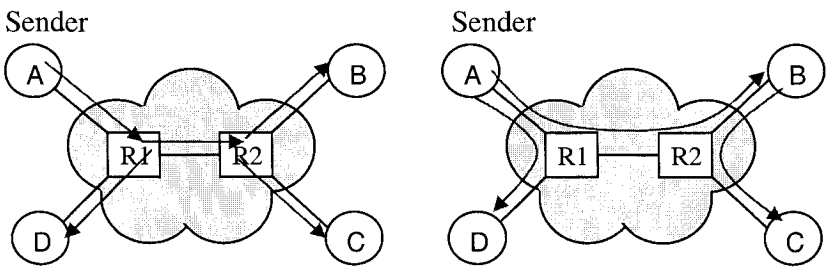


Figure 1.1 IP Multicasting and Application-Layer Multicasting
(Square nodes are routers and circular nodes are end-hosts)

Application-Layer Multicasting has advantages over IP Multicasting. Since the routing information is maintained by applications, it is more scalable. It supports dynamically scaled large number of users and concurrent groups; and since it needs no infrastructure support, it is fully deployable on the Internet. So in theory, by using ALM, content providers can deliver bandwidth-intensive contents such as TV programs to large number of users via the Internet.

Our research objective is to design and implement an ALM protocol for content providers to deliver video/audio content to a large number of receivers from a single source via the Internet.

1.4 Research Contributions

This thesis contains a number of contributions, including:

- Analysis of existing ALM protocols by using different classifications
- Proposal of an overlay multicast architecture which is different from existing overlay multicast architectures. It combines advantages of both proxy-based architecture and end-system based architecture to support scalable overlay multicast applications
- Design and implementation of a proxy-based single source ALM protocol for content providers to deliver video/audio to a group of users via the Internet
- Field tests and performance measurements of the proposed protocol
- Implementation of a Video/Audio conferencing application which supports real-time multicast video/audio streaming
- Publications:
  - Yong Zhong, Shervin Shirmohammadi, and Abdulmotaleb El Saddik, "Measurement of the Effectiveness of Application-Layer Multicasting", *Proc. IEEE Instrumentation and Measurement Technology Conference,* Ottawa, Canada, May 17-19, 2005 (Accepted, to appear).
  - Yong Zhong, Shervin Shirmohammadi, and Abdulmotaleb El Saddik, "ProBaSS: An Application-Layer Multicast Protocol for Media Streaming", *ACM International Conference on Multimedia,* Singapore, November 06-12, 2005. In submission.

1.5 Organization of the Thesis

We have stated our research motivation, problem, objective and contributions. The remaining part of the thesis is organized as follows:

Chapter 2 provides the background knowledge of multicast technologies. The IP Multicast technology is discussed first. Then an overview of ALM is presented from the aspects of application domains, classifications, existing protocols, and ALM evaluation criteria.

Chapter 3 presents the proxy-based single source ALM protocol. First, the overlay multicast architecture and algorithm are proposed. Then a theoretical evaluation is given with respect to ALM evaluation criteria.

In **Chapter 4**, first the system is depicted with respect to its substrate network protocol supports. Then the system is modeled using Unified Modeling Language (UML).

Chapter 5 presents the implementation of the system following its client-server

architecture. More specific, main components of the system are detailed and significant issues about the system are highlighted.

Chapter 6 evaluates the system via two tests. The first test extends the system to support text-based simple message exchanges. Particular measurements are taken during this test. The second test extends the system to support real-time video/audio streaming.

Chapter 7 summarizes and concludes the thesis. It provides recommendations for future research.

Chapter 2

Background

2.1 Multicast Overview

Any form of network communication involving the transmission of information to multiple recipients can benefit from the bandwidth efficiency of multicast technology. Examples of applications involving one-to-many or many-to-many communications include: video and audio streaming, video conferencing, dissemination of news and stock quotes, database replication, and software downloads [34].

A sender can use two different approaches to send data to three recipients: multiple unicasting and multicasting. Figure 2.1 shows the comparison of the network traffic using these two approaches.

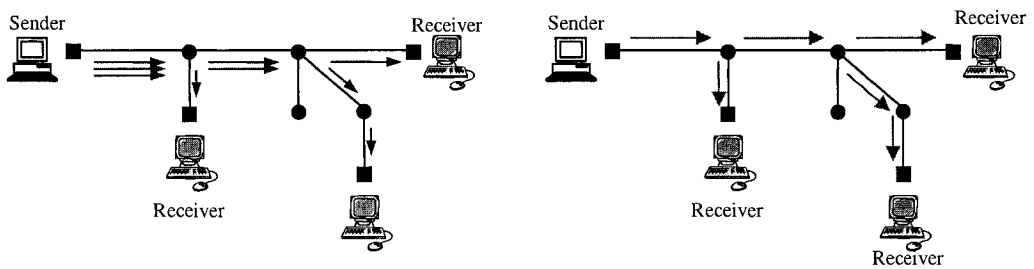


Figure 2.1 Network traffic in multiple unicasting and multicasting

Obviously, when there are a large number of recipients of a replicated transmission, multicast technology makes a tremendous difference in both server load and network load, because only one copy of data traverses each network link. Another feature of multicasting is that the multicast data are delivered nearly simultaneously to all members of the recipient group. The variability in delivery time is only limited to differences in end-to-end network delay. While in unicasting, the variability in delivery time can be large, especially for large transmissions or large distribution lists. Financial services are especially beneficial from this feature [34]. In addition, in multicasting, the sender does not need to know the unicast network address of any particular recipient. All participants use the same

multicast group address to communicate with the group rather than any single recipient.

IP Multicasting, proposed in 1988, was the first attempt to implement multicasting in a large scale using the Internet. In IP Multicasting, three special mechanisms are involved [34]:

Membership Management: There must be a mechanism that informs the network that the computer is a member of a particular group. Without this information, the network would be forced to flood rather than multicast the transmissions for each group. For IP networks, the Internet Group Multicast Protocol (IGMP) [36] is an IP datagram protocol between routers and hosts that allows group membership lists to be dynamically maintained. The host sends an IGMP "report" to the router to join the group. Periodically, the router sends a "query" to learn which hosts are still part of a group. If a host wants to continue its group membership, it responds to the query with a report. If the host sends no report, the router prunes the group list to minimize unnecessary transmissions. With IGMP V2, a host may send a "leave" message to inform the router that it is no longer participating in a multicast group. This allows the router to prune the group list before the next query is scheduled.

Multicast Forwarding: Most IP multicast applications are based on UDP, which uses "best effort delivery" and lacks the congestion control mechanism of TCP. As a result, multicast packets may be dropped more often than unicast TCP packets. Since it is not practical for real-time applications to request retransmissions, audio and video streaming may suffer degradation due to packet drops.

Multicast Routing: The network must be able to build packet distribution trees that specify a unique forwarding path between the subnet of the source and each subnet containing members of the multicast group. A primary goal in the construction of distribution trees is to ensure that at most, one copy of each packet is forwarded over each branch of the tree. This is accomplished by constructing a spanning tree rooted at the designated multicast router of the sending host, and providing connectivity to the designated multicast routers of each receiving host. There are several multicast routing

protocols for IP Multicasting. These include: the Distance Vector Multicast Routing Protocol (DVMRP), Protocol-Independent Multicast (PIM), and Core-Based Trees (CBT).

2.1.1 IP Multicast Problems

IP multicasting provides a powerful abstraction where a group address identifies a group and any host can send a message to a group by sending to the group's address. To support IP multicasting, the router needs to be extended to support per group state. Figure 2.2 illustrates this concept [12].

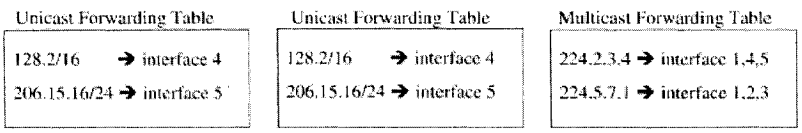


Figure 2.2 Unicast and multicast forwarding tables

A routing table is a mapping between network layer addresses and their underlying data link layer interfaces. In unicast routing, a router uses a unicast routing table to determine which interface to send an incoming packet with a given destination address. Therefore, the unicast routing table contains an output interface for each possible destination address. Of course, having an entry for each address is impractical, so the table aggregates addresses by prefixes. In the example, 128.2/16 means the prefix is 128.2 and the length of the prefix is 16 bits. A core router in the Internet may currently have about 120 K entries even after aggregation. A multicast router, in addition to the unicast routing table, requires a multicast routing table. This table specifies a list of output interfaces that a packet with a particular multicast group address should be sent to. There are two differences between unicast routing table and multicast routing table: 1) it is difficult to aggregate the multicast routing table and 2) an entry in the multicast routing table may change when hosts join or leave any group, where as an entry in the unicast routing table may change only when routes in the network change. The first one causes the multicast table to grow very large and the second one causes the multicast table to be modified frequently, which is a costly operation for routers [12].

Although IP Multicast technology is very powerful, implementation of IP Multicasting turns out to be extremely complex. To maintain per group state introduces scalability challenges at routers.

Quality of service (QoS) is another challenge that confronts IP Multicast. Most IP multicast applications use Real-Time Transport Protocol (RTP) based on UDP. UDP uses best effort delivery mechanism, while TCP uses congestion avoidance windowing mechanism. As a result, multicast packets may be dropped more often than unicast TCP packets. It is possible to improve the reliability of multicast transmissions by using priority mechanism, but this solution is also costly [34].

Further, the charging model is not straightforward. The traditional charging model is that only downstream is charged. But in multicasting, any participant may introduce large amount of upstream (e.g. video conferencing), therefore the traditional charging model is no longer proper for multicasting.

Due to the reasons above, IP Multicast service over the Internet is still not available today after more than a decade's efforts. It should be noted that IP multicast service has been successfully deployed in enterprise network environments. It may also play a role in metropolitan networks [12].

2.1.2 MBone

Since the majority of the routers on the Internet do not support multicasting, in 1992, the Internet Engineering Task Force (IETF) determined to implement multicasting using a software solution. A "virtual network", a network that runs on top of the Internet, was built, which allows multicast data to traverse the Internet. Thus the MBone was born.

MBone stands for "Multicast Backbone". It particularly refers to the IP Multicast backbone. It was designed to provide multicast functionality over the Internet. In order to achieve this goal, the multicast capable regions ("islands") are connected by virtual links called "tunnels", which permit multicast packets to traverse. Endpoints of these virtual links, or tunnels, are delicate Unix workstations that run a multicast routing daemon called "mrouted". These workstations encapsulate multicast packets into normal IP packets and route them to other multicast capable subnets over the unicast path between the tunnel endpoints. When the multicast packets that are encapsulated in unicast packets reach a router that understands multicast packets, or a workstation running "mrouted", the packets are recognized and processed as the multicast packets. These tunnels and multicast capable

islands form a spanning tree on top of the Internet, which is the MBone [31, 32].

To join the MBone, one has to set up a tunnel to the nearest MBone link. This requires knowledge of both the MBone logical map and the underlying physical network topology. For example, when a regional network wants to join in, they must make a request on the appropriate MBone list, then, the participants at "close" nodes answer and cooperate in setting up the ends of the appropriate tunnels. This is a cooperative task combining knowledge distributed among the participants. And the entire process requires at least a week or two. Once the regional network has multicast routing enabled, the MBone applications need to be installed [31].

Although the MBone is available for years, applications of the MBone are still limited. The complicated joining procedure and the manual configuration mechanism make the MBone expensive to set up and maintain, which has limited the MBone to be deployed in a large scale.

2.2 Application-Layer Multicasting

Due to the problems of IP Multicasting, an alternative has been proposed to shift multicast support to the application layer. This approach expects end-hosts participating in the application to perform multicast functions: membership management, data duplication and forwarding, and multicast tree maintenance. This is Application-Layer Multicasting (see Figure 1.1). The goal of ALM is to construct and maintain an overlay network, on top of the real network, for multicast data delivery.

Compared with IP Multicasting, ALM has several advantages. First, it needs no infrastructure support. It can be deployed on unicast-only infrastructure, so it can be applied universally. Moreover, since the membership knowledge is maintained by applications, it is capable of supporting dynamically scaled large number of users and concurrent groups. However, in ALM, data can be delivered multiple times over the same network link, so it is less efficient than IP Multicasting. Also, ALM protocols introduce more control overhead (bandwidth and delay) for the sake of supporting more users and groups. In this section, we give an overview of ALM.

2.2.1 Application Domains

Quite a lot of ALM protocols have been proposed in recent years. They are designed for different application domains. ALM targeted applications include live media streaming, media on demand, Audio/Video conferencing, bulk data transfer (such as P2P file sharing), publish-subscribe event notification service, shared whiteboard, etc. Different ALM protocols are designed to target different application domains. For example: Narada [6] targets small to media sized group Video conferencing applications; Overcast [4] targets bulk data transfer applications; while Scribe [21] targets large-scale event notification applications such as stock market or sports event info dissemination.

Different classes of applications have different sets of requirements. Therefore they have different metrics. For example, in file transfer application, bandwidth is the only metric. In Video conferencing, bandwidth, latency and degree of interactivity are metrics. While in other applications, reliability and scalability are among the most important metrics.

2.2.2 Classifications

There are different ways to classify existing ALM protocols. From the point of the number of source exists in an application, there are single source protocols and multiple sources protocols. From the point of deployment level, there are end-system based and proxy-based ALM protocols. From the point of membership management mode, there are centralized approach and distributed approach. From the point of the overlay topology building algorithm, there are mesh-first, tree first and implicit approaches. In this section, we will give a brief introduction of existing ALM protocols.

Single Source and Multiple Sources

In some applications data are distributed from a single source (one-to-many), while in others, like Video conferencing and interactive E-learning applications, data can come from multiple sources (many-to-many).

Overcast [4] and OMNI [5] are single source ALM protocols. They support to construct a single tree rooted at the source for data delivery. Single source ALM protocol is simple both conceptually and in practice, as it offers reduced multicast functionality. However, a single source ALM protocol still can be used for multi-source multicasting. For example, in a distributed lecture that allows questions from the class, there is only one non-root sender active at a time. It is possible for the sender to unicast to the root, and the root then

perform true multicasting on behalf of the sender [4].

ZIGZAG [10], NICE [9], Narada [6], HMTP [3], and Yoid [11] are all multi-source ALM protocols. These protocols either support to construct a set of source-specific trees rooted at each source, or a shared tree shared by multiple sources. [10], [9], [6] belong to the former, while [3] and [11] belong to the latter. In the former approach, multiple sources are handled one source at a time. That is, at any particular time only one source is allowed to send data. While in the letter approach, a tree is shared by multiple sources. The branches of the tree, the data delivery paths, are bi-directional.

End-System Based and Proxy-Based

From the point of deployment level, whether an infrastructure support is needed to form the overlay, there are infrastructure level protocols and end-system level protocols. Infrastructure level protocols are also called proxy-based protocols. In these protocols, dedicated proxy servers are deployed where they self-organize into an overlay backbone and provide a transparent multicast service to end-systems. End-system level are also called peer-to-peer. These protocols assume only a unicast service from the infrastructure and expect end hosts to participate in providing the multicast service by taking on some package replicating and forwarding responsibilities. The two types of architectures are shown in figure 2.3.

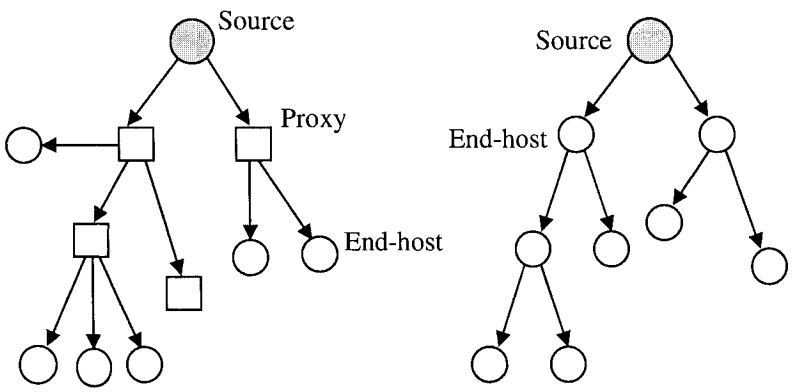


Figure 2.3 Proxy-based and end-system based ALM architectures

HMTP [3], Overcast [4], OMNI [5], and Scribe [21] are proxy-based protocols. In this model, proxy servers are pre-provisioned infrastructure nodes to perform multicast functions; just like in IP Multicast routers are infrastructure nodes to perform multicast functions. A receiver simply needs to connect to one of the infrastructure node to receive

streams. But since the supply bandwidth comes only from the provisioned nodes, these nodes are more likely to be bottleneck points. This means this model also has scalability problems [12].

NICE [9], Narada [6], ZIGZAG [10], DONet [13] and SplitStream [14] are end-system based protocols. This model uses only end-hosts participating in the application to perform multicast functions. In this model, since the supply bandwidth comes from the end-hosts, it is self-scaling. As more hosts join more bandwidth is supplied. However, end-hosts are transient and unreliable. The protocols must be complex to handle group dynamics. This means this model is only suitable to support small to medium sized groups [12].

Centralized and Distributed

In network layer multicast, the membership knowledge is distributed among multicast routers. Whereas in ALM, the membership knowledge can be maintained either by source, a rendezvous point (RP), or is distributed among members [2].

In the centralized approach, the membership knowledge is maintained in one node (root or RP). New member that joins should query the node first. A member that leaves should also inform the node. If there is an end system failure, it is this node's responsibility to update the membership knowledge, and then reconstruct the overlay topology. Failure of this node will cause part or all of the system functions to be blocked.

In contrast, in the distributed approach, either everybody knows everybody else, or the membership knowledge is maintained by some members. In the former case, that member joins/leaves is first detected by its neighbours, and then is propagated to the whole community. In the latter case, the situation is more complex and is different case by case. Some protocols use hierarchical membership management, whereas others use locally centralized but globally distributed approach. Compared with the centralized approach, the distributed approach is more robust because part of the system failure does not affect the whole system from functioning. Thus most of the existing protocols use the distributed approach.

Mesh-First, Tree-First and Implicit [1]

All application layer multicast protocols organize the group members into two topologies,

a control topology and a data topology. The control topology is for members to exchange periodic refresh messages to identify and recover from "ungraceful" departures from the group. The data topology is usually a subset of the control topology that identifies the data path for multicast data delivery. The control topology has greater connectivity between members than data topology. Therefore, the control topology is called a mesh and the data topology is called a tree. Depending on the sequence of construction of the control and data topologies, ALM protocols can be classified into three categories: mesh-first, tree-first and implicit approaches.

In the mesh-first approach, group members are first organized into an overlay mesh topology. Multiple paths exist on the mesh between each pair of members. Then a shortest path tree is calculated, where there is unique path between each pair of members. Narada [6] uses this approach to construct overlay.

In tree-first approach, a shared data delivery tree is first constructed. Then each member discovers a few other members that are not its neighbors on the overlay tree, and then establishes additional control links to these members. No matter a mesh control topology is constructed explicitly or not, the knowledge of other members is used for partition recovery. HMTP [3], Overcast [4], OMNI [5], and Yoid [11] belong to this category.

In implicit approach, the mesh and the tree are simultaneously defined by the protocol. The data delivery path is implicitly defined on the control topology by some packet forwarding rules. Therefore, there is no need for routing algorithm. NICE [9], Delaunay [7], CAN Multicast [8], Scribe [21] and ZIGZAG [10] all use implicit approach for overlay construction.

2.2.3 Related work

There is a plethora of ALM protocols, each designed for a specific scenario and application. A comprehensive discussion of such protocols is beyond the scope of this thesis; however, in this section we talk about some of them (especially HMTP, Overcast, and OMNI) that are relevant to our proposed protocol.

Narada [6] is an end-system based ALM protocol that targets small to medium sized groups (tens to hundreds of members) multiple sources applications such as Video

conferencing. It uses a mesh-first approach to build a multicast tree. More specifically, the protocol first abstracts the physical topology as a Complete Virtual Graph (CVG). Further, it constructs a set of source specific trees rooted at each source based on the CVG. Figure 2.4 depicts the mesh and tree building procedure in Narada.

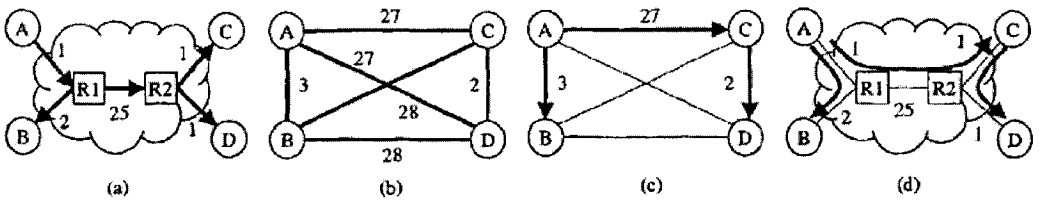


Figure 2.4 Mesh and tree building procedure in Narada

This protocol has two important features: 1) joining member exchanges refresh messages only with a subset of group members (its neighbors); 2) the mesh is improved by adding efficient links and dropping slow links dynamically. [29] shows how to implement Video conferencing applications by using Narada.

NICE [9] uses hierarchical management of nodes as an overlay building strategy. In NICE, nodes are clustered into groups. Nodes that physically close to each other are put into same cluster. And the head of the cluster is the most possible physical center of the cluster to the furthest extent. The hierarchy is created by assigning members to different layers as shown in Figure 2.5. All members in the overlay belong to the lowest layer, while heads of each layer form the above layer. The data delivery path is implicitly defined by each cluster head forwarding data to members of its cluster at every layer as shown in Figure 2.6. And there is no additional route computation required.

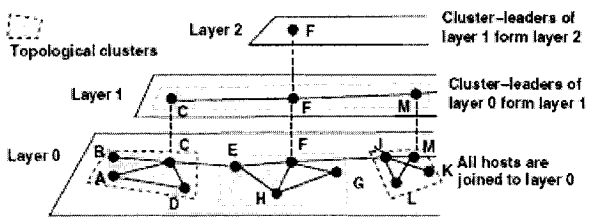


Figure 2.5 Hierarchical arrangement of hosts in NICE

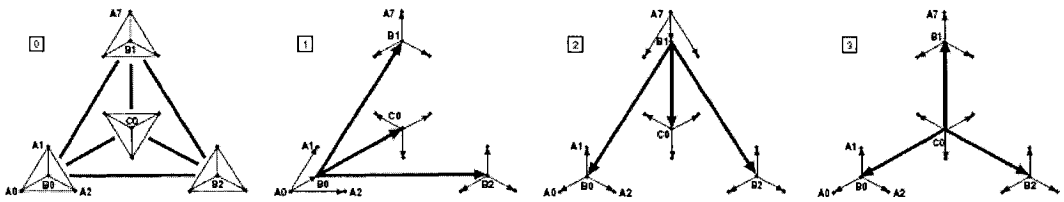


Figure 2.6 Control and data delivery paths for a two-layer hierarchy in NICE

In Figure 2.4, all A_i hosts belong to Layer 0. B_i hosts are heads of Layer 0 that form Layer 1. C_0 is the only host of Layer 2, which is the head of B_i hosts. Panel 1, 2, 3 are source-specific trees when the sources are at members A_0, A_7, and C_0 respectively.

Features of NICE are: 1) each member only knows few other nodes in the overlay; 2) each node can be source; 3) the protocol improves scalability and can support large sized groups.

Similar work has been done in **ZIGZAG** [10]. It attempts to reduce the number of nodes a head at high layer has to forward data to by using a different data forwarding rule. That is, the head forwards data to non-head nodes of foreign clusters of its subordinate layer. The multicast trees constructed by NICE and ZIGZAG are shown in Figure 2.7 [12]. This research team also proved that ZIGZAG has better performance than NICE by simulation tests.

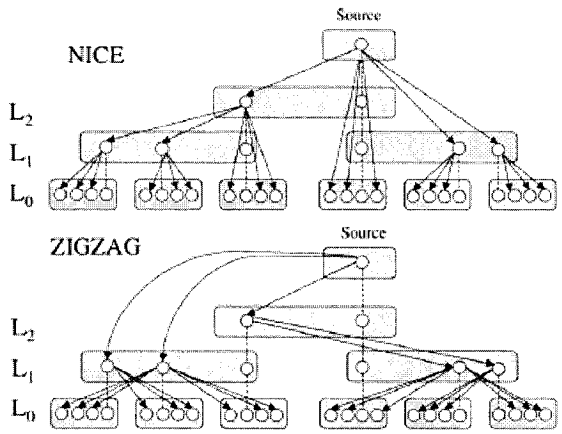


Figure 2.7 Data delivery paths in NICE and ZIGZAG

CAN [30] maps nodes in the Internet as coordinates in a 2D plane that can be stored in hashtables and can be queried later. This is called Content Addressable Network. Each

member on the CAN owns a portion of this coordinate space. Each node has information about adjacent nodes. The data topology is implicitly defined by performing directed flooding on the control topology. As shown in Figure 2.8, nodes on the CAN just forward data to their neighbors until all nodes get data packages. CAN Multicast [8] is an ALM protocol that is built on top of CAN.

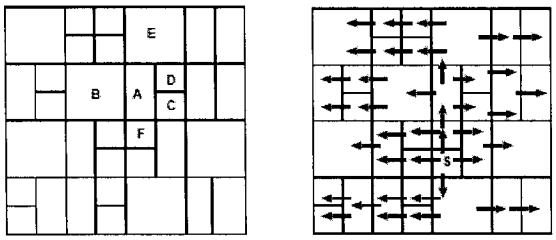


Figure 2.8 CAN with 2D coordinate space: control and data topologies

Delaunay [7] is another protocol using implicit approach to construct overlay network. In Figure 2.9 left panel, node A considers B as its parent and C as its child in the tree with root R only if angle $\angle$RAB is smaller than angle $\angle$RAC. By this way, each node in the overlay can locally determine its child nodes with respect to a given tree using the coordinates of its neighbors and the coordinates of the sender. Therefore there is no need for an extra routing algorithm. The coordinates are assigned via some external mechanism (e.g., GPS, user input). As long as the coordinates reflect the geographical locations of nodes, this algorithm shows an acceptable latency. And there is no network delay measurements required to establish a Delaunay Triangulation overlay network.

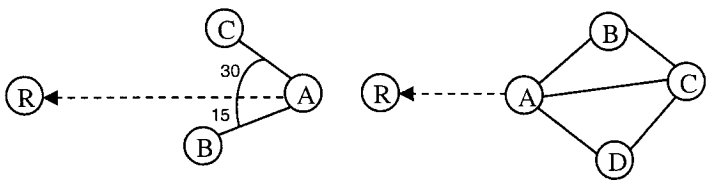


Figure 2.9 Routing in the Delaunay triangulation overlay

HMTP [3] attempts to connect IP Multicast islands by using application layer multicast. Since IP Multicast is still efficient in local area network, the protocol tries to connect such IP Multicast islands by using unicast tunnels, that is, connect the representatives of such islands into a shared tree. In building the tree, delay is the primary metric. A joining node chooses the closest node on the tree as its parent. Bandwidth is another metric. The

algorithm targets large scale multicast applications. The system architecture is shown in Figure 2.10.

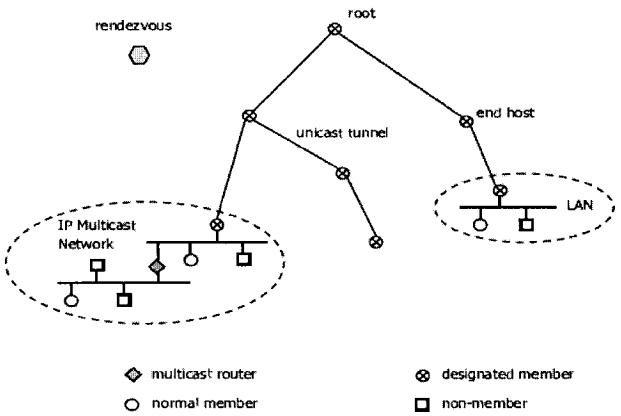


Figure 2.10 Host multicast architecture

Similar work is done in **Yoid** [11]. They all use tree-first approach but Yoid constructs a mesh later on and the mesh is used for partition recovery.

Overcast [4] is a proxy-based single source ALM protocol which targets non real-time applications such as bulk data transfer, where the primary metric is bandwidth. In building the tree, a new node tries to find a node as far away to the root as possible without sacrificing the bandwidth to the root.

OMNI [5] is another proxy-based single source protocol. The proxies are called MSNs (Multicast Service Nodes) in OMNI. The system architecture is shown in Figure 2.11. First the source and the MSNs are self-organized into an overlay network. The MSNs join the tree in an increasing order of distance to the source. Then five different transformations are introduced to reshape the tree to minimize global average latency.

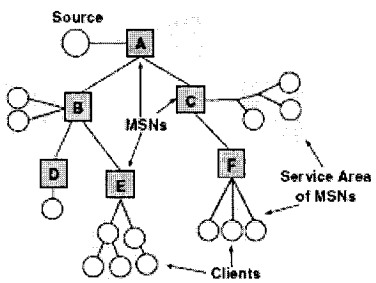


Figure 2.11 OMNI architecture

The objective of the above two proxy-based protocols is to organize the infrastructure nodes into an overlay before data delivery commences. A client's host just needs to connect directly to an infrastructure node to receive the service. In OMNI, at different level of the tree, the MSN serves different number of clients.

2.2.4 ALM Metrics and Evaluation Criteria

ALM protocols also bring us some new metrics and evaluation criteria. In this section, we will give a summary to these metrics and criteria [1].

The first metric is ***Stress***. It is defined per link of the topology and counts the number of identical packets sent by the protocol over that link or node. For network layer multicast there is no redundant packet replication and hence the stress metric is one at each link or node of the network.

The second metric is ***Stretch***. It is defined per-member and is the ratio of the path length along the overlay from the source to the member, to the length of the direct unicast path. Clearly, a sequence of direct unicasts from the source to all the other members has unit stretch for each member.

The third metric is ***Control Overhead***. Each member on the overlay exchanges refresh messages with all its peers on the overlay. These messages constitute the control overhead at the different routers, links and members of the multicast group. The control overhead is an important metric to consider from the scalability standpoint.

There are also some other metrics like bandwidth and latency that characterize ALM protocols' performance.

To evaluate ALM protocols, we follow the non-functional evaluation criteria listed in [2].

Ease of Deployment

An ALM solution should be easy to deploy with no infrastructure support. The easier the deployment, the more universally it can be employed. Research on issues that involve Network Address Translation (NAT) and firewalls still needs to be carried out.

Robustness

The solution should be robust when the topology is partitioned by a single node failure. A fast detection and recovery mechanism is required. Sometimes redundancy is added to make the overlay topology more robust, at the cost of increasing control overheads.

Security

This is a point neglected by most of the existing ALM proposals, which is especially important in centralized approaches, where a failure of RP will impair the whole system from function. This aspect needs to be addressed more in the future work.

Performance

The most basic point of an ALM solution is whether it can build an efficient overlay topology for data delivery. However, improvements for robustness, security and scalability may affect final performance. A tradeoff needs to be found between these properties and performance, cost.

Scalability

Scalability can be broken down into three dimensions: number of multicast groups, number of receivers per group, and number of senders per group [12]. The significance of scalability depends on applications. Some applications have higher needs for scalability over others. Scalability is often achieved by a hierarchical overlay topology and distributed membership knowledge.

Dynamic Discovery of Sources and Receivers

ALM solutions should be able to discover sources and receivers immediately and at the same time limit useless traffic.

Chapter 3

The Proposed Protocol: ProBaSS[1]

In this section, an ALM protocol for single-source media streaming applications is proposed. The application of such a protocol is for scenarios where one source is "broadcasting", in the television sense, news, sporting events (such as a hockey game), or a movie, to a large number of receivers. This protocol uses the Proxy-based approach described in the previous chapter, and is referred to, in this thesis, as ProBaSS: Proxy-Based Single Source protocol.

3.1 Algorithm

The proposed protocol – ProBaSS – is a proxy-based single source ALM protocol. The overlay multicast architecture of ProBaSS is shown in Figure 3.1.

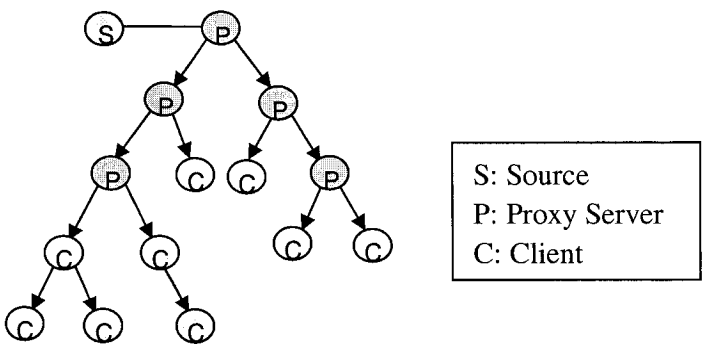


Figure 3.1 The overlay multicast architecture of ProBaSS

The source provides media content. The proxy servers are pre-provisioned infrastructure nodes which provide media distribution services. They are provided by the content providers. The clients are end users. They are the consumer of the service.

In this overlay multicast architecture, on the one hand, dedicated proxy servers are used where they self-organize into an overlay backbone and provide multicast services to end-systems; on the other hand, end-systems also take the responsibilities of forwarding

[1] Pronounced Pro-Base.

data to other end-systems.

Assume that the source and proxy servers have already been organized into an overlay backbone before data delivery commences. The algorithm of ProBaSS is described below.

1. Rationale

   In building the tree, the first metric is the out degree constraint. Each node of the tree (either proxy or end host) has an out degree bound. Out degree means the maximum number of children nodes a parent node can handle. Out degree reflects the maximum bandwidth each node can provide.

   The second metric is the end-to-end distance metric. The member-to-member round trip time (rtt) can serve as a distance metric in building the tree.

2. Member join

   A new member joining a group first queries the proxy server, the server sets itself as its potential parent and sends a list of its children (and itself if it still has void out degree) to the new member. The new member then queries each node on the list to measure the distance by measuring round trip time to find a nearest node. The proxy then sets the nearest node as the new member's potential parent and sends a list of the new potential parent and its children to the new member. This process is repeated until the nearest node remains to be the potential parent or the nearest node is a leaf node. The proxy server then sets this node as parent of the new member and adds the new member to the multicast tree. Figure 3.2 [1] shows how the member join algorithm works via an example. Here, the out degree is 3.

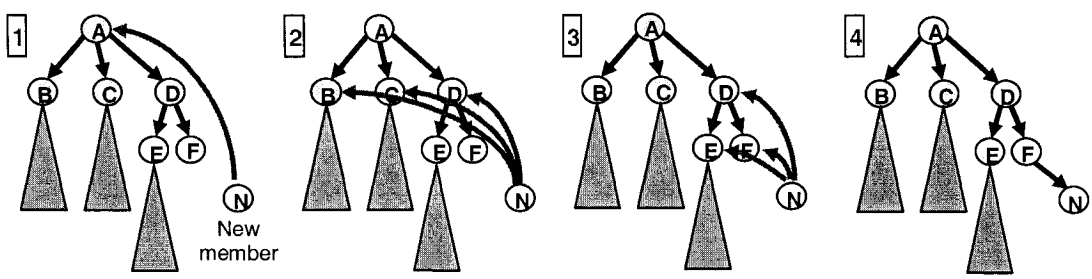


Figure 3.2 Member join algorithm

In Figure 3.2, N is the newcomer. By querying the source, N learns that A is the nearest

proxy.

1. N sends join request to A. Since A has no available degree, A sets itself as potential parent of N, and then responds with its children list: B, C, D.
2. N queries B, C, D. N finds D is the nearest node.
3. D is set as its new potential parent and N queries D, E, F afterwards. This time N finds F is the nearest node.
4. Since F is a leaf node, the proxy sets F as its parent and adds new comer N to the multicast tree.

3. Refresh Message Exchange

   After joining a group successfully, each member sends refresh messages to the proxy server periodically. The proxy server updates its report time, checks the status of its neighbors (parent, children), and then responds with the status of its parent. This periodic message exchange is used to identify "ungraceful" departures from the group.

4. Member leave

   A member leaving the group first notifies the proxy server. The member is permitted to leave immediately if it is a leaf node. If it is not a leaf node, the proxy first breaks the links between itself and its neighbors, and then notifies its children by refresh message exchange. Its children then initiate member join requests to the server. The member join requests are processed just like new member join requests. The member is permitted to leave after all its children joined new parents. The proxy then deletes the node from the multicast tree. Figure 3.3 illustrates member leave algorithm.

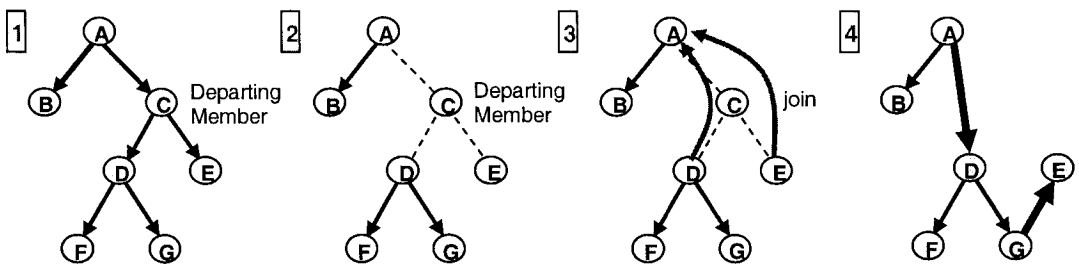


Figure 3.3 Member leave algorithm

5. Member leave without informing group

   When a non-leaf node leaves without informing the group, the tree is partitioned. The

partition is detected by noticing repeatedly missing a member's refresh message, and is handled similarly to member leave requests. That is, the proxy deletes the node from the multicast tree, notifies its children about its departure, and then the children initiate member join requests to the server.

6. Routing

   During the periodic refresh message exchange, the proxy server sends each node a children list. The streaming channel of the node reads this info and forwards steaming data to its children according to the list when it receives data.

7. Race condition solution

   That members join and/or leave simultaneously may cause race condition. To solve these synchronization problems, a queuing mechanism is introduced. New member join requests are put into a queue. Member leave requests do not need to be put into the queue, for a non-leaf member leave request may cause member join requests consequently. Member join requests in partition recovery have higher priority over new member join requests and are put into a high priority queue. This is to guarantee that a new member always joins a repaired tree.

Note that the proxy servers also use the same algorithm described above to form the overlay backbone.

In addition, HMTP [3] is an important reference in the building of ProBaSS.

3.2 Theoretical Evaluation

In building the tree, the member join algorithm takes end-to-end distance as metric. It supports building an efficient multicast tree which is congruent to the underlying network topology to furthest extent. A well organized shortest path tree rooted at source is efficient for both control and data delivery. The member rejoin algorithm is consistent with new member join algorithm, that is, the partitioned member seeks parent from the root. This helps to maintain the latency metric.

In repairing the tree, instead of all the descendants of the departing member seeking for new parents respectively, the algorithm only requires those nodes who are direct children

of the departing member to seek for new parents. Once a child node rejoins, its all descendants formed sub-tree rejoins. This reduces the broken links caused by member departure and reduces time for the system to come back to stable status, therefore minimizes the effect causing by member departure.

The overlay multicast architecture combines advantages of both infrastructure architecture (more reliable) and peer-to-peer architecture (self-scaling as more peers join more bandwidth supplies). It also eliminates disadvantages of both architectures. On the one hand, in infrastructure architecture proxy servers are often bottleneck points, but in our system they are not, because some end-systems take over part of the workload from the proxy to forward data to other end-systems. On the other hand, a peer-to-peer architecture has the drawbacks of: 1) always unreliable since self-organized autonomous peers are too transient; and 2) not scaling well to support large sized group, since membership management is complex to handle group dynamics. Compared with P2P systems, our system has the merits of better reliability because the proxy servers are always there to handle member dynamics; and better scalability because proxy servers help to reduce management complexity therefore enhance system performance.

Moreover, a locally centralized but globally distributed approach is adopted. A centralized approach simplifies membership management, therefore reduces redundant traffic and control overheads. On the other hand, a distributed approach has the advantage of better robustness. In our system, membership knowledge is maintained by proxy servers. Therefore from the point of a single proxy server and its serviced area it is centralized, but from the system's point of view it is distributed. Failure of parts does not affect the whole. In case one proxy is down, its managed hosts can be switched to other proxies.

Chapter 4

System Design

In this section we present our system design following the Unified Modeling Language (UML) version 1.4 standard. The diagrams in the system design are drawn by using the UML tool Rational Rose Enterprise Edition.

4.1 System Architecture

The system architecture is shown in Figure 4.1.

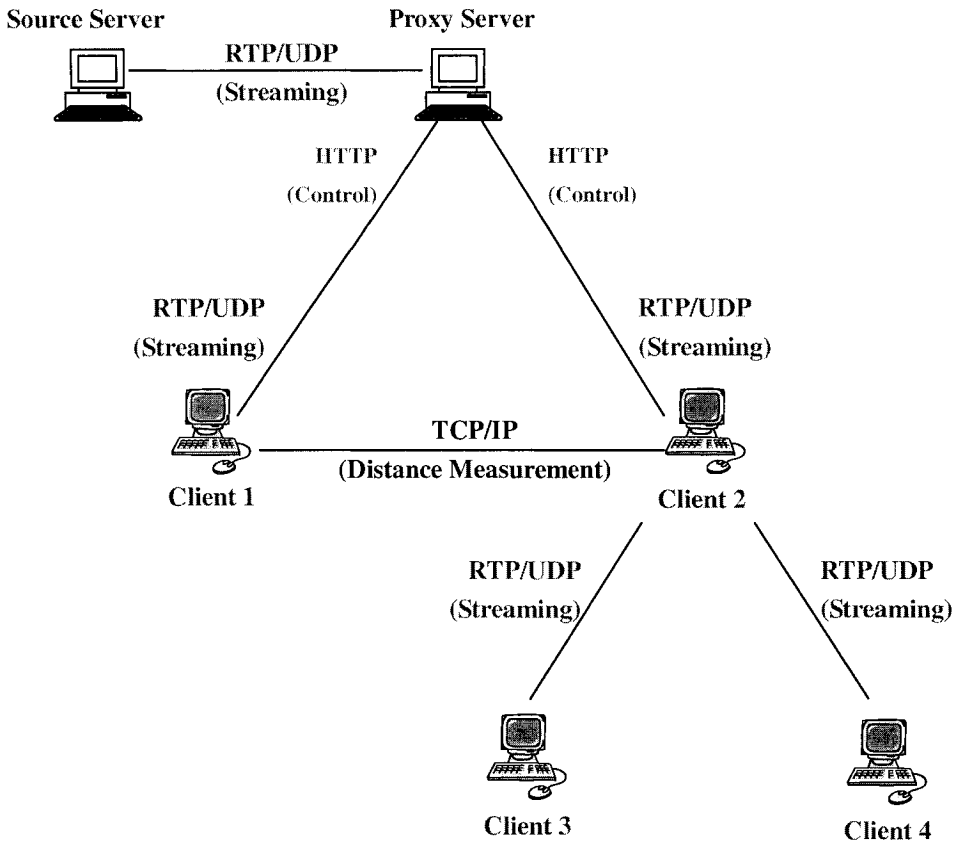


Figure 4.1 System architecture

We use HTTP for multicast control between proxy server and clients, use TCP/IP between clients for end-to-end distance measurements, and use RTP (Real-Time Transport Protocol)

and RTCP (Real-Time Control Protocol) over UDP for streaming between source and proxy, proxy and client, and between clients. In addition, because the proxy server itself is also a node in the multicast tree, it is also a client, so each client also has a TCP/IP connection with the server for this reason. Figure 4.2 shows the protocols used by the system.

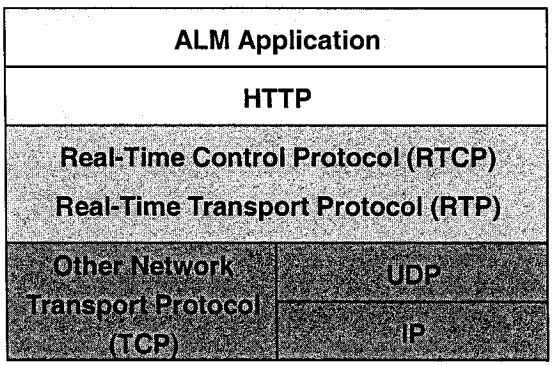


Figure 4.2 Layered system architecture

The proxy server runs the Tomcat Server to provide multicast service to clients, who use browsers to access such service. The server side programs are Servlets or JSP pages, while the client side programs are Applets embedded in HTML pages. There are three types of connections between server and client: HTTP, TCP/IP and RTP/UDP, which are described above. These are illustrated in Figure 4.3.

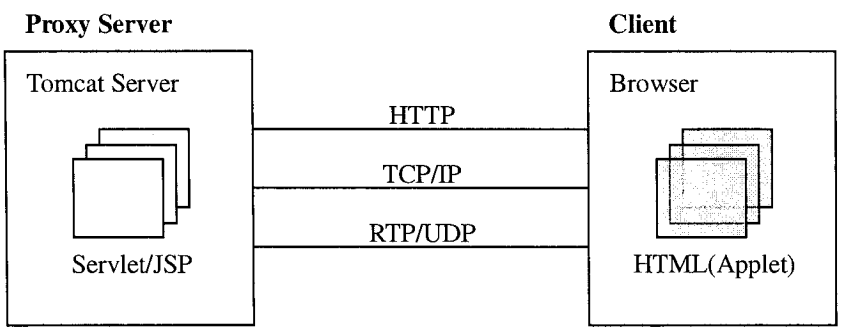


Figure 4.3 Client-Server architecture

4.2 Use cases

The use case diagram is shown in Figure 4.4.

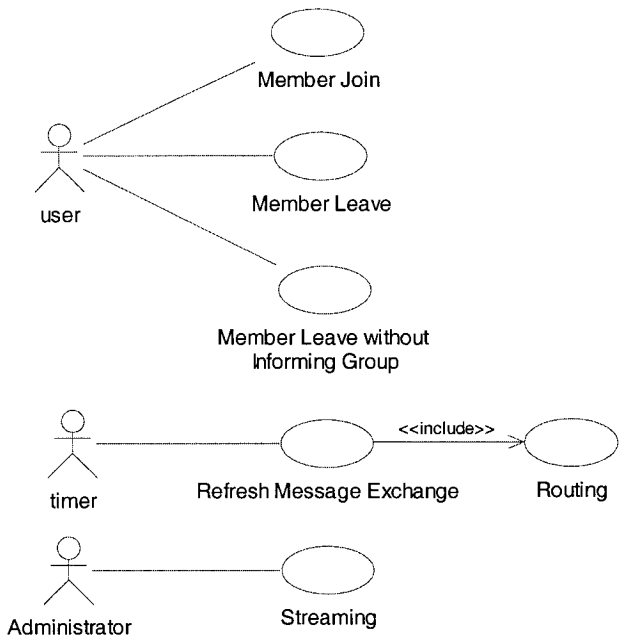


Figure 4.4 Use case diagram

The use cases are described below:

Member Join:

Actor: user

Description:

1. The user joins the multicast group.
2. The system measures the distance between the client and all existing nodes in the overlay, chooses one node as the new member's parent and adds the client to the multicast tree.
3. The system responds by starting to play media.

Member Leave:

Actor: user

Description:

1. The user sends leave request to the multicast group.
2. The system deletes the node from the multicast tree and finds new parents to its children if it is not a leaf node.
3. The system responds with a message shows that the user has quit the multicast group

successfully.

Member Leave without Informing Group:

Actor: user

Description:

1. The user leaves the multicast group without sending leave request.
2. The system detects the departure of the node, deletes the node from the multicast tree and finds new parents to its children if it is not a leaf node.

Refresh Message Exchange (Routing):

Actor: timer

Description:

1. Each node sends refresh messages periodically to the proxy server as soon as it joins multicast group.
2. The system records the report time, checks the status of its neighbors, responds with dynamic routing info if its parent is still alive, or with parent died response if its parent has left or is waiting to leave.
3. The client initiates a member rejoin request if its parent died. It waits for the server to process the request until it receives rejoined response.

Streaming:

Actor: administrator

Description:

1. The administrator initiates sessions for media steaming.
2. Each user watches media playing as soon as he/she joins the multicast group.

Then the system is decomposed into three sub-systems: source, proxy and client. At the proxy sub-system, a MemberJoin Agent is introduced to process member join/rejoin requests and a Maintain Agent is introduced to process member leave requests and reports from clients. The use case diagram of sub-systems is shown in Figure 4.5.

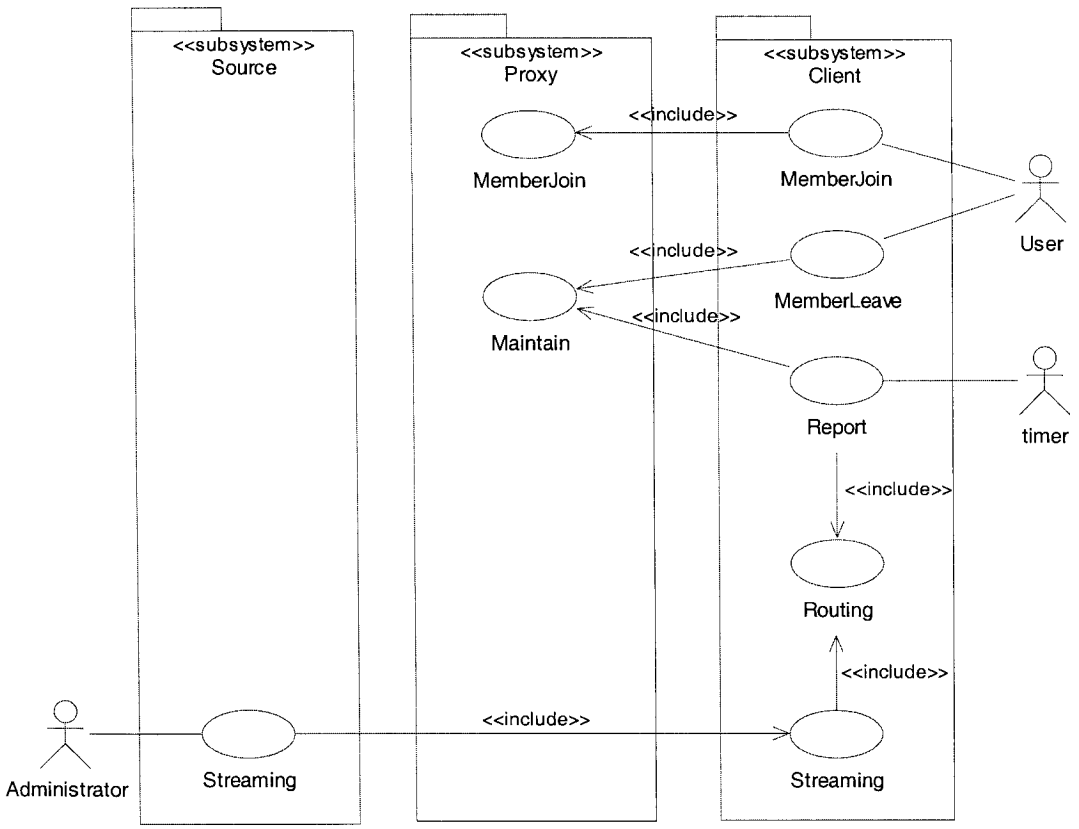


Figure 4.5 Use case diagram of sub-systems

4.3 Static Model

The classes of the proxy sub-system are shown in Figure 4.6. A hashtable is used to store the routing table. Each node of the overlay network has a record in the table. The key is the IP address and the value is an object of NodeInfo class. The object contains the node's IP address, IP addresses of its parent and its children, the last refresh time and its status ("alive" for normal status, "dead" when the node has left or is waiting to leave). In addition, a queueing mechanism is introduced for race condition control. More specific, a queue is used to store new member join requests and a high priority queue is used to store member rejoin requests. The MemberJoin Agent just puts users' requests into proper queues. Another Servlet called ProcessJoinRequest Agent is introduced to process these requests.

Figure 4.7 shows an example of a multicast tree and its object diagram.

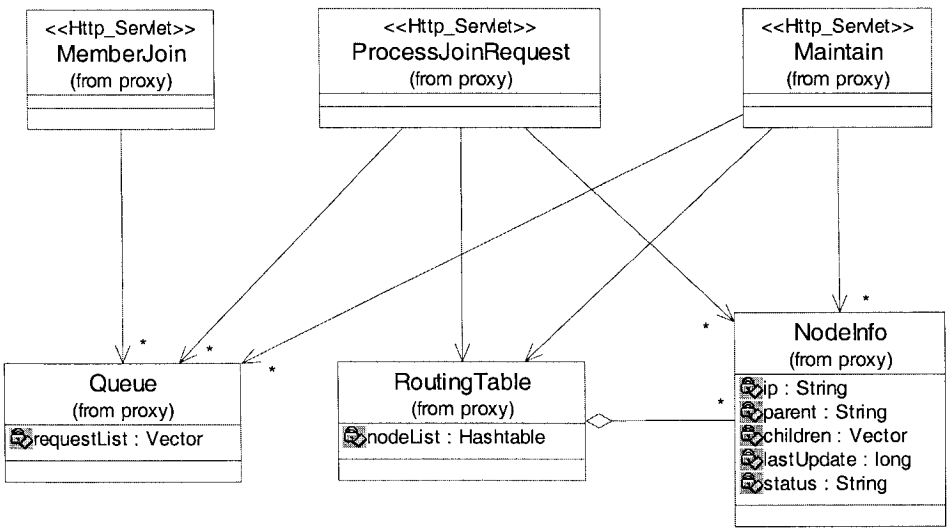


Figure 4.6 Class diagram for proxy server sub-system

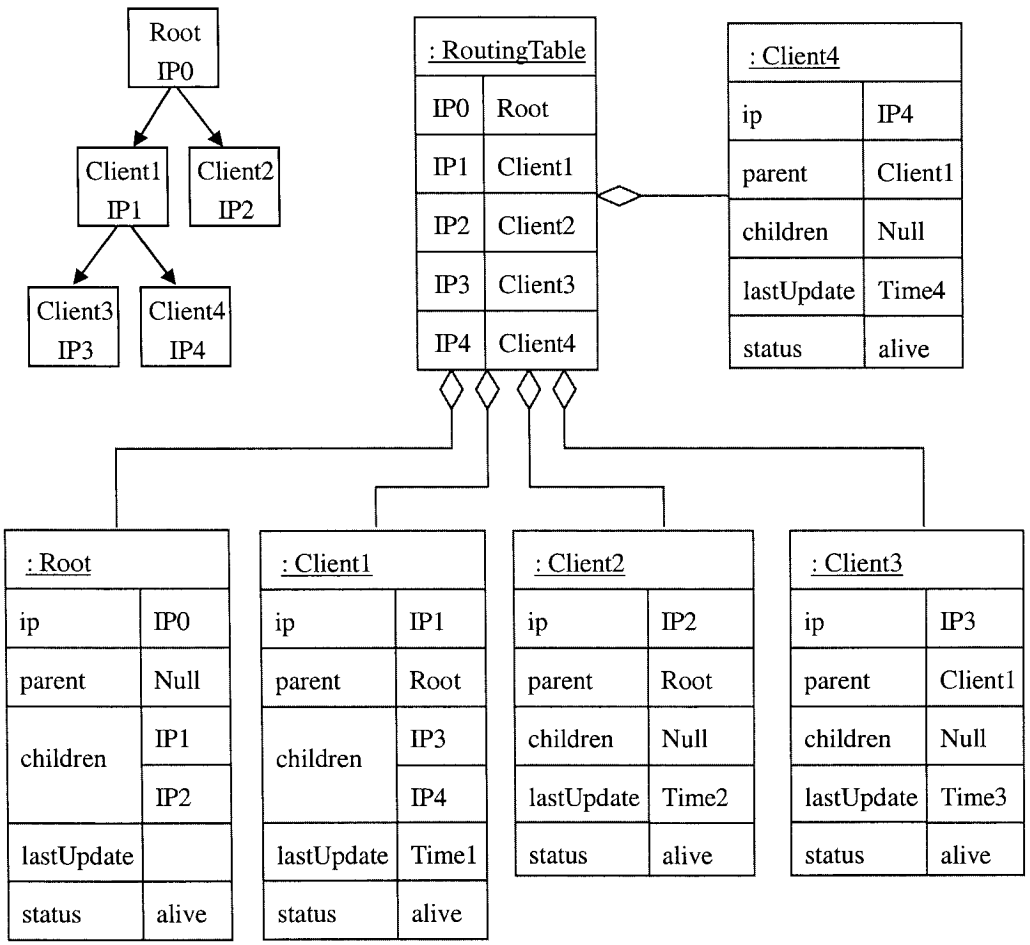
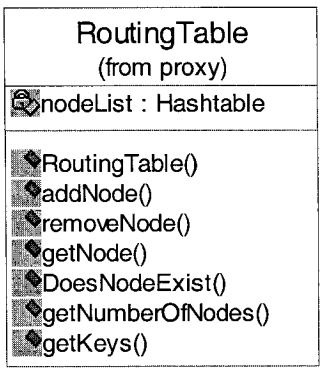


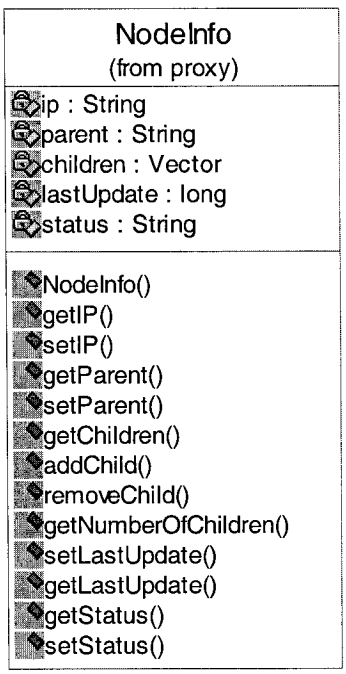
Figure 4.7 An example of a multicast tree and its object diagram

The detailed design of classes (attributes and operations) is listed below:

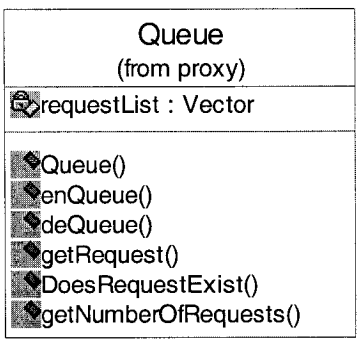
```
RoutingTable
- nodeList:Hashtable
+ RoutingTable()
+ addNode(node:NodeInfo)
+ removeNode(ip:String)
+ getNode(ip:String):NodeInfo
+ DoesNodeExist(ip:String):boolean
+ getNumberOfNodes():int
+ getKeys():Enumeration
```




```
NodeInfo
- ip:String
- parent:String
- children:Vector
- lastUpdate:long
- status:String
+ NodeInfo()
+ getIP():String
+ setIP(ip:String)
+ getParent():String
+ setParent(parent:String)
+ getChildren():Vector
+ addChild(ip:String)
+ removeChild(ip:String)
+ getNumberOfChildren():int
+ setLastUpdate()
+ getLastUpdate():long
+ getStatus():String
+ setStatus(status:String)
```




```
Queue
- requestList:Vector
+ Queue()
+ enQueue(ip:String)
+ deQueue()
+ getRequest():String
+ DoesRequestExist(ip:String):boolean
+ getNumberOfRequests():int
```



The classes of the client sub-system are shown in Figure 4.8. All the classes in the client package are Applets embedded in the HTML pages.

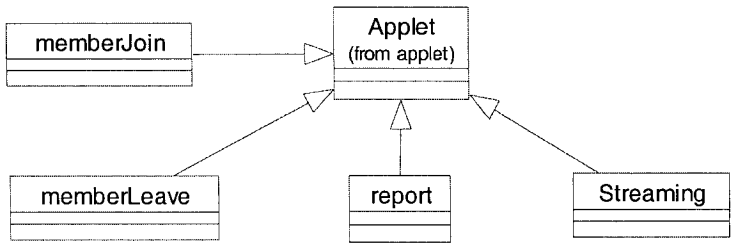


Figure 4.8 Class diagram for client sub-system

The package diagram of the system is shown in Figure 4.9.

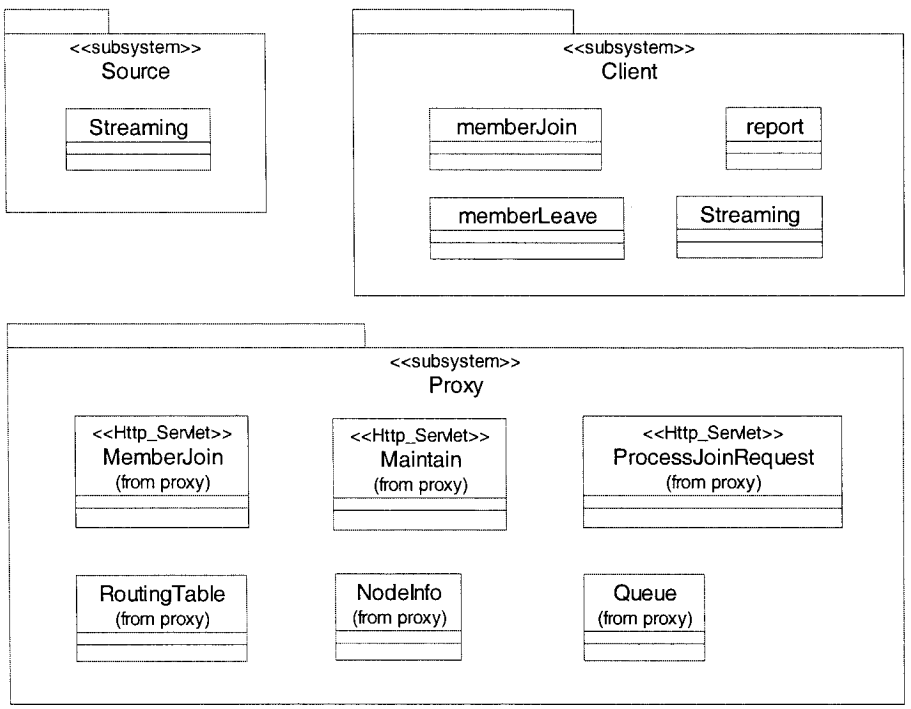


Figure 4.9 Package diagram

Additionally, because the proxy server itself is also a node of the multicast tree, it runs a proxy package as well as a client package. The proxy side client package is a little bit different from the client side client package. This is to be discussed in Chapter 5. Figure 4.10 illustrates packages at source, proxy server and client sub-systems.

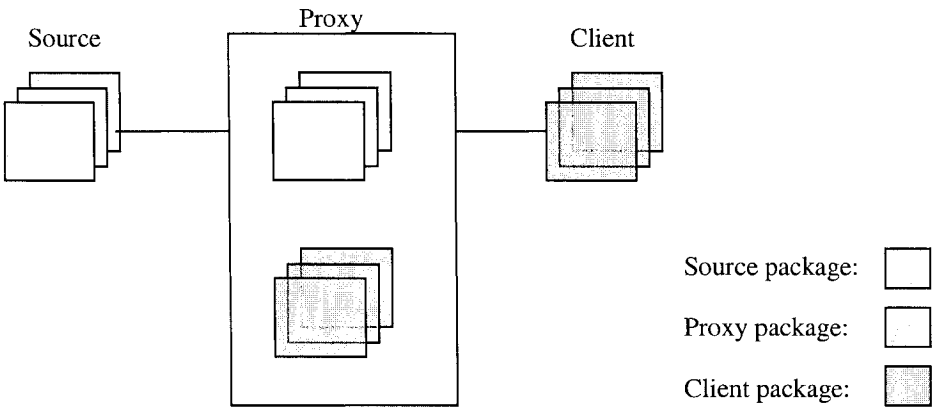


Figure 4.10 Packages of each sub-system

4.4 Dynamic Model

1 Member Join:

The communication between client and proxy in member join use case is defined as follows:

1. When client sends member join request, the MemberJoin Agent puts it in a queue, then calls ProcessJoinRequest Agent to process the request.
2. If it is the first member coming, it is added as child of the server into the tree.
3. If it is not the first member coming, the server first sets root as its potential parent, then checks the out degree bound and sends an IP list of potential parent and its children to the client
4. The client then sends measure rtt request to each host on the list. Server or end-host who receives the "MeasureRttRequest" just returns "MeasureRtt Response" directly to the client. The client then measures the distance to each hosts on the list to find a nearest node
5. If the nearest node is potential parent or the nearest node is a leaf node, then the new node is added as child of the nearest node, else the nearest node is set as new potential parent and steps 3 to 5 are repeated until the nearest node is the potential parent or a leaf node.
6. The server adds the new node to the routing table and sends "ready to play media" response to client.

Figure 4.11 is the activity diagram of member join use case, which leaves out some steps during the process to make it simple and clear. Process member rejoin request is

similar to process new member join request.

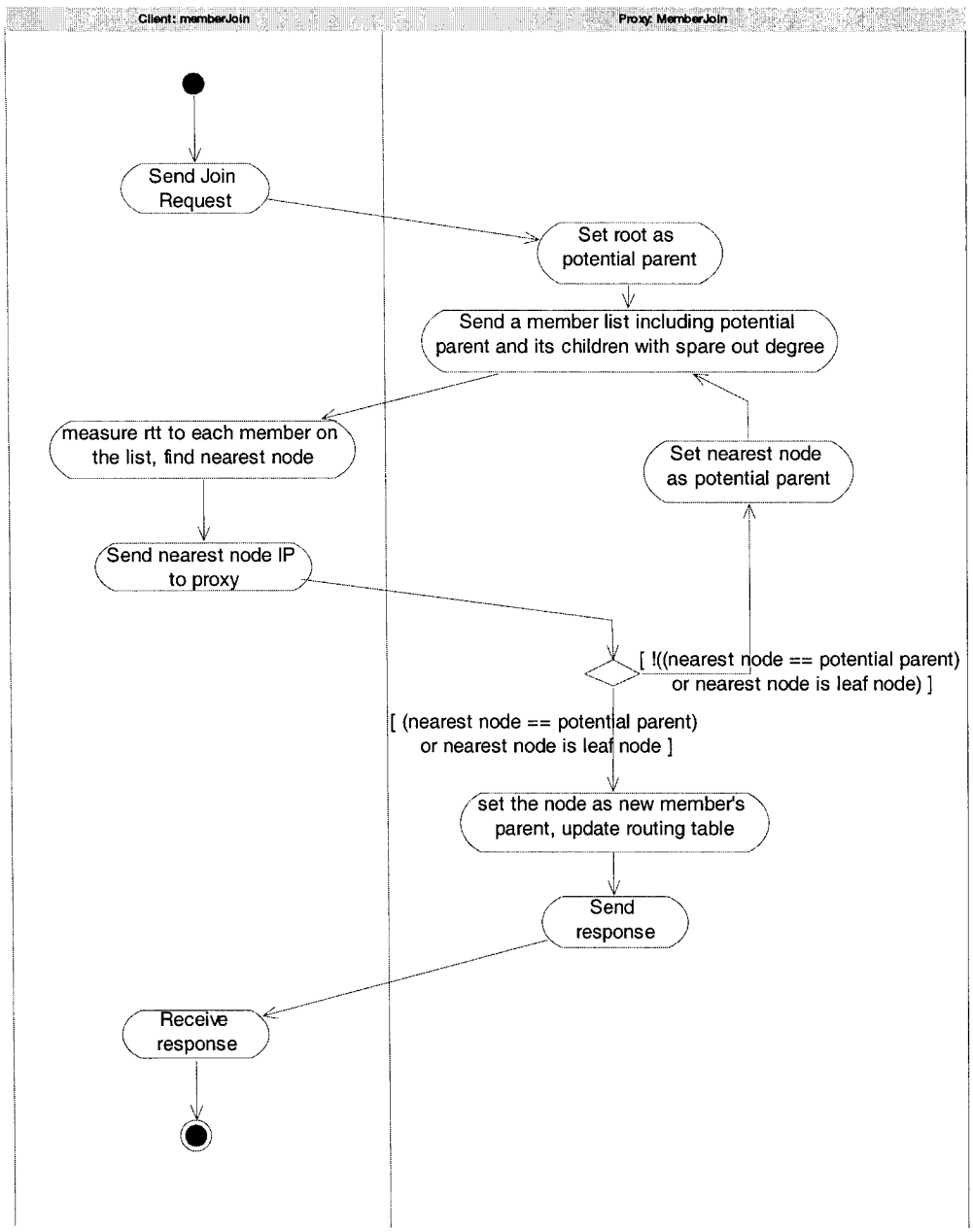


Figure 4.11 Activity diagram of the member join use case

2 Refresh Message Exchange (Routing):

The communication between client and proxy during refresh message exchange is defined as follows:

1. When a node joins the multicast group successfully, it starts to send refresh message to the Maintain Agent periodically. There are two types of client report: "normal report" and "member leave request".
2. When the Maintain Agent receives "member leave request" from the client, it first breaks the link between the node and its parent, and the links between the node and all its children from the routing table, and then marks the status of the node as "dead". Then it sends "waiting response" to the node. The node keeps sending "member leave request" to the Maintain Agent for a period of time until the Maintain Agent sends "leave permitted response". Then the node receives "quit the system successfully" page.
3. If a node leaves without informing the group, when one of its children report to the Maintain Agent, the Maintain Agent checks its parent's last refresh time and finds that the Maintain Agent has missed report from its parent for a period of time (e.g.: 3 times report interval), then it breaks the link between the parent and the grandparent, and then sets all children of the parent node as no parent.
4. When the Maintain Agent receives "normal report" from the client, it modifies the last refresh time of this client, and then checks the status of its parent and its children on the routing table. If its parent is still alive, the Maintain Agent then sends "normal response" with routing info to the client. If its parent died or the node has no parent currently, the Maintain Agent sends "parent died response" to the client. Receiving such response, the client should initiate a "member rejoin request" to the MemberJoin Agent accordingly.
5. When the MemberJoin Agent receives a "member rejoin request", it directly puts the request into the high priority queue. Another program checks if it is the first request in the queue. If it is, it is processed immediately, else it sends "waiting response" to the client. Receiving such response, the client keeps sending the request until the request is processed and a "rejoined response" is received.
6. When the server itself reports, the Maintain Agent checks all the nodes in the routing table and deletes those nodes that do not report for a period of time (e.g.: 5 times report interval). The Maintain Agent also checks the queues and deletes those requests that do not be processed for a period of time. This is to avoid deadlock.

The activity diagram of refresh message exchange (routing) use case is shown in Figure 4.12.

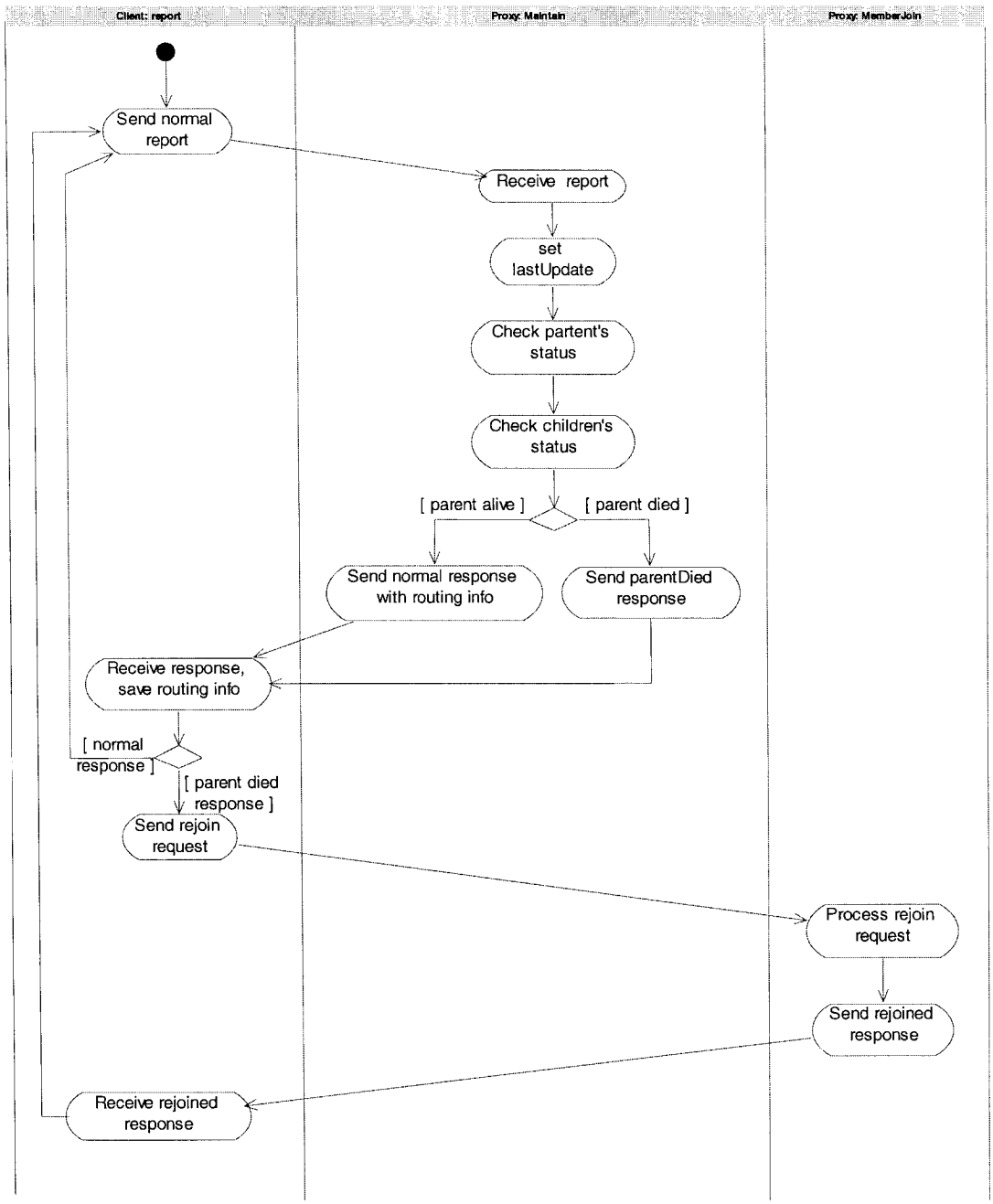


Figure 4.12 Activity diagram of the refresh message exchange use case

Figure 4.13 and 4.14 are sequence diagrams of refresh message exchange. Figure 4.13 shows "NormalReport" with "NormalResponse" case, while Figure 4.14 shows "NormalReport" with "ParentDiedResponse" case.

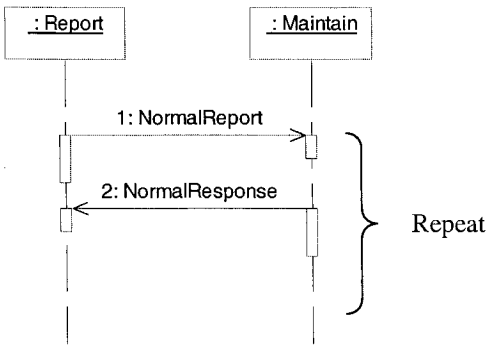


Figure 4.13 Sequence diagram of the refresh message exchange use case (1)

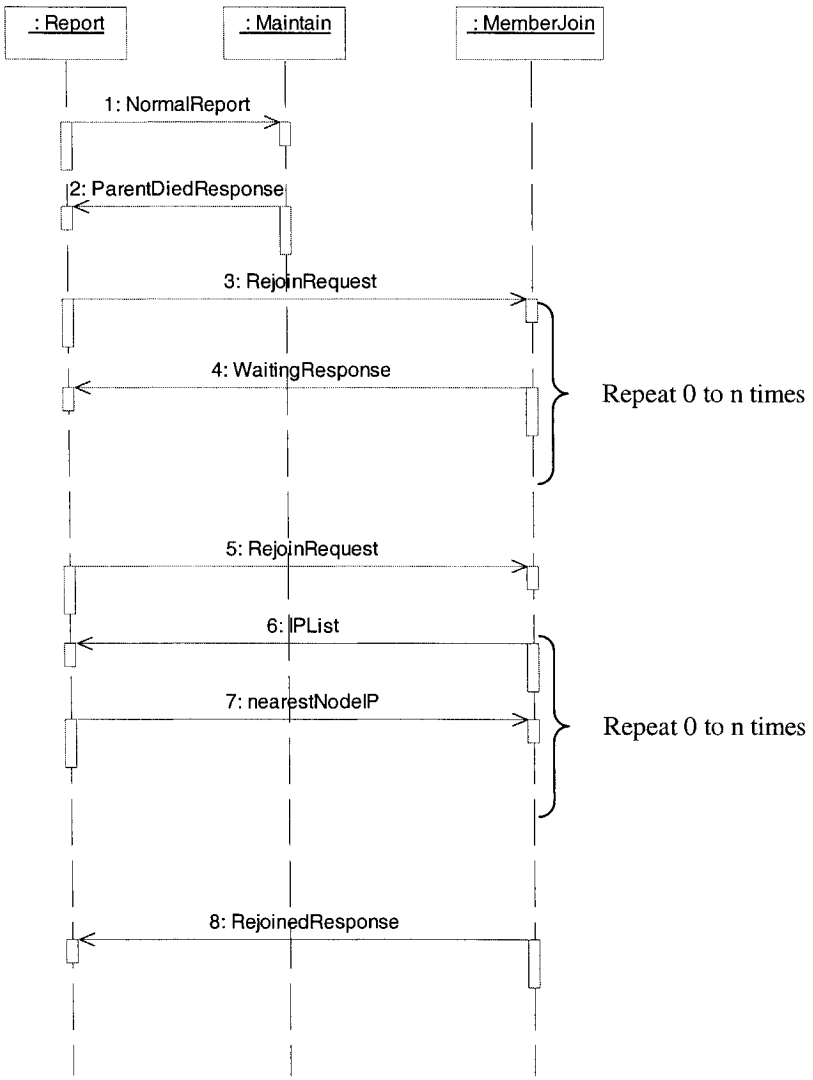


Figure 4.14 Sequence diagram of the refresh message exchange use case (2)

3 Member Leave:

The communication between client and proxy in member leave use case is defined as follows:

1 When the Maintain Agent receives "member leave request" from the client, it first breaks the link between the node and its parent, and the links between the node and all its children from the routing table.

2 If it is a leaf node, it is directly deleted from the routing table. The node then receives "quit the system successfully" page.

3 If it is a leaf node, the node is the marked as "dead", and then receives "waiting response".

4 Each time the node sends "member leave request" again, the Maintain Agent checks its children nodes. If one of its children has not rejoined the multicast tree yet, the node receives "waiting response" again.

5 After all its children rejoined the tree the node receives "quit the system successfully" page.

Figure 4.15 is the activity diagram of member leave use case, and Figure 4.16 shows the message sequence during member leave.

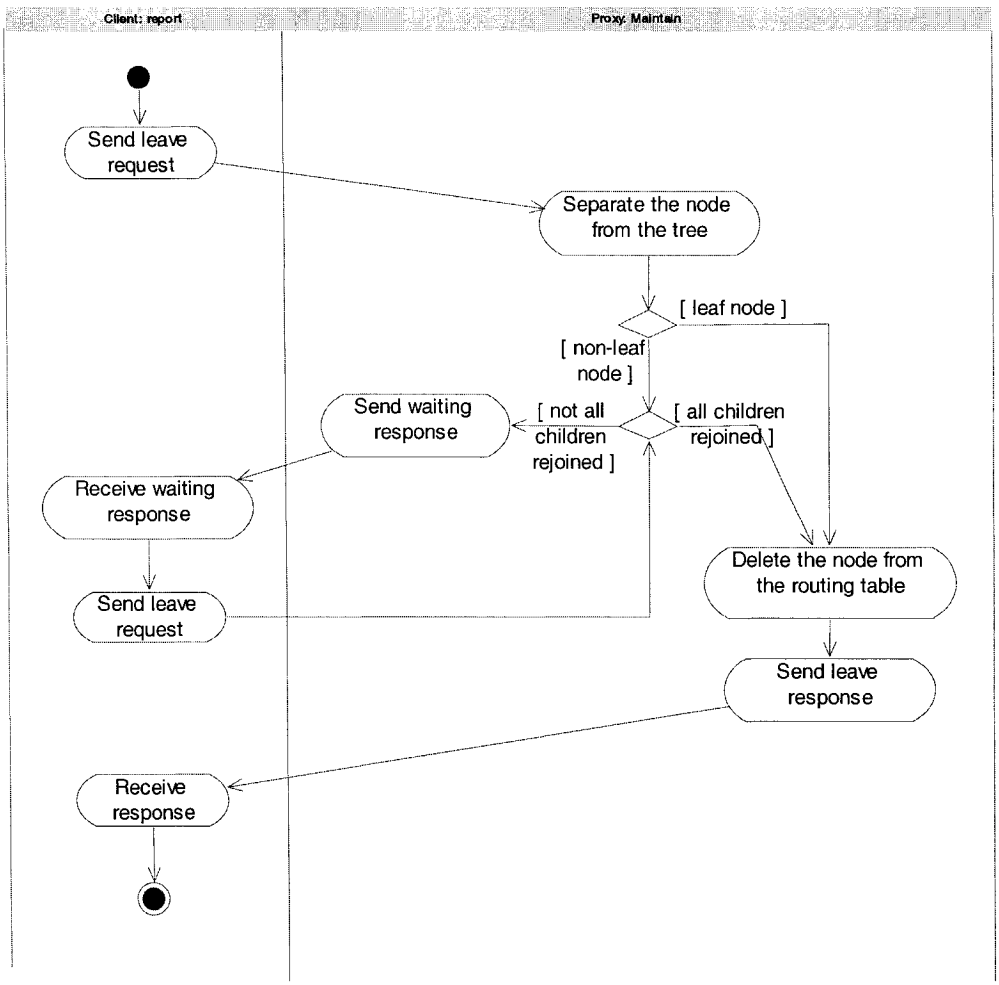


Figure 4.15 Activity diagram of the member leave use case

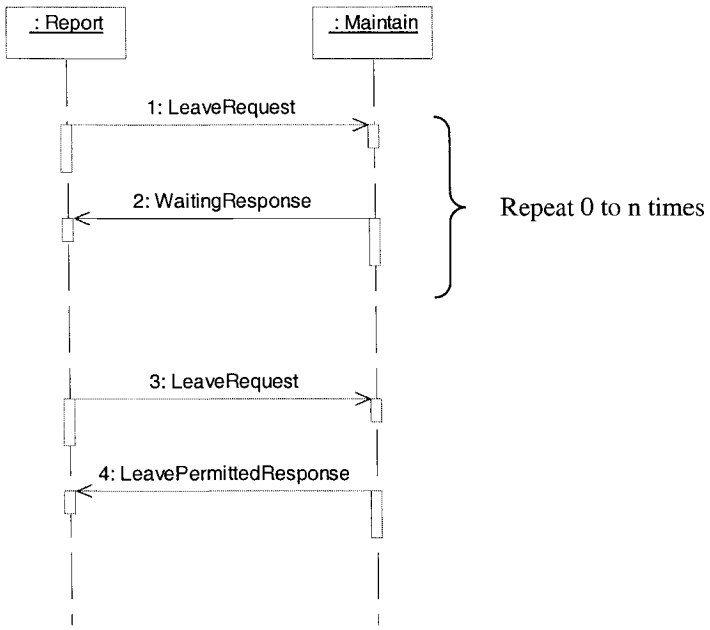


Figure 4.16 Sequence diagram of the member leave use case

4 Member Leave without Informing Group:

There is no activity or message exchange in this use case. The departure of the node is detected by its neighbors (parent, children) and is handled by refresh message exchange use case.

5 Streaming:

This use case is to be discussed in Chapter 6.

Chapter 5

Implementation

5.1 System Flow Chart

The system flow chart is shown in Figure 5.1. The proxy package has three agents, while the client package has five pages embedded with applet classes. They are listed in Figure 5.1.

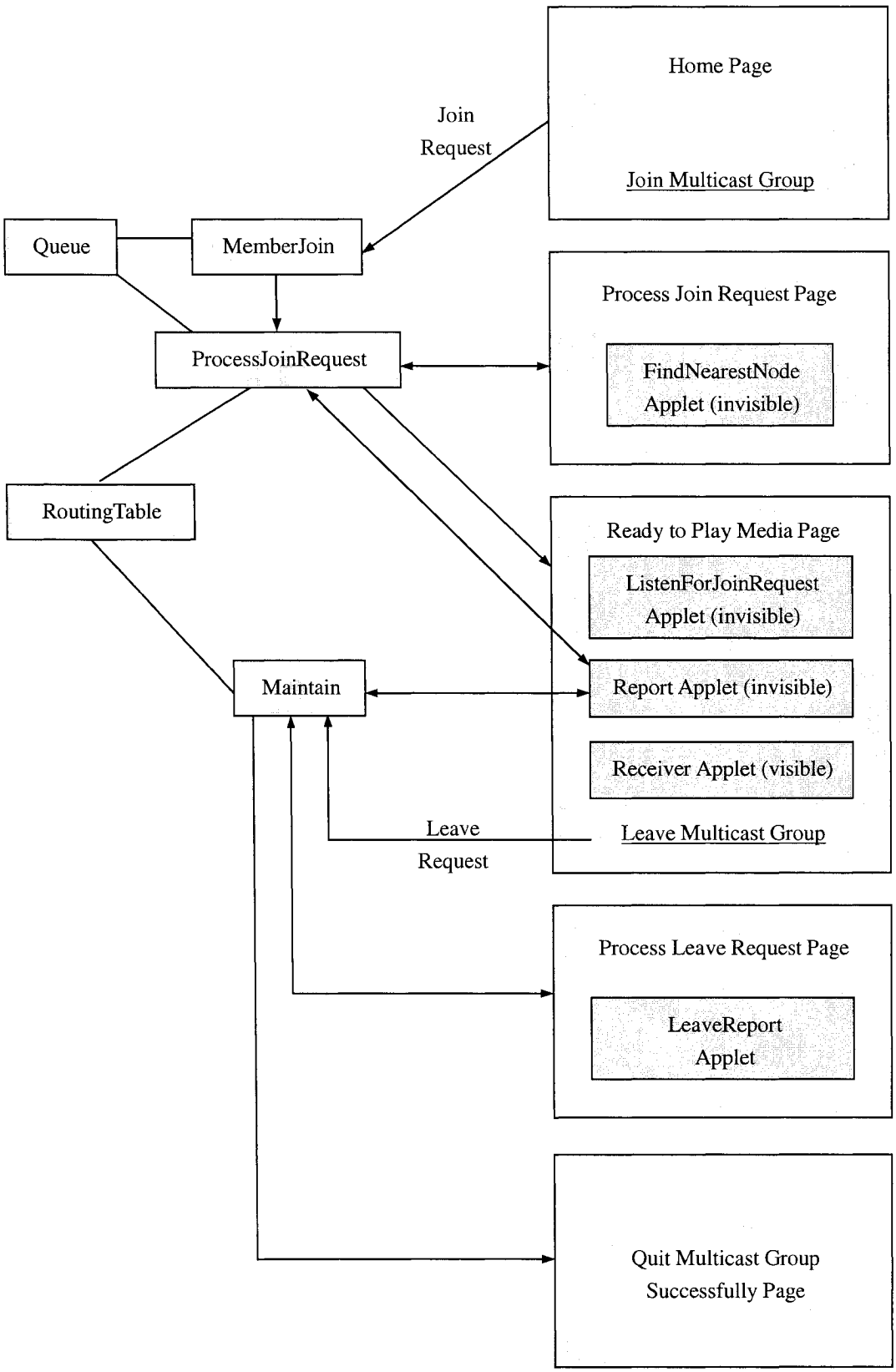


Figure 5.1 System flow chart

5.2 Proxy Package

MemberJoin Agent and ProcessJoinRequest Agent:

The MemberJoin Agent and ProcessJoinRequest Agent are described as follows:

1. When a client sends a "member join/rejoin request", it is directly put into a queue and time-stamped. There are two queues: normal queue and high priority queue. All new member join requests are put into the normal queue, while member rejoin requests in partition recovery are put into the high priority queue and are always processed first. This is based on the fact that waiting for some time at the initializing stage to join the multicast group is acceptable, but waiting for even a short period of time while media is playing is unendurable, for it causes missing some part of the media.
2. The ProcessJoinRequest Agent who processes the member join/rejoin requests first checks if there is a request in the high priority queue. If there is at least one request in high priority queue, then if the client's "member rejoin request" is the first one in the queue, it is processed immediately, otherwise it sends "waiting response" to the client. If there is no request in high priority queue, then if the client's request is the first one in the normal queue, it is processed immediately, otherwise it sends "waiting response" to the client.
3. The response of new member join request is different from response of member rejoin request in that the responses of member rejoin request are messages while the responses of new member join request are pages.

MemberJoin Agent processes as follows:

1. put request in the proper queue
2. record request timestamp
3. call ProcessJoinRequest Servlet

ProcessJoinRequest Agent processes as follows:

```
if it is run for the first time, load configure file;
if there is request in high priority queue {
   if this node's request is the first request in the high priority queue
      process rejoin request;
   else if it is a new member
      send "process join request" page;
   else
```

```
      send "WaitingResponse";
}
else if there is request in low priority queue {
   if this node's request is the first request in the normal queue
      process new member join request;
   else
      send "process join request" page;
}
```

Note that the configure file stores deployment parameters such as out degree, and report time interval, etc.

Process new member join request:

```
if it is run for the first time: create routingTable, add server as root of
   the routing table;
if the request is sent from server: send source page;
else if the client has already joined: send "already joined" page;
else if root is the only node in the routingTable {
   newNode.setIP(newNodeIP);
   newNode.setParent(rootIP);
   newNode.setLastUpdate();
   newNode.setStatus("alive");
   root.addChild(newNodeIP);
   rt.addNode(newNode);
   announce member join;
   send "ready to play media" page;
}
else {
   get nearestNodeIP from the request;
   get potentialParentIP from the request's session;
   if ((nearestNodeIP == null) or (potentialParentIP == null)) {
      set root as potentialParent of the node;
      send "process join request" page;
   }
   else {
      if ((nearestNodeIP == potentialParentIP) or (the nearest node is a leaf
      node)) {
         newNode.setIP(newNodeIP);
         newNode.setParent(nearestNodeIP);
         nearestNode.addChild(newNodeIP);
         newNode.setLastUpdate();
         newNode.setStatus("alive");
```

```
            rt.addNode(newNode);
            announce member join;
            send "ready to play media" page;
        }
         else {
            set nearest node as potentialParent of the node;
            send "process join request" page;
        }
    }
}
```

Process rejoin request:

```
if the root has no children {
    root.addChild(thisNodeIP);
    node.setParent(root);
    announce member rejoin;
    send "RejoinedResponse";
}
else {
    get nearestNodeIP from the request;
    get potentialParentIP from the request's session;
    if ((nearestNodeIP == null) or (potentialParentIP == null)) {
        set root as potentialParent of the node;
        send IPList of the potential parent and its children;
    }
    else {
        if ((nearestNodeIP == potentialParentIP) or (the nearest node is a leaf
        node)) {
            nearestNode.addChild(thisNodeIP);
            thisNode.setParent(nearestNodeIP);
            announce member rejoin;
            send "RejoinedResponse";
        }
        else {
            set nearest node as potentialParent of the node;
            send IPList of the potential parent and its children;
        }
    }
}
```

Maintain Agent:

The main function of the Maintain Agent is to process refresh messages from clients. There are two types of report: "NormalReport" and "LeaveRequest".

Process leave request:

1. If the leave request is from a leaf node, it can quit the multicast group immediately;
2. Otherwise if the node sends leave request for the first time, the server sets the status of the node as "dead", breaks the link between the node and its parent, and the links between the node and all its children, then sends "processing leave request" page to the node;
3. Every time the node sends leave request again, the server checks if all its children have found new parents and have rejoined the multicast group successfully. The server sends "quit multicast group successfully" page after all it children rejoined the group.

The pseudo code of processing leave request is listed here:

```
if the report is "LeaveRequest" {
   if it is a leaf node {
      parentNode.removeChild(thisNodeIP);
      routingTable.removeNode(thisNodeIP);
      announce node leave;
      send "quit multicast group successfully" page;
   }
   if the status of the node is "alive" {
      node.setStatus("dead");
      parentNode.removeChild(thisNodeIP);
      node.setParent(null); //this node now has no parent
      for each child on its children list: childNode.setParent(null);
      send "processing leave request" page;
   }
   if the status of the node is "dead" {
      leavePermitted = true;
      for each child on its children list: if it has not found a new parent
      yet, then leavePermitted = false;
      if (leavePermitted) {
         routingTable.removeNode(thisNodeIP);
         announce node leave;
         send "quit multicast group successfully" page;
      }
      else: send "process leave request" page;
   }
}
```

Process normal report:

1. Record the last refresh time of the node.
2. Check parent's status: if the server misses report from the node's parent for 3 times, the server considers it leaves without notifying group. The Server breaks the link between the parent and the grandparent, sets all children of the parent node as no parent, and then deletes the parent node from the routing table.
3. Check children's status: if a child of the node fails to report to the server for 3 times, it is considered left without notifying group. But the server only deletes those leaf nodes and announces their departure.
4. Send response: if the parent node is still "alive", it sends "NormalResponse"; if it is "dead" or the node has no parent currently, then it sends "ParentDiedResponse".
5. Process server's report: if the server reports, the Maintain Agent checks all nodes on the routing table and deletes those nodes that do not report for 5 time. The Maintain Agent also checks the queues and deletes those requests that do not be processed for a period of time.

The pseudo code of processing normal report is listed here:

```
if the report is "NormalReport" {
   node.setLastUpdateTime();
   // Check parent's status
   if the node has parent and the parent node is still "alive" {
      if the server has missed report from its parent node 3 times {
         for each child of the parent node: childNode.setParent(null);
         if the node has grandParent:
            grandParentNode.removeChild(parentNodeIP);
         routingTable.removeNode(parentNodeIP);
         announce parent node leave;
      }
   }

   // Check children's status
   for each child on its children list {
      if the server has missed report from the child node 3 times {
         childNode.setStatus("dead");
         if the child is a leaf node {
            node.removeChild(childNodeIP);
            routingTable.removeNode(childNodeIP);
            announce child node leave without informing group;
```

```
        }
      }
   }

   // Send response
   if the node has parent {
      if the status of the parent node is "alive": send "NormalResponse";
      if the status of the parent node is "dead": send "ParentDiedResponse";
   }
   else { //if the node has no parent
      if the node is not the server
         send "ParentDiedResponse";
      else { //if the node is the server, do routine checks
         for each node on the routingTable {
            if the server has missed report from the node 3 times:
               node.setStatus("dead");
            if the status of the node is "dead" {
               if the node has parent {
                  parentNode.removeChild(thisNodeIP);
                  node.setParent(null);
               }
               if the node has no parent and has failed to report 5 times{
                  routingTable.removeNode(thisNodeIP);
                  announce node leave;
               }
            }
         }
         if there is request in high priority queue
            if the first request has not been processed for a period of time
               remove it from the queue;
         if there is request in low priority queue
            if the first request has not been processed for a period of time
               remove it from the queue;
      }
   }
}
```

5.3 Client Package

1. Member Join:

   The client package of member join use case includes two classes (two applets): one applet perform the task of searching for the nearest node during the processing of new member join, the FindNearestNode applet; the other listening at a port on the client's

machine for member join/rejoin requests from other members after itself joined the multicast group successfully, the ListenForJoinRequest applet.

The FindNearestNode applet is embedded in "process join request" page. It only works in new member join case. After it receives a member list from the proxy, it sends "MeasureRttRequest" to all the members on the list by using socket. Hosts who receive "MeasureRttRequest" simply return "MeasureRttResponse". The FindNearestNode applet records the rount-trip time to each hosts and return the nearest node IP address to the proxy.

After each new member joins the multicast group successfully, it shows "ready to play media" page. The ListenForJoinRequest applet is embedded in this page. It setups a socket at the client's machine listening for "MeasureRttRequest" from new members who come later, or existing members whose parents leave therefore need to find new parents.

2. Report:

   The Report applet is embedded in "ready to play media" page. As soon as the node joins, it starts to send refresh message to the proxy periodically. The Report applet processes as follows:

```
while(true) {
    sleep for a report time interval;
    send "NormalReport";
    receive response;
    if the response is "NormalResponse": save routing info;
    else if it is "ParentDiedResponse" {
        call MemberJoin Agent: MemberJoin?priority=high;
        receive response;
        while(response != "RejoinedResponse"){
            if (response == "WaitingResponse") {
                sleep for a period;
                call ProcessJoinRequest Agent: MemberJoin?priority=high;
                receive response;
            } else if response is IP list {
                find nearest node on the IP list;
                call ProcessJoinRequest Agent, bring with nearest node IP;
                receive response;
```

```
            }
        } // end of while
    }
} // end of while
```

3. Member Leave:

   If the node which sends leave request is a leaf node, it gets "quit multicast group successfully" page. Otherwise it receives "process leave request" page. The LeaveReport applet is embedded in this page. It sends leave request periodically until it gets "quit multicast group successfully" page.

4. Streaming:

   It is embedded in "ready to play media" page. It runs Receiver applet listening for incoming media (audio, video) streams, and then plays media after it receives streams. It is implemented by using JMF (Java Media Framework).

5.4 Proxy Side Client Package

The proxy side client package is a little bit different from client side client package. It does not perform the task of searching for the nearest node since it is the first node that joins the multicast tree. But after it joins, it runs ListenForJoinRequest applet, Report applet, and Receiver applet. The proxy side client package does not have member leave use case.

5.5 Race Condition Solution

We have already discussed race condition solution from chapter 3 to chapter 5. In this section we summarize them.

Race condition includes:

1. Two members send join requests simultaneously.
2. A member has sent leave request, all its children send join requests almost at the same time.
3. A new member sends join request at the point when the tree is partitioned and has not been recovered. Can the new member join the partitioned sub-tree, or it can only join the sub-tree rooted at root?

4. A node died, one of its children nodes sends join request, almost at the same time the user of this node sends leave request. The join request and the leave request reach the server at the same time, which one should be processed?
5. Several nodes died simultaneously.

Race condition cases are illustrated in Figure 5.2.

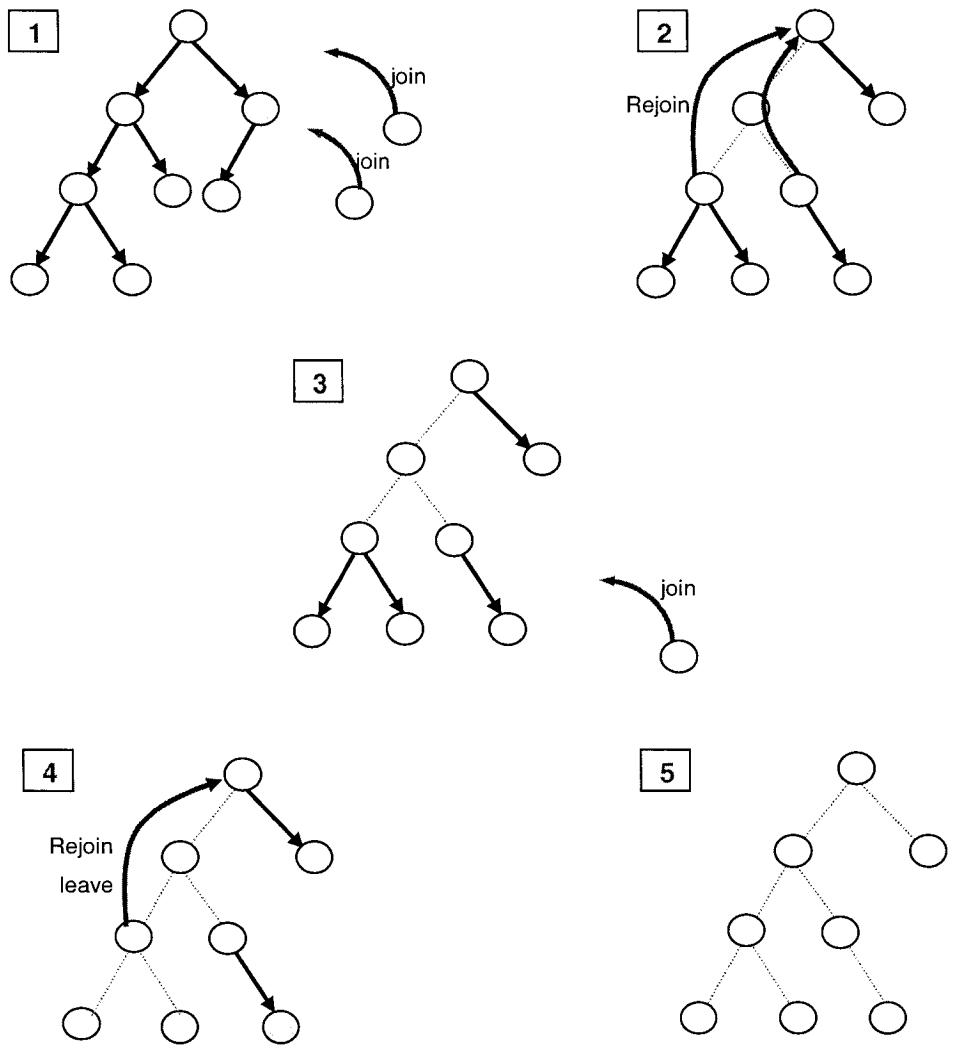


Figure 5.2 Race condition cases

Cure for race condition case 1:

All new member join requests are put into a queue. Another servlet ProcessJoinRequest Agent retrieves the requests from the queue and process them in a serialized order. This is to guarantee that new members join one by one.

Cure for race condition case 2 and 3:

Member rejoin requests in partition recovery are put into a high priority queue and are always processed with higher priority. This is to guarantee that a new member always joins a repaired tree.

Cure for race condition case 4:

The requests are time stamped when it is put into the queues. When proxy server reports, the Maintain Agent checks all the requests in each queue and deletes those which have not been processed for a certain period of time. If a member leaves after it sends join/rejoin request, the request will be deleted. This is to avoid deadlock.

Cure for race condition case 5:

When node reports, the Maintain Agent checks the status of its parent and children. A non-leaf node failed to report 3 times would be perceived by its children, while a leaf node failed to report 3 times would be discovered by its parent. This way the system is able to respond to members' dynamic departure in an acceptable period of time so that the streaming data would not be interrupted for so long.

In panel 5 of Figure 5.2, many nodes leave simultaneously, but only the departure of children of root would be recognized by the root, other members' departure would not be noticed since their neighbors have also left. When number of the dead nodes increases, they hold considerable system resources. To reclaim system resource, when proxy server reports, the Maintain Agent checks all the nodes in the routing table and deletes those which do not report for a quite long period of time. This guarantees that the routing table always keeps a list of live nodes.

Chapter 6

Testing and Measurement

Now we have proposed the proxy-based single source ALM protocol – ProBaSS. To test our system, we use the multicast tree constructed by the protocol to transmit data: first text-based simple message, and then media (video, audio). First we build a text-based online chatting system and use it to measure the control overhead introduced by the protocol, and then we implement a single-source Video/Audio conferencing system to test if the system supports dynamic member join/leave in our overlay network in a real-time media streaming application.

6.1 Text-Based Online Chatting System

6.1.1 System Design

To transmit text, we made some modifications to our system:

1. During the refresh message exchange, the server should return to each client the routing info that comprises not only its children list but also its parent IP.
2. In ListenForJoinRequest applet, we add a GUI that allows user to type text, then click the "send" button to send message to other users of the multicast group. The user interface is shown in Figure 6.1.

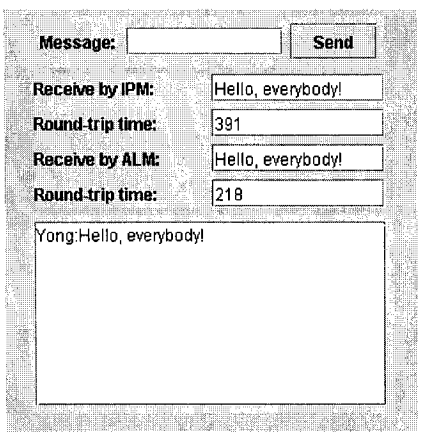


Figure 6.1 User interface in simple text transmission

3. The message is first sent and returned using unicasting. That is, the message is sent directly to the server and returned.
4. The message is then sent and returned using ALM. That is, the message is sent to the proxy server along the branches of the multicast tree, and is returned the same way. At the same time, the message is multicasted to the group by the proxy. This is illustrated in Figure 6.2.

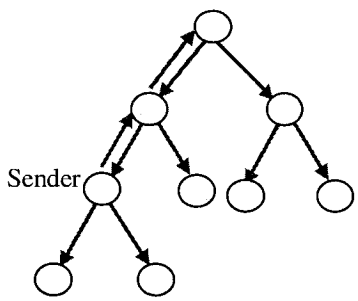


Figure 6.2 Message flow in simple text transmission

This is, actually, a many-to-many text-based online chatting system with multiple sources and receivers.

6.1.2 Measurement

Compared with IP Multicasting, ALM protocols introduce more bandwidth and delay for the sake of supporting more users and scalability. In this section, we measure the control overheads introduced by ProBaSS.

Delay Measurement

To measure the control overhead introduced by the protocol, we use the online chatting system but set degree to 1. First we ask clients to join to construct an overlay multicast tree. Then we ask client sends simple messages using the applet shown in Figure 6.1. The message is first sent directly to the server and returned. Then it is sent along the branches of the overlay tree to the server and is returned along the same path. We measure the round trip time of message transmission by using the above two approaches respectively at each host. The round trip time (RTT) in the former approach represents the time that would have been resulted by using IP Multicast in the wide-area network (RTT_IPM), while the time in the latter approach represents the time due to using the ALM protocol in the overlay network (RTT_ALM). Figure 6.3 denotes the round trip time calculated in each approach.

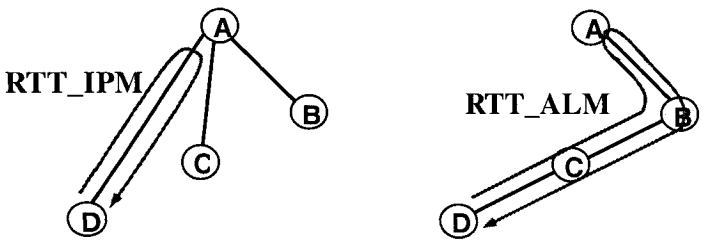


Figure 6.3 Round trip time of IP Multicast and ALM

After the measurements, the *extra* end-to-end delay introduced by the ALM algorithm is:

$$\text{DELAY} = \frac{\text{RTT\_ALM} - \text{RTT\_IPM}}{2}$$

The configuration of deployment parameters in this test is shown in Table 6.1.

out degree	1
report time interval	2 Sec
socket time out	2 Sec

Table 6.1 Online chatting system deployment configuration

The test was conducted at the Multimedia Communications Research Lab (MCRLab) at the University of Ottawa. First, we tested the RTT on the LAN between one client and the server, that where physically next to each other. In this test, the RTT_IPM roughly equals to the RTT_ALM, and the network delay is almost zero. We therefore use this test to measure the processing time of ProBaSS. The test scenario and data are shown in Figure 6.4 and Table 6.2. The result is shown in milliseconds. The result shows the maximum, minimum and average client-to-server RTT.

Figure 6.4 Test on ALM processing time

	MAX	MIN	Average
RTT_IPM	375	219	300.34
RTT_ALM	391	281	299.71
Average			300.03

Table 6.2 ALM processing time

Then the test was conducted on the intranet at MCRLab. Six clients joined the multicast group. It is illustrated in Figure 6.5.

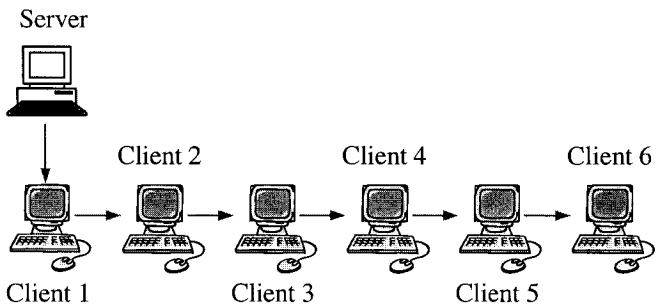


Figure 6.5 Test on the intranet

The test data is shown in Table 6.3. The result is shown in milliseconds. The result reflects both the average and the variation of the RTTs.

Clients		Client 1	Client 2	Client 3	Client 4	Client 5	Client 6
RTT_IPM	MAX	375	390	391	391	375	391
	MIN	219	219	219	203	234	234
	AVG	300.34	298.68	304.00	297.88	311.92	304.96
RTT_ALM	MAX	391	704	1109	1532	1719	2110
	MIN	281	500	594	703	1313	1704
	AVG	300.03	644.66	941.00	1220.30	1563.48	1905.56
Delay	MAX	N/A	234	391	617	726	921
	MIN	N/A	94	141	211	516	680
	AVG	N/A	172.72	318.28	460.98	625.52	800.04

Table 6.3 Test data from the intranet

Measurements were then taken on the Internet. This time client 2 and client 3 were high-speed Internet users at home, with cable access to the Internet. Client 3 is 30 km away from the server, while client 2 is 2 km away. This is illustrated in Figure 6.6. The test data is shown in Table 6.4. Again, the result is in milliseconds, and the result reflects the average

RTTs and their variations.

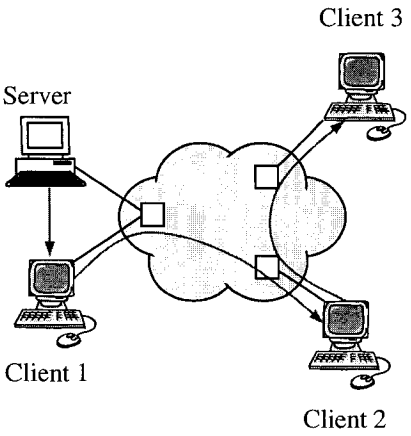


Figure 6.6 Test on the Internet

Clients		Client 1	Client 2	Client 3
RTT_IPM	MAX	375	718	531
	MIN	219	312	260
	AVG	300.34	418.54	432.81
RTT_ALM	MAX	391	1453	4006
	MIN	281	640	451
	AVG	299.71	829.75	1396.87
Delay	MAX	N/A	492	1782
	MIN	N/A	16	15
	AVG	N/A	205.36	481.69

Table 6.4 Test data from the Internet

Looking at the measurements and the calculated results, we can draw the following conclusions:

1. The ALM algorithm provides practical scalability at the expense of delay.
2. It is very interesting to note that most of the delay is caused primarily by processing time at the intermediate clients, rather than network propagation time in terms of simple message transmission.
3. The control overhead introduced by the ALM algorithm is 150 msec more per extra node on the intranet, or 250 to 300 msec more per extra node on the Internet on the average. This is quite acceptable in the context of a single-source event multicasting application, since messages are sent only one way and receiving messages with a certain consistent delay does not detriment users' experience significantly (think of

watching a sports game, or any TV program for that matter, with a 10-second delay: the viewer does not notice the delay due to lack of a reference point).

4. The control overhead is introduced by the refresh message exchanges. It is influenced by the network load as well as the deployment configuration.
5. The algorithm also has an impact on reliability. We noticed that when we were testing on the intranet, data were never lost. But when we moved our test on the Internet, things changed. Especially when the network traffic was busy, data sent by the ALM approach were sometimes lost, but data sent by the IP Multicast approach were seldom lost. Although some media types such as video are fault-tolerant, we have to say that the reliability or effectiveness of ALM depends on the network condition of that particular time.

Bandwidth Measurement

After the system becomes stable, the bandwidth overhead introduced by the protocol is caused by refresh message exchange. That is: a normal report with a normal response per report interval. The report and response strings are listed in Table 6.5. The bandwidth overhead caused by the protocol is:

$$\frac{(12+14)*8}{2} = 104 bit/s$$

"NormalReport"	12 character
"NormalResponse"	14 character
Report interval	2 Sec
Bandwidth overhead	104 bit/s

Table 6.5 Bandwidth overhead

The measurements show that the proposed ALM protocol ProBaSS introduces acceptable control overheads.

6.2 Single Source Video/Audio Conferencing System

The difference between text-based message and time-based media content (video/audio) lies in: 1) the latter contains huge volume of data, therefore requires considerable bandwidth for delivery, and 2) the latter must be delivered within strict time frame in order to present acceptable results. Therefore time-based media content delivery is often called "streaming", as it is delivered in a steady stream [28].

Transmitting media data across the net in real-time requires high network throughput. Even one second raw video may contain hundreds megabits data (e.g. 177Mb/s for 640*480, 24 bit color, 24 frames/sec video). It must be compressed before it is transmitted across the network, and decompressed before it is presented. The algorithms of compression/ decompression are called codecs. The compressed video content is stored in the storage of the streaming server. On the receiver end, the received streams are first buffered, and then are decompressed for playback. The buffering mechanism is to remove jitter and present acceptable result. If live video is distributed, a buffer is also needed on the server for data processing before transmission.

H.323 is an ITU (International Telecommunication Union) standard for Video conferencing on IP based network. It supports several video codecs (e.g. H.261, H.263) and audio codecs (e.g. G.711, G.723) for conferencing.

It's easier to compensate for lost data than to compensate for large delays in receiving the data. This is very different from accessing static data such as a file, where the most important thing is that all of the data arrive at its destination. Consequently, the protocols used for static data don't work well for streaming media.

The HTTP protocol is based on the Transmission Control Protocol (TCP). TCP is designed for reliable data communications on low-bandwidth, high-error-rate networks. When a packet is lost or corrupted, it's retransmitted. The overhead of guaranteeing reliable data transfer slows the overall transmission rate.

For this reason, underlying protocols other than TCP are typically used for streaming media. One that's commonly used is the User Datagram Protocol (UDP). UDP is an unreliable protocol; it does not guarantee that each packet will reach its destination. There's also no guarantee that the packets will arrive in the order that they were sent. The receiver has to be able to compensate for lost data, duplicate packets, and packets that arrive out of order.

Like TCP, UDP is a general transport-layer protocol, a lower-level networking protocol on top of which more application-specific protocols are built. The Internet standard for

transporting real-time data such as audio and video is the Real-Time Transport Protocol (RTP). While RTP does not provide any mechanism to ensure timely delivery or provide other QoS guarantees, it is augmented by a control protocol (RTCP) that enables user to monitor the quality of the data distribution. RTCP also provides control and identification mechanisms for RTP transmissions [28].

An RTP "session" is an association among a set of applications communicating with RTP. A session is identified by a network address and a pair of ports. One port is used for the media data and the other is used for control (RTCP) data.

A "participant" is a single machine, host, or user participating in the session. Participation in a session can consist of passive reception of data (receiver), active transmission of data (sender), or both.

Each media type is transmitted in a different session. For example, if both audio and video are used in a conference, one session is used to transmit the audio data and a separate session is used to transmit the video data.

These are the background knowledge of the system. Our purpose of building a single source Video/Audio conferencing system is to prove that ProBaSS provide a way for time-based media content to be delivered to a large number of users simultaneity via the Internet.

6.2.1 System Design

To transmit video/audio, some modifications are made to our system:

1. We use a web camera to capture video/audio. This real-time media serves as data source. To make it simple, the capture device is connected to the proxy server.
2. First, the system administrator joins the multicast group on proxy server. This enables the proxy server to play conferencing video/audio captured by the capture device.
3. Then by refresh message exchange, the proxy receives routing info (its children list). When a new client joins, the proxy opens a RTP session, adds a new destination to the session, and starts the session to send a copy of the captured video/audio to that client while it is playing locally.

4. Same thing happens at each client. At the time a client joins the multicast group, it is a leaf node. It only plays the received media streams. Then each time a new member joins this client, this client reinitializes the session and adds a new destination to send a copy of the streaming data to that client while still plays locally.
5. The departure of a leaf node does not affect its neighbors. When a non-leaf node leaves with or without informing group, after all its children rejoin the multicast tree, the parent of the departing node reinitializes its session to send media to all its new children. The children of the departing node also reinitialize their sessions to receive new media streams from new parent.

6.2.2 Implementation

The system is implemented by using Java Media Framework (JMF). JMF is an application programming interface (API) for incorporating time-based media into Java applications and applets. It provides control over media capturing, processing and rendering. It also supports RTP for transmission and reception of real-time media streams across the network.

Before the data is ready for transmission over the network it needs to be compressed, and after the transmission it needs to be decompressed before it can be presented. This is called codec (compression/decompression). The JMF RTP API supports an H.323 compliant set of audio and video codecs. The audio codecs are: G.711, DVI, G.723, and GSM, while the video codecs are: JPEG, H.261, and H.263. The DVI audio codec and JPEG video codec are suitable for Video/Audio conferencing over the Internet.

Here is how the Video/Audio conferencing system is implemented by using JMF:

1. The proxy server runs the Source applet at its client package to capture and play the conferencing video/audio. The client runs the Receiver applet to play the received media streams. Non-leaf node also forwards the received streaming data to the next hop on the tree. In short, Source applet performs the functionality of capturing, playing and forwarding media, while Receiver applet performs the functionality of receiving, playing and forwarding. Figure 6.7 shows the streaming connection between the proxy server and one of its children clients.

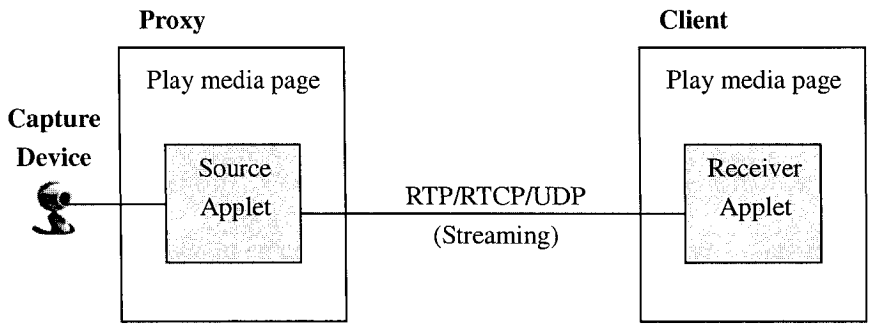


Figure 6.7 Streaming between proxy and client

Note that in this section we should call the "ready to play media page" as "play media page".

2. When the administrator (say the source) joins, the Source applet plays the captured video/audio.
3. When Client A joins, the Source applet performs three tasks:
   a. Create a processor. A processor is to compress media data into RTP supported format so that it can be transmitted across the network. The input of the processor is raw media data. The output of the processor includes two streams: video and audio. In this application, the video is set to JPEG/RTP (320x240) format, and the audio is set to dvi/rtp (8000.0 Hz, 4-bit, Mono) format. After the processing, the data is ready for transmission.
   b. Create a RTP session manager. A session manager is the local representative of a session. Then it sets the data source as push data source so that the client only needs to listening for incoming data streams. The session manager initializes the local and destination session addresses for transmission. The transmission uses four ports on each machine for video/rtp, video/rtcp, audio/rtp, and audio/rtcp.
   c. Initialize the local session. It creates a session manager listening for loopback streaming media for local playing.
4. Client A then receives the "play media page" as soon as it joins the multicast group successfully. It contains the Receiver applet that initializes the session to listen for incoming data. When a new stream comes, the session listener receives an event about participant, and the receive stream listener receives an event of the data streams. The data source can be retrieved from the streams for local media playing.
5. When client B comes, client A stops old session and starts new session to send streaming data to itself and Client B.

6. When client A leaves, the source finds its children changed from its received routing info. Then the source reinitializes the session to send media to new children. After Client B receives "ParentDiedResponse" and then "RejoinedResponse" it also reinitializes the session to receive new media streams. Then after being interrupted for a short period of time, client B will receive new media streams from the source.

The configuration of deployment parameters is shown in Table 6.6.

out degree	1
streaming buffer size	200
JPEG quality	0.5

Table 6.6 Video/Audio conferencing system deployment configuration

Note:

1. The received streaming data is first put in buffer. It will not be played until the buffer is full. Larger buffer size means more delay, but smaller buffer size means lower transmission efficiency because the package header forms a considerable proportion of data during transmission.
2. The JPEG quality control is a control for specifying the parameters for quality. It is referenced by a float value of 0.0 for minimal quality and 1.0 for maximum quality. If is used for the quality of encoding and decoding. A higher quality setting will result in better image quality for video, but requires more CPU time to process and higher bandwidth to transmit. There is usually a tradeoff between CPU and bandwidth usage and the quality.

The test was conducted on the intranet at MCRLab. Four clients joined the multicast group to form the overlay. The test leads to the following conclusions:

1. A new member can receive audio/video streams as soon as he joins the multicast group.
2. A member's departure does not affect other group members to receive streams continuously after a short period of interruption.
3. Each end-system receives audio/video streams with a consistent delay. It does not affect users from enjoying the audio and video.
4. The delay is introduced by the protocol as well as the buffering mechanism. We can adjust buffer size to adapt to the users' requirements.

5. There is no evidence of package drops in the test on the intranet since the received audio and video streams can be played continuously.

The screenshot of the "play media page" at each receiver is shown in Figure 6.8. Embedded in the page is the Receiver applet playing the received video stream.

Figure 6.8 Receiver's screen

The single source Video/Audio conferencing system shows that ProBaSS provides full multicast function support in group media distribution applications.

Chapter 7

Conclusions and Future Work

Multicasting is the most efficient way to deliver large amount of data in one-to-many communications. For it sends just one copy of data, replicates only as necessary. The traditional way to implement multicasting is that network routers perform membership management, maintain data delivery path and replicate and forward data. This is IP Multicasting.

However, it has been shown that it is extremely difficult to deploy IP Multicast at a large scale due to scalability issues, deployment hurdles and marketing reasons. Therefore ALM has been proposed as an alternative to IP Multicasting. This approach uses end-hosts participating in the application to perform multicast functions. That is to construct and maintain a multicast overlay for data delivery.

This thesis has investigated the possibility of transmitting bandwidth-intensive content like video to a large number of users via the Internet, and has proposed a proxy-based single source ALM protocol "ProBaSS", its architecture, design and implementation. By using ProBaSS, it is possible for content providers to solve the bottleneck bandwidth problem and deliver bandwidth-intensive audio and video to a group of users in real-time.

7.1 Conclusions

Based on the experience with the implemented ALM system, the following conclusions can be drawn with respecting to the ALM evaluation criteria discussed in section 2.2.3:

Ease of Deployment

It is practical for content providers to deploy dedicated proxy servers to provide multicast services to end-users universally. A proxy-based ALM solution is much easier to be deployed compared with IP Multicasting, since it does not need network infrastructure support.

Robustness

The proposed locally centralized but globally distributed overlay multicast architecture of ProBaSS has better robustness compared with both the traditional proxy-based architecture and the end-system based architecture, which have been discussed in section 2.2.2. The proposed protocol ProBaSS is built based on this architecture. It targets high robustness in providing multicast services.

Performance

The proposed protocol "ProBaSS" introduces acceptable control overheads (bandwidth and delay). It is possible for content providers to use "ProBaSS" to deliver audio/video to a large number of users in real time via the Internet. The reliability or effectiveness of the proposed protocol "ProBaSS" depends on the network condition of the particular time when data is delivered.

Scalability

Scalability is addressed to some extend in ProBaSS. The overlay multicast architecture of ProBaSS has better scalability compared with the traditional proxy-based architecture and the end-system based architecture, for the reasons which have been discussed in section 3.2. Besides, a system built based on ProBaSS would serve more users if more proxy servers could be deployed.

Dynamic Discovery of Sources and Receivers

The proposed ALM protocol "ProBaSS" is efficient in performing full multicast functions, including membership management, data delivery path construction, and data replication and forwarding.

7.2 Future Work

The proposed protocol "ProBaSS" provides the possibility of transmitting bandwidth-intensive content to a large number of users via the Internet. However, there is still a long way to go in the commercialization process. Some work remains to be done in future research.

First, NAT (Network Address Translator) and firewalls impose restrictions on the connectivity of hosts participating in the overlay. Some NAT/firewalls may prevent the

system from functioning (e.g. some firewalls prevent the system from getting IP address of end-hosts). Therefore intensive research needs to be done to solve these problems.

Second, ProBaSS supports data transmission from a single source. It can not compete with multi-source protocols which are much more complex. However, single source ALM protocol still can be used for multi-source multicasting in some cases. For example, in an interactive lecture there is only one non-root sender active at a time. It is possible for the sender to unicast to the root, and the root then performs multicasting behalf of the sender. Research on this area can be done to retain the simplicity of the protocol on one hand and improve it to support multi-source collaboration on the other hand.

Moreover, the format that is used in video conferencing is JPEG 320*240. This resolution is acceptable in a video conferencing application, but is not acceptable for a movie playing, not to say HDTV. The latter demands much more bandwidth for delivery and may cause packages to be dropped more often. So some streaming technologies need to be combined with ALM to deliver better quality service.

To improve QoS, congestion control needs to be done. The network load may change and a link that is efficient at a time may slow down later. So each member should keep a record of end-to-end delay to its parent. Then based on the statistics of these records, inefficient links can be dropped by informing some members to find new parents.

Besides, multi-thread simulation test and wide area field test need to be done on the implemented system. Race condition support need to be tested to make the system more scalable and robust.

Last but not least, is the copyright issue. If the content relates to some copyright issue and the user saves the content somewhere in his hard space while enjoying the content, how can copyright be protected? Therefore something needs to be done to protect copyright.

References

[1] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols", available at: http://minoas.di.uoa.gr/P2PBackground/a-comparative-study-ofALM.pdf, last visited on April 5, 2005.

[2] A. El-Sayed and V. Roca, "A Survey of Proposals for an Alternative Group Communication Service", *IEEE Network*, 17(1), 46-51, Feb. 2003.

[3] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users", In *Proc IEEE INFOCOM*, June 2002.

[4] J. Jannotti, D. Gifford, K. L. Johnson, and M. F. Kaashoek, "Overcast: Reliable multicasting with an overlay network", in *Proc. 4th Symp. Operating System Design Implementation (OSDI)*, Oct. 2000.

[5] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, "Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications", *IEEE INFOCOM'03*, volume 2, pp. 1521-1531.

[6] Y. Chu, S.G. Rao, S. Seshan, and H.S. Zhang, "A Case for End System Multicast", *IEEE J. Select. Areas in Comm.*, special issue on networking support for multicast, 2002, Volume 20, Issue 8, pp. 1456-1471.

[7] J. Liebeherr and M. Nahas, "Application-layer multicast with delaunay triangulations", *Proc. IEEE Globecom*, Nov. 2001.

[8] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application level multicast using content-addressable networks", in *Proc. NGC,* 2001.

[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast", in *Proc. ACM SIGCOMM*, Aug. 2002.

[10] D. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming", in *Proc. IEEE INFOCOM*, Mar. 2003.

[11] P. Francis, Yoid: Your own internet distribution, Apr. 2000. Available at: http://www.icir.org/yoid/, last visited on April 5, 2005.

[12] A. Ganjam, H. Zhang, "Internet Multicast video delivery", in *Proc IEEE*, Volume 93, Issue 1, pp: 159 – 170, Jan. 2005.

[13] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Live Media Streaming", *IEEE INFOCOM,* Miami, FL, USA, Mar. 2005.

[14] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in cooperative environments", in *Proc. SOSP*, 2003.

[15] L. Guo, S. Chen, S. Ren, X. Chen, and S. Jiang, "PROP: a scalable and reliable P2P assisted proxy streaming system", in *Proc. ICDCS*, Tokyo, Japan, Mar. 2004.

[16] Y. Cui, B. Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast", in *IEEE J. Select. Areas in Comm.*, vol. 22, Jan. 2004.

[17] X. Zhang, Q. Zhang, Z. Zhang, G. Song, W. Zhu, "A construction of locality-aware overlay network: mOverlay and its performance", *IEEE J. Select. Areas in Comm.*, vol. 22, Jan. 2004.

[18] M. Hefeeda, B. Bhargava, and D. K.-Y. Yau, "A hybrid architecture for cost-effective on-demand media streaming", *Computer Networks*, 44(3), Feb. 2004.

[19] J. Liu, B. Li, and Y.-Q. Zhang, "Adaptive video multicast over the Internet", *IEEE Multimedia*, vol. 10, no. 1, pp. 22-31, Jan./Feb. 2003.

[20] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol", *RFC 1075*, Nov. 1998.

[21] M. Castro, P. Druschel, A.M. Kermarrec, A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure", *IEEE J. Select. Areas in Comm.*, 20(8), 2002, pp. 1489-1499.

[22] D. Kostic, A. Rodriguez, J. Albrecht, A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", *ACM 2003 Symposium on Operating System Principles (SOSP'03)*, pp. 282-297.

[23] Java Applet tutorial, available at http://java.sun.com/docs/books/tutorial/applet/, last visited on April 5, 2005.

[24] Security in Java 2 SDK 1.2, available at: http://java.sun.com/docs/books/tutorial/security1.2/TOC.html#tour1, last visited on April 5, 2005.

[25] Chád Darby, Applet and Servlet Communication, Java Developer's Journal, September 1998, http://www.j-nine.com/pubs/applet2servlet/Applet2Servlet.html, last visited on April 5, 2005.

[26] John Wiley & Sons, Inc., "Securing JAVA", ISBN: 047131952X, available at: http://www.securingjava.com, last visited on April 5, 2005.

[27] How to Sign Applets Using RSA Certificates, available at: http://java.sun.com/products/plugin/1.3/docs/rsa_signing.html, last visited on April 5, 2005.

[28] Java Media Framework API Guide, available at: http://java.sun.com/products/java-media/jmf/2.1.1/guide/JMFTOC.html, last visited on April 5, 2005.

[29] Y. Chu, S.G. Rao, S. Seshan, H. Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture", in *Proc SIGCOMM'01,* pp. 55-67.

[30] S. Ratnasamy, P. Francis, M. Handley, and R. Karp, "A scalable Content-Addressable Network", *ACM SIGCOMM'01*, pp. 161-172.

[31] K. Savetz, N. Randall, and Y. Lepage, "MBONE: Multicasting Tomorrow's Internet", ISBN: 1-56884-723-8, 1996, available at: http://www.savetz.com/mbone/, last visited on April 5, 2005.

[32] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner "Visualizing the Global Topology of the MBone", *Proceedings of the 1996 IEEE Symposium on Information Visualization*, pp. 85-92, San Francisco, CA, Oct.1996.

[33] Multicast Routing, available at: http://docsrv.sco.com/NET_tcpip/iproutingD.mrouted_descrip.html, last visited on April 5, 2005.

[34] Cisco Systems Inc., "Overview of IP Multicast", available at: http://www.meridianitsolutions.com/mitsivoice/ip_multicast.pdf, last visited on April 5, 2005.

[35] C. Zaharia and Z. Yang, "MBone Report", Oct. 1998. Available at: http://beethoven.site.uottawa.ca/dsrg/PublicDocuments/REPORTS-THESES/MBone%20Report.pdf, last visited on April 5, 2005.

[36] Steve Deering, "Host Extension for IP Multicasting", *RFC 1112*, August, 1989.

[37] Mojtaba Hosseini, "An End System Multicast Protocol for Multi-sender 3D Video conferencing Applications", Ph.D. thesis, School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada, October 2004.