

國立交通大學

資訊科學與工程研究所

碩士論文

適用於非對稱網路連線之動態用戶的

彈性應用層多點傳播

Resilient Application Layer Multicast Tailored for

Dynamic Peers with Asymmetric Connectivity

研究生：郭宇軒

指導教授：邵家健 副教授

中華民國 九十五年 七月

適用於非對稱網路連線之動態用戶的彈性應用層多點傳播

Resilient Application Layer Multicast Tailored for
Dynamic Peers with Asymmetric Connectivity

研究生：郭宇軒

Student：Yu-Hsuang Guo

指導教授：邵家健

Advisor：John Kar-Kin Zao

國立交通大學

資訊科學與工程研究所

碩士論文



A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年七月

適用於非對稱網路連線之動態用戶的彈性應用層多點傳播

學生：郭宇軒

指導教授：邵家健

國立交通大學資訊科學與工程研究所 碩士班

摘 要

我們的目的是設計出一個有彈性的應用層多點傳播架構，來提供即時多媒體串流服務。我們主要是針對應用層多點傳播的兩個問題。第一、組成應用層多點傳播的用戶隨時都會動態的加入或離開，因此資料傳輸並不可靠。第二、使用非對稱網路連線的用戶，它們上傳的頻寬遠遠少於下載頻寬，上傳的頻寬不足將會是應用層多點傳播的瓶頸。

我們提出三種方法的結合來產生出一個有彈性的應用層多點傳播機制，並且解決上傳頻寬不足的問題。這三種方法是：一、information dispersal algorithm，二、multiple stripes/trees，三、helper。我們增強彈性的方法是藉由保證訂閱戶就算遺失一些封包仍然可以得到完整資料，以及任一條網路連線中斷將不會有訂閱戶收不到任何封包。我們解決上傳頻寬不足的方法是充分利用所有用戶的上傳頻寬，以及藉由 helper 的加入來增加上傳頻寬的總量。

我們從模擬實驗中觀察到幾項結果：一、每個訂閱戶的訊息延遲時間是穩定的，而且訂閱戶之間的訊息延遲時間差距很少，二、就算對上傳頻寬不足的非對稱網路連線而言，每條網路連線的平均頻寬消耗是少的，三、結果顯示就算用戶有機率會發生錯誤時，訂閱戶仍然有良好的訊息成功還原率。

Resilient Application Layer Multicast Tailored for Dynamic Peers with Asymmetric Connectivity

Student: Yu-Hsuang Guo

Advisor: John Kar-kin Zao

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Our purpose is to devise a new application layer multicast scheme to provide real time multimedia streaming service. There are two challenges for application layer multicast that we focus on. First challenge is peers that form the application layer multicast service may dynamically join or leave at any time. Data transmission is not reliable. Second one is peers with asymmetric connectivity that upstream bandwidth is much less than downstream bandwidth. Insufficient upstream bandwidth will be the bottleneck of application layer multicast.

We propose the combination of three approaches to provide a resilient application layer multicast mechanism and solve the issue of peers with insufficient upstream bandwidth. The three approaches are: (1) information dispersal algorithm, (2) multiple stripes/trees, (3) helper. We improve resilience by promising subscribers can tolerate some packets loss without losing data completeness and when any link break, none of subscribers can't receive any packets. We solve the insufficient upstream bandwidth issue by fully utilizing the upstream bandwidth of all peers and increasing the total amount of upstream bandwidth by the participation of helpers.

We observe several results from the simulation: (1) the delay of message restoration for each subscriber is stable and the difference of delay between subscribers is small, (2) the average bandwidth consumption of one link is low, even for insufficient upstream bandwidth links, (3) it shows subscribers have good successful probability of message restoration even if peers have failure probability.

誌 謝

首先我要感謝邵家健老師，整個論文題目、方向、作法都是經由不斷和老師討論，以及老師給予許多建議與指引才有辦法產生，如果沒有老師的協助，我將無法獨力完成；再感謝老師在這兩年的教導，無論是研究方法或是待人處事，真的令我獲益良多。另外感謝林哲民學弟協助撰寫工具程式，令 GT-ITM 產生的網路拓樸可以匯入到 OMNeT++；感謝黃凌軒學弟幫忙分析實驗數據來畫出其中一張數據圖；感謝尤清華協助畫出兩張口試投影片所使用的樹狀結構圖；感謝張哲維學長與我討論技術問題並且提供寶貴建議；最後感謝其它 620 實驗室的成員，包含龔哲正、吳事修、陳奕興、黃為霖、朱書玄、黃國晉、李明龍、劉育志，你們是我最好的伙伴。



Contents

摘 要.....	I
ABSTRACT	II
誌 謝.....	III
CONTENTS.....	IV
LIST OF FIGURES.....	VII
LIST OF TABLES	VIII
CHAPTER 1 RESEARCH OVERVIEW	1
1.1 PROBLEM STATEMENT	1
1.1.1 <i>Dynamic Peers</i>	1
1.1.2 <i>Asymmetric Connectivity</i>	1
1.2 RESEARCH APPROACH.....	2
1.2.1 <i>Definition of Resilience</i>	3
1.2.2 <i>Three Approaches</i>	3
1.3 OUTLINE OF THESIS	4
CHAPTER 2 BACKGROUND	5
2.1 ARCHITECTURE OF MULTICAST	5
2.2 RELATED WORK.....	6
2.2.1 <i>Multiple Description Coding (MDC)</i>	6
2.2.2 <i>Multiple Stripes</i>	6
2.2.3 <i>Waypoint</i>	7
2.2.4 <i>Tree Building Algorithm</i>	7
2.2.5 <i>Enhancement of Resilience</i>	8
CHAPTER 3 PRINCIPLE.....	9
3.1 OVERALL SYSTEM.....	9
3.2 INFORMATION DISPERSAL ALGORITHM (IDA)	10
3.2.1 <i>Split</i>	10
3.2.2 <i>Restoration</i>	12
3.2.3 <i>Efficiency</i>	12
3.2.4 <i>Advantage</i>	13
3.2.5 <i>Influence of n and m</i>	13

3.3 MULTIPLE STRIPES/TREES	14
3.3.1 <i>Combine with IDA</i>	14
3.3.2 <i>Building Disjoint Paths</i>	15
3.3.3 <i>Become Interior Node</i>	16
3.4 HELPER.....	17
3.4.1 <i>Demand of Helper</i>	17
3.4.2 <i>Find Helper</i>	17
3.4.3 <i>Restriction of Helper</i>	18
3.4.4 <i>Helper Join</i>	19
3.4.5 <i>Enhancement</i>	20
3.4.6 <i>Difference between Helper and Waypoint</i>	21
CHAPTER 4 MECHANISM.....	22
4.1 SERVICE STARTUP	22
4.1.1 <i>Establish Neighbor Table</i>	22
4.1.2 <i>Source of Service</i>	23
4.1.3 <i>IDA Configuration</i>	24
4.1.3.1 <i>(n, o, m) IDA</i>	24
4.1.3.2 <i>Apply to Streaming Data</i>	25
4.1.3.3 <i>Discussion</i>	25
4.2 BEHAVIOR OF PEERS	26
4.2.1 <i>Architecture of Peer Behavior</i>	26
4.2.2 <i>Join</i>	26
4.2.2.1 <i>Randomly Join o Trees</i>	26
4.2.2.2 <i>Choose Parents</i>	27
4.2.2.3 <i>Cases of Join Accept</i>	28
4.2.3 <i>Receive and Retransmit</i>	30
4.2.3.1 <i>General Retransmission</i>	30
4.2.3.2 <i>Stripe Regeneration</i>	31
4.2.4 <i>Adjust Number of Joining Trees</i>	32
4.2.4.1 <i>Foundation</i>	32
4.2.4.2 <i>Increase and Decrease</i>	33
4.2.4.3 <i>Priority of Joining Tree Selection</i>	34
4.2.5 <i>Leave</i>	34
CHAPTER 5 PERFORMANCE ANALYSIS.....	36
5.1 PURPOSE	36
5.2 SYSTEM CONFIGURATION.....	36
5.2.1 <i>Network Topology</i>	36

5.2.2 Link and Traffic Condition.....	37
5.2.3 Trickle Setup.....	38
5.2.4 Streaming Data Setup.....	39
5.3 EXPERIMENT	39
5.3.1 Metrics	39
5.3.2 Procedure.....	40
5.4 RESULTS AND ANALYSIS	41
5.4.1 Restoration Delay and Stripe Delay.....	42
5.4.1.1 Delay for Individual Subscriber	42
5.4.1.2 Delay for Trickle	43
5.4.1.3 Relative Delay Penalty (RDP).....	46
5.4.2 Link Stress.....	47
5.4.3 Probability of Success Restoration	48
CHAPTER 6 CONCLUSION.....	50
6.1 ACCOMPLISHMENT	50
6.2 FUTURE WORK.....	51
REFERENCE	53
APPENDIX	55
A. CODE OF IMPLEMENTATION.....	55
A.1 Join and Accept.....	55
A.2 Helper Approach	56
A.3 Receive and Retransmit.....	57
B. STATISTICS OF SIMULATION.....	58
B.1 Statistics for Relative Delay Penalty	58
B.2 Statistics for Probability of Success Restoration	58
B.3 Distance between Source and Subscribers	60
B.4 Distance between Each Subscribers.....	61



List of Figures

FIGURE 1: LARGE END TO END DELAY WHEN NODE DEGREE IS FEW	2
FIGURE 2: THREE DIFFERENT ARCHITECTURE OF MULTICAST.....	5
FIGURE 3: APPLICATION LAYER MULTICAST SYSTEM.....	9
FIGURE 4: AN EXAMPLE THAT A MESSAGE IS DIVIDED INTO 2 STRIPES	14
FIGURE 5: AN ILLUSTRATION OF HELPER SELECTION	18
FIGURE 6: AN EXAMPLE OF HELPER JOIN	19
FIGURE 7: AN EXAMPLE OF HELPER APPROACH ENHANCEMENT	20
FIGURE 8: NEGOTIATION BETWEEN PEERS AND NEIGHBORS.....	23
FIGURE 9: AN ILLUSTRATION THAT SOURCE PEER IS A SOURCE PROXY	24
FIGURE 10: ARCHITECTURE OF PEER BEHAVIOR	26
FIGURE 11: NEGOTIATION BETWEEN CHILD AND PARENT.....	28
FIGURE 12: NEGOTIATION BETWEEN CHILD, PARENT, AND HELPER	29
FIGURE 13: NEGOTIATION AFTER THE FAILURE OF REQUESTING HELPER	29
FIGURE 14: FLOWCHART OF SUBSCRIBER JOIN.....	29
FIGURE 15: AN ILLUSTRATION OF STRIPE TRANSMISSION FOR SOURCE PEER.....	30
FIGURE 16: PEERS RETRANSMISSION	31
FIGURE 17: AN EXAMPLE OF STRIPE REGENERATION	31
FIGURE 18: FLOWCHART FOR ADJUSTING THE NUMBER OF JOINING TREE	33
FIGURE 19: AN ILLUSTRATION OF PRIORITY QUEUE FOR TREE ID	34
FIGURE 20: NETWORK TOPOLOGY	37
FIGURE 21: HISTOGRAM OF STRIPE AND RESTORATION DELAY FOR DIFFERENT SUBSCRIBER	42
FIGURE 22: HISTOGRAM OF MEAN DELAY FOR ALL SUBSCRIBERS	44
FIGURE 23: HISTOGRAM OF THE COEFFICIENT OF VARIATION FOR ALL SUBSCRIBERS	44
FIGURE 24: RESTORATION DELAY VERSUS PATH LENGTH FOR EACH SUBSCRIBER	45
FIGURE 25: HISTOGRAM OF RELATIVE DELAY PENALTY	46
FIGURE 26: PROBABILITY OF SUCCESS RESTORATION VERSUS DIFFERENT PEER FAILURE PROBABILITY	48
FIGURE 27: HISTOGRAM OF THE PROBABILITY OF SUCCESS RESTORATION.....	49

List of Tables

TABLE 1: REACTION AFTER RECEIVING A NEIGHBOR ACK MESSAGE.....	23
TABLE 2: REACTION AFTER RECEIVING JOIN REQUEST	30
TABLE 3: REACTION AFTER RECEIVING ONE STRIPE	31
TABLE 4: AVERAGE DELAY, VARIANCE, AND STDDEV FOR ALL SUBSCRIBERS WITH TRICKLE	43
TABLE 5: AVERAGE DELAY, VARIANCE, AND STDDEV FOR ALL SUBSCRIBERS WITH IP MULTICAST	46
TABLE 6: AVERAGE RELATIVE DELAY PENALTY FOR ALL SUBSCRIBERS	46
TABLE 7: LINK STRESS.....	47
TABLE 8: AVERAGE PROBABILITY OF SUCCESS RESTORATION	48



Chapter 1

Research Overview

1.1 Problem Statement

Our purpose is to devise a new application layer multicast scheme to provide real time multimedia streaming service. The scheme provides resilience and security of data transmission. It also removes the bottleneck of data transmission in peer upstream bandwidth.

There are two main challenges we focus on. First challenge is peers that form the application layer multicast service are dynamic. Second one is peers with asymmetric connectivity for up and down links, especially with narrow upstream bandwidth. These two issues only appear in application layer multicast but not in IP multicast.

1.1.1 Dynamic Peers

The concept of application layer multicast is when users' desktops or mobile devices (called peer) receive streaming data, they need to replicate data and send to other peers. There is no steady server and the whole service is formed by peers. Therefore, the operation of application layer multicast fully depends on peers. But peers may dynamically join or leave the service at any time. Data transmission is not reliable that service can't promise users will receive all the data. If the peers in the application layer multicast are dynamically changed frequently, the service won't work without any remedy.

1.1.2 Asymmetric Connectivity

Currently, most users use asymmetric upstream and downstream connectivity, such as ADSL, VDSL or Cable modem [19], to connect to the Internet. The upstream bandwidth is much less than downstream bandwidth. In application layer multicast, the capability of peer's upstream bandwidth will influence on how many peers it can retransmit data to them. For

example, if application layer multicast using tree structure, one peer needs to retransmit the streaming data to several peers in the same time. But upstream bandwidth might be very less than downstream bandwidth. The lack of upstream bandwidth will cause the performance of application layer multicast being poor. Otherwise, the application layer multicast can only provide low quality data streaming service.

In the present day, the data rate of most data streaming service on the Internet is around 300 to 500 Kbps [24], [25]. If we have no abundant bandwidth and want to provide streaming data to others, all we can do is use application layer multicast. When some receivers are the peers with asymmetric connectivity, it will bring an issue that they can only retransmit data to very few peers. (In Taiwan, the maximum upstream bandwidth of asymmetric connectivity is 1 Mbps and the general is 512 Kbps [20].)

This issue will limit the amount of peers that can join the service or the quality of streaming data the service can provide will be poor. Furthermore, because the degree of nodes in the tree is few, the height of the tree will be bigger and the end to end delay from source to peers will be larger. There is an example in Figure 1. Furthermore, in some special cases, if the number of users that want to join the service is few, we can't even build a workable tree between them because the choice of parent is few.

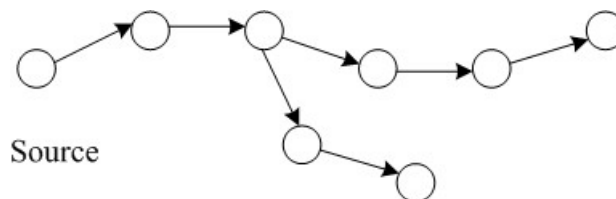


Figure 1: Large end to end delay when node degree is few

1.2 Research Approach

The name of the scheme we propose is called Trickle. The transmission paths among peers will be tree structure. In current stage, we don't focus on how to build a tree among peers. Our purpose is to improve resilience of existent tree building algorithm by integrated

with Trickle. Moreover, Trickle won't violate decentralized control if the tree building algorithm is decentralized. Following, we will define the meaning of resilience and then introduce our three approaches: information dispersal algorithm, multiple stripes, and helper.

1.2.1 Definition of Resilience

We will devise a resilient application layer multicast approach including fault tolerance, robustness, security, and solving asymmetric connectivity issue. For fault tolerance, peers can tolerate a small number of packet delay or loss and they still can receive complete message in time. For robustness, a sudden breakdown of some peers or links won't disrupt reception of downstream peers. For security, we provide a partial security protection for streaming data.

1.2.2 Three Approaches

Information dispersal algorithm (IDA) [8] IDA will be used to process the message generated from source peer and divide the message into several pieces. Peers can restore message from a part of message pieces. We use IDA for two purposes: (1) peers don't need to receive whole pieces of a message and they still can have the complete message. It will improve the resilience of data transmission, (2) peers can't know any content of the message if they don't have enough pieces. Therefore, if some peers have no authority to read the message, we only need to make sure not to let them receive too many pieces of the message. In short, the participation of helper is the main reason we want to enhance security.

Multiple stripes The pieces of messages divided by IDA approach is called stripes. In tree building step, we not only construct one tree, we construct as many different trees as the number of stripes that one message is divided into. And each tree transmits different stripe of the message. This approach will lead to two advantages. First, it can make sure that peers can receive enough stripes. When one peer leaves or crashed, descendants of the peer will only lose one stripes and they still can receive other stripes from other trees. Furthermore, when a link of tree is congested, it only makes one stripe delay. Second, every peer can fully utilize

their upstream bandwidth. A stripe is much smaller than a message, so peers with low upstream bandwidth can also support several children.

Helper Because the lack of upstream bandwidth, it is possible that new peers can not find a parent in the tree that can provide them smooth streaming data. It is because the peers in the trees can support no more children or because the peers can support more children is far from new peers. Therefore, the service will request some peers called helpers that still have unnecessary bandwidth to join and share their bandwidth to increase the total amount of upstream bandwidth of the service. By the help of helper, the service can accept more peers and each peer can receive the streaming data smoothly by having better parent choices.

1.3 Outline of Thesis

In chapter 2, we will briefly introduce three different architecture of multicast and the related work of application layer multicast. In chapter 3, we will describe the detail of the three approaches. In chapter 4, we will describe the use of three approaches in the mechanism of application layer multicast and how it works. In chapter 5, we will show the simulation results and evaluate our performance. In chapter 6, it is the conclusion and the future work.

Chapter 2

Background

2.1 Architecture of Multicast

Since multi-receiver multimedia applications, like video-conferencing, video streaming, e-learning and online-gaming, are more and more popular on the Internet, multicast is an important mechanism need to be developed. Multicast is a one-to-many transmission mechanism and is very efficient to reduce duplicate packets and bandwidth consumption when it is used for multi-receiver applications.

There are three different architecture of multicast: IP multicast, application layer multicast (ALM), and overlay multicast [21]. Figure 2 shows the difference between these three architecture clearly. IP multicast is developed on network layer and use routers as the relaying nodes. IP multicast is the most directly implementation of multicast and can reduce most duplicate packets in transmission process. However, because of several issues [1], IP multicast is not globally deployed yet.

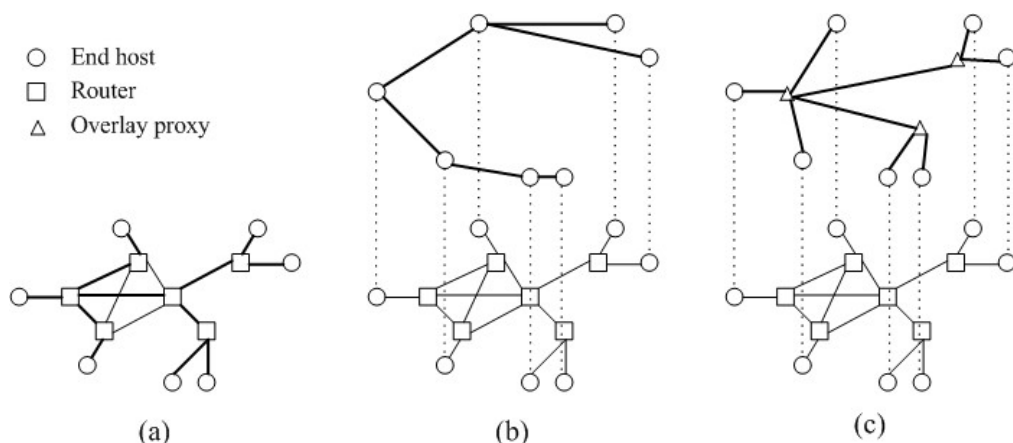


Figure 2: Three different architecture of multicast.

(a) IP multicast, (b) application layer multicast, (c) overlay multicast

Overlay multicast is the architecture that constructs a backbone by overlay proxy first. And then it establishes multicast trees among overlay proxy and end hosts. It has good performance that is close to IP multicast. However, the main issue is the deployment of overlay proxy.

Application layer multicast is a hot research topic in recent few years [2~5, 7, 11, 12, 14, 16] and has become an attractive solution of multicast. It is also the easiest one for immediate deployment among these three architectures. ALM is developed on application layer and doesn't need to modify any existent protocol in lower layer. User's desktops or mobile devices (called peer) replace the function of routers in IP multicast. Data packets are replicated at peers and send to other peers in the same application layer multicast service.

2.2 Related Work

We briefly introduce some research about application layer multicast.

2.2.1 Multiple Description Coding (MDC)

The overview of multiple description coding is in [22]. It is a method to encode signal into multiple separate descriptions (streams) and any subset of descriptions can be restore to the original signal with different quality. If users receive more description, the distortion will be less and the quality of restored signal will be higher. One difference between MDC and IDA is MDC is a source coding method and IDA is a channel coding method.

2.2.2 Multiple Stripes

SplitStream [2] and CoopNet [3] are two mechanisms using multiple stripes. Both of them get a result that using multiple stripes will increase the robustness, resilience, and load balance. The detail of multiple stripes will be described in next chapter. There are two main differences between them. First, CoopNet uses a centralized tree building algorithm while SplitStream is decentralized because it bases on Scribe. Second, CoopNet does not handle the

bandwidth contribution of peers in the trees.

2.2.3 Waypoint

There is new overlay architecture in [9]. Besides normal participants, they employ some machines call waypoints. In the overlay tree, these two kinds of nodes are interwoven. It is different to the overlay proxy in overlay multicast. Waypoints are the same as normal participants and they run the same protocol. Therefore, its behavior is the same as other users, rather than statically provisioned infrastructure nodes, such as Overcast nodes in [6]. The purpose of waypoint's participation is to increase the total amount of resource in the system. In their experiences, waypoint is needed in some cases and their investigation is still in progress. Besides, the difference between waypoint and helper will be described in next chapter.

2.2.4 Tree Building Algorithm

The two main categories of tree building are based on distributed hash table (DHT) and hierarchical clustering. Scribe [14] and Bayeux [16] are the ALM tree building methods that base on the Pastry [13] and Tapestry [15] DHT mechanism. In the beginning, the purpose of DHT is to search and route efficiently (with the bound of hops, usually $\log(N)$ hops). Every peer will be assigned an id that generated by hash mechanism and the routing path will base on the hash id. And then, the tree building methods that base on DHT build transmission paths along the routing path by reverse path forwarding (RPF) method. These transmission paths have same source peer, so these paths will form a tree structure.

NICE [4] and ZIGZAG [5] are the ALM tree building methods that base on hierarchical clustering. Each peer who wants to join the tree will be put into a cluster base on specified metrics (such as distance). One or more peers in the cluster will be promoted to a higher level cluster and their responsibility is to transmit data to all peers in lower level cluster. And then, there is also a higher level cluster to send data to the high level clusters. Finally, the only one member in the highest level cluster will be the source peer. These clusters construct a tree

structure and a level of clusters represents a level of the tree.

2.2.5 Enhancement of Resilience

Probabilistic Resilient Multicast (PRM) [11] uses a proactive forwarding approach to increase the data delivery ratio. The method is to use randomized forwarding. Every node randomly chooses a constant number of other nodes and forwards data to them with low probability. The randomized forwarding is simultaneous with the usual data forwarding mechanism. Hence, some nodes may receive the same data. But this approach will make the nodes can receive the data even when their parents fail. And then, it proposes an extension called Ephemeral Guaranteed Forwarding (EGF). When some nodes are repairing their transmission paths (finding new parents), they can request other nodes to temporarily increase the probability of forwarding data to them. Therefore, they still can receive the data in the repairing process.

A proactive approach to reconstruct multicast trees is proposed in [12]. When some interior nodes leave trees or fail, it will minimize the disruption of service for those affected nodes. The approach is every interior node should compute a parent-to-be for each of its children. In the computation process, it will consider the degree constraints of each node. And it also deals with the situation of multiple leaves. After that, if an interior node leaves the tree or fails, all its children will find their new parent (parent-to-be) immediately. By this approach, when some edges of the tree broken, every node can find a new parent quickly and also recover the transmission path.

Chapter 3

Principle

3.1 Overall System

The three approaches we use are information dispersal algorithm, multiple stripes, and helper. Base on these three approaches, we can establish the scheme of application layer multicast system presented in Figure 3.

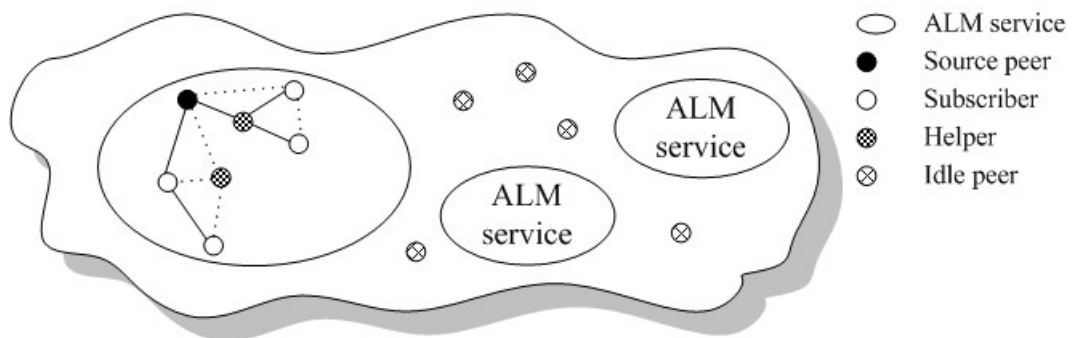


Figure 3: Application Layer Multicast System

The whole graph is an ALM system with many independent ALM services inside. Each service provides different streaming data. The circle nodes are the users who join the system and generally called peer. Each peer has different name based on what they do in the system. Peers who provide streaming data are called source peers. Peers who subscribe the service are called subscribers. Peers who don't subscribe the service are called helpers of the service. Peers who don't subscribe any service are called idle peers and they can be helpers for all services. Now, we point out the place our three approaches performed.

Information dispersal algorithm The dark circle node is the source peer. Before it sends out the streaming data, the data will be processed by IDA.

Multiple stripes In Figure 3, there are two different lines inside the ALM service in left side. The two kinds of line represent two different trees and source peer transmit different

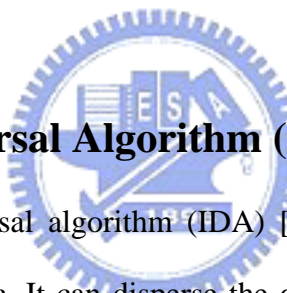
stripes which are generated from IDA through different trees.

Helper The ALM service in left side shows there are two helpers contribute their bandwidth to the service. They help to retransmit stripes to the subscribers of the service.

We have several hypotheses for the ALM system. One, most peers of the ALM system are using asymmetric connectivity. It means they have poor upstream bandwidth. Two, all peers have degree constraint which is determined by the upstream bandwidth offered by individual peers. Three, the data rate of the streaming data provided in ALM system is too high for general ALM mechanism. Four, the number of peers in the ALM system is very large. In other words, the lack of helper is not a problem. Five, the streaming data transmitted in ALM service might have security requirement.

In the following sections, we will introduce the principle detail of the three approaches.

3.2 Information Dispersal Algorithm (IDA)



(n, m) information dispersal algorithm (IDA) [8] is a method that disperses data for security, fault tolerance and etc. It can disperse the original data into n pieces and we must have m pieces or more, $m \leq n$, to be able to restore to the original data. For security, peers can't know any content of data if they have less than m pieces. For fault tolerance, it means that it could tolerate some of pieces missing and still can restore to the original data. Moreover, it has a special characteristic that the m pieces we mention before is any m pieces without any order, needn't to be continues, and no any piece is must have. On the contrary, using IDA will cause the data size much bigger. If the size of data F is $|F|$, we disperse it into n pieces, the size of each piece will be $|F|/m$. Therefore, the total size of n pieces will be $|F| \cdot (n/m)$.

Following we will brief describe how to use IDA in transmission process.

3.2.1 Split

First, we must decide (n, m) and then we use IDA on the transmitted data F . Let the

content of F be b_1, b_2, \dots, b_N , F is divided into N units. We use the content of F to generate a matrix B and the blank places are filled in 0. The size of matrix B is $m \times \lceil N/m \rceil$.

$$F = b_1, b_2, \dots, b_N = (b_1, b_2, \dots, b_m), (b_{m+1}, b_{m+2}, \dots, b_{2m}), \dots \quad (1)$$

$$B = \begin{pmatrix} b_1 & b_{m+1} & \cdots & b_{\lceil N/m \rceil} \\ b_2 & b_{m+2} & & \vdots \\ \vdots & \vdots & & b_N \\ b_m & b_{2m} & & 0 \end{pmatrix} \quad (2)$$

Then we choose n vectors a_i , $1 \leq i \leq n$, and the length of each vector is m . Every subset of m different vectors must be linearly independent. And then we use a_i to compose matrix A that the size of matrix A is $n \times m$.

$$a_i = (a_{i1}, \dots, a_{im}), \quad 1 \leq i \leq n \quad (3)$$



$$A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad (4)$$

It is the initialization step from (1) to (4). Then we will use IDA in the transmission process. First, we will divide F into n pieces. Base on the matrix A and B we calculate before, $A \cdot B = C$, C is a $n \times \lceil N/m \rceil$ matrix. We can divide C into n vectors called c_i , $1 \leq i \leq n$, the length of a vector is $\lceil N/m \rceil$. These n vectors are the pieces we want. And then these n pieces will be transmitted through different transmission paths to all peers (based on multiple stripes approach).

$$A \cdot B = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = C \quad (5)$$

$$c_i = (c_{i1}, c_{i2}, \dots, c_{\lceil N/m \rceil}) \quad (6)$$

$$c_{ik} = a_{i1} \cdot b_{(k-1)m+1} + \dots + a_{im} \cdot b_{km} \quad (7)$$

3.2.2 Restoration

After splitting F into n pieces in previous section, we will show we can restore F from any m pieces. When peer receives m pieces or more, the peer can restore the original F . We choose any m pieces $(c_i, c_{i+1}, \dots, c_{i+m})$ and choose the corresponding m vectors $(a_i, a_{i+1}, \dots, a_{i+m})$ in A . Through the expression below, we can restore the matrix B and also we obtain the original data F .

$$B = \begin{pmatrix} a_i \\ a_{i+1} \\ \vdots \\ a_{i+m} \end{pmatrix}^{-1} \cdot \begin{pmatrix} c_i \\ c_{i+1} \\ \vdots \\ c_{i+m} \end{pmatrix} \quad (8)$$

Therefore, as long as all peers generate the same matrix A as source peer, they could restore the original data from any m pieces of the data.

3.2.3 Efficiency

Only produce matrix C and restore matrix B in IDA will influence on the efficiency of Trickle. It is because the production of matrix A and A^{-1} is only once, we don't need to reproduce. Now, we consider the number of operations split and restoration need.

For split, it needs $n \times m \times \lceil N/m \rceil$ multiplication operations and $n \times (m-1) \times \lceil N/m \rceil$ addition operations. The complexity of operation is also associated with the size of b which is $|F|/N$. Therefore, split is affected by the parameter n and the size of data. The complexity of split is $O(n \times |F|)$. For restoration, it needs $m \times m \times \lceil N/m \rceil$ multiplication operations and $m \times (m-1) \times \lceil N/m \rceil$ addition operations. Therefore, restoration is affected by the parameter m and the size of data. The complexity of restoration is $O(m \times |F|)$.

3.2.4 Advantage

Using the combination of IDA and multiple stripes in transmission process has several advantages. First, security and fault tolerance, these are the design purposes of IDA. These two characteristics quite match our requirement. Because we join the concept of helper into Trickle, security of data transmission is necessary when the data is sensitive. And further, because the whole ALM service is composed by peers (end hosts), the service can't promise peers can receive all data. By the fault tolerance of IDA, the transmission process is more resilient. Second, we only need to receive m of n pieces to restore original data, so we do not need to wait the remand $n-m$ pieces. Because every piece is transmitted through different path, some paths are congested and some are not. Therefore, the thing that influences on our delay of receiving data is the most quick m paths, not all n paths. Even if the congestion of transmission paths will change, we still can prevent to be influenced by most congested $n-m$ paths without changing our transmission paths. Hence, the receiving process is more efficient.

3.2.5 Influence of n and m

Because our IDA and multiple stripes approach will influence each other, the decision of n and m will cause some effect. First, n will influence on how many trees we will build in an ALM service. Too many trees will lead to increase amount of control signal and the control overhead will consume our rare upstream bandwidth. Second, the ratio of n and m will influence on the transmission overhead. As we mention before, the size of data that using IDA will (n/m) times bigger than the original one. It means that the data transmitted through network will increase (n/m) times. However, we don't really need to receive all n stripes. We will explain the improvement in next chapter. In addition, the ratio of n and m will influence on the resilience of ALM service. The larger ratio of n and m is, the more resilient ALM service will be. In other words, we can tolerate more stripes lose. Hence, the decision of n and m is very important and must base on what application the service wants to provide.

3.3 Multiple Stripes/Trees

The concept of multiple stripes has been brought up in previous research [2], [3]. Each message transported from source peer will be divided into several stripes. In tree building step, we not only construct one tree, we construct as many different trees as the number of stripes that one message is divided into. And each tree transmits different stripe of the message to subscribers. Hence, the peer receives different stripes of the message through different transmission paths. Figure 4 present an example of multiple stripes. By this approach, we get a lot of advantages mentioned before. Moreover, we enhance multiple stripes by combining with IDA approach. The description is in following content.

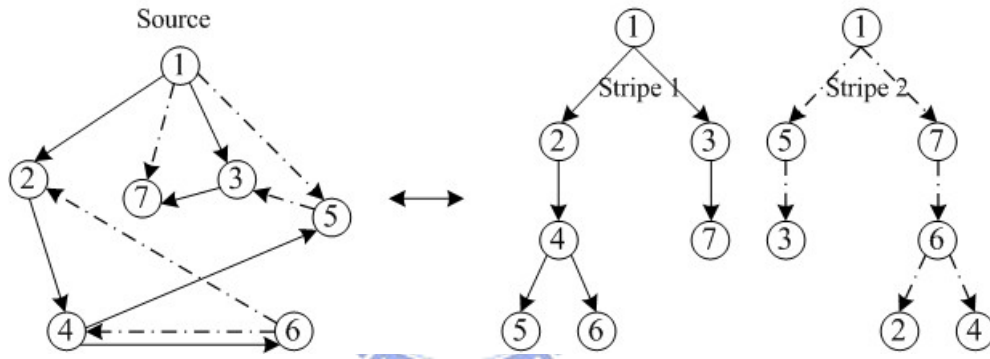


Figure 4: An example that a message is divided into 2 stripes

3.3.1 Combine with IDA

The combination of multiple stripes and IDA mean that one message will be divided into several stripes by IDA approach. And each stripe will be transmitted through different trees. The purpose of multiple stripes/trees is to prevent losing a whole message when one link broken. The purpose of IDA is to promise peers will restore the original message even some stripes lose.

When we use (n, m) IDA technology, every peer joining in the ALM service will have n different transmission paths (actually, it is mostly less than n , we will explain about it in next chapter) that the origination is source peer and destination is itself. Each transmission path of the peer transmits different stripe and peers only need to receive any m stripes of the total n

stripes, then those stripes can be restored to the original message. In other words, the transmission process can tolerate to loss at most $n-m$ stripes.

Therefore, we must make sure that any peer leave the ALM service will not cause the remand peers who are still in the ALM service can not restore the original message. This situation happens only when the leaving peer is in more than $n-m$ of n transmission paths of any peer. When such peer leaves the ALM service, it will cause more than $n-m$ transmission paths of the peer break down and the peer can only receive less than m stripes. Hence, the peer can not restore the original message. This situation can be prevented by building disjoint transmission paths.

3.3.2 Building Disjoint Paths

Base on the description before, we know that we must prevent any peer is in more than $n-m$ transmission path of other peers. The solution for building this kind of transmission paths is to restrict that every peer become interior node in constant number of trees. And the number must not bigger than $n-m$. Therefore, there are two different choices to decide the number of being interior node for each peer. The first choice is every peer becomes interior node in more than one tree and the second choice is to restrict every peer can become interior node in only one tree. The advantage of first choice is that some peers with high upstream bandwidth will contribute their bandwidth averagely in multiple trees without contributing whole upstream bandwidth in only one tree. It is good for Trickle because in transmission process with IDA, the importance of all stripes is the same. However, there is a critical disadvantage to contribute bandwidth in multiple trees. It will cause the height of trees in ALM service much bigger and the end to end delay will be higher because the degree of nodes in each tree are smaller.

Therefore we decide to pick the second choice that every peer becomes interior node in only one tree. This method restrict that any peer at most in one of n transmission paths of other subscribers. And then, we have to use a method to decide peers become interior node in

which tree. In SplitStream [2], it provides a method that can establish interior-node-disjoint trees. Each tree root has different prefix of groupId. Due to the tree building method, groupId and nodeId of peers will cause which peers become interior node. Therefore, it will restrict that every peer will play the role of interior node in no more than one tree and being leaf node in the remand trees. This method is only fit for the tree building algorithms that base on DHT. In our hypothesis, we hope Trickle can be applied to any tree building algorithm. Therefore, we use different method to decide peers become interior node in which tree.

3.3.3 Become Interior Node

Each peer decides it will become interior node in which tree by itself. When a new peer is in the process of joining service, there are two decision methods to decide which tree it should be interior node. One, in order to prevent violating the DHT concept, if the tree building algorithm is based on DHT, it decides to be interior node in the tree whose root has the same prefix hash id. It is approximately the same as SplitStream, but the only difference is the peer already decides the tree it wants to be interior node by itself in the beginning of joining service. The second method is the peer randomly decides which tree it should be interior node and be leaf node in other trees.

The advantage of random solution is that we don't need a server to do resource (upstream bandwidth) management. However, it may probably cause large number of peer being interior node in the same tree and the resource distributed unfairness. It will cause that the upstream bandwidth of some trees is not enough. We solve this problem by using the concept of helper. Those trees with poor resource will ask helper for help. The helper will share their upstream bandwidth to those trees that need it. Therefore, when the number of helper is nearly infinite, the upstream bandwidth won't be exhausted. And the concept of helper will be described in next section.

3.4 Helper

The peers are idle or the subscribers of other services can be the helper of the service. For example, in one service, the peers are regarded as subscribers, but for other services in ALM system, they are not subscribers. Therefore, we call them helper when they are not regarded as subscribers of the service. We hope those peers that are idle or have ample upstream bandwidth can share their bandwidth with other ALM services. Hence, for those ALM services that can not provide smooth streaming data to the subscribers will request helper for help.

3.4.1 Demand of Helper

In order to make every peer receive streaming data smoothly, every peer will restrict the number of children it can support based on its upstream bandwidth. A peer will request to be some peer's child in three cases: (1) the peer is a new subscriber and it chooses a peer in the service to be its parent by tree building algorithm, (2) the transmission path between the peer and source is broken, it wants to change transmission path, (3) the tree structure adaptation algorithm will cause peers to change their parents. When the peer already reaches its upper bound of child number, it receives the request from the other peer that wants to be its child. It is the time to request helper's help in order to accept the request.

3.4.2 Find Helper

There are two issues need to be consider when peers need helper's help.

How to find the helper Because we expect our infrastructure of ALM system is based on decentralized control, not centralized control by some peers, like source peer. Hence, every peer needs to have the capability of finding helper. When every peer joins the ALM system, they will establish a neighbor table to record peers that are close to them. The definition of neighbor is that the number of IP routing hops between its neighbor and itself is below a predefined number. When the peer joins one ALM service and the neighbors of the peer who

are not in this service can be helper candidates (because of the restriction in next section). Hence, peers can request assistance of helper candidates in neighbor table when they need helper's help.

Helper selection The second issue is that if there is more than one helper candidate, which one of candidates we should choose. Because the participation of helper will make the transmission path longer and it will increase the end to end delay of affected peers, we must choose the candidate that increases the transmission path length least. We explain it by an example presented in Figure 5. d_{AH} means the number of IP routing hops from A to helper candidate (who are in the neighbor table of A), d_{HB} means the number of IP routing hops from helper candidate to B . If we need the assistance of helper between A and B , we will choose the candidate that it makes the $d_{AH}+d_{HB}$ to be smallest and then we request the helper candidate to join the ALM service and place it between A and B .

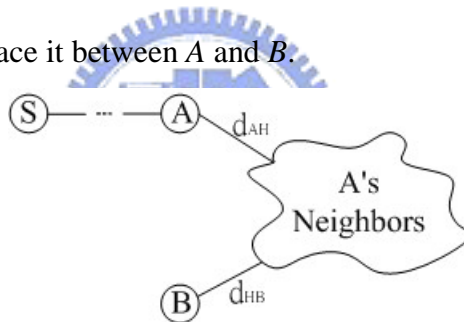


Figure 5: An illustration of helper selection

3.4.3 Restriction of Helper

The selection of helper has two restrictions. First, helper might be the subscriber of other ALM service. We can not request them to share too much bandwidth that will cause them can not smoothly receive the streaming data they want. Therefore, every peer will share the fix ratio of bandwidth for being helper. The ratio of bandwidth it will share is defined by ALM system. Second, for security consideration, we can't allow helper to know the content they transmit. Base on the characteristic of (n, m) IDA, we restrict a helper can help only one tree in one ALM service. It means a helper can only join one tree of the service and become interior node in the tree. Hence, helper can't know the content it transmits because it doesn't have enough stripes. And also, every peer still can receive the whole message when it leaves.

3.4.4 Helper Join

After finding one helper, we need to explain how the helper joins the tree. We explain the process by an example presented in Figure 6. In Figure 6(a), peer *A*'s upper bound of child number is 3. It shows *A* is already reach its upper bound and new peer *E* send request to *A* that it wants to be *A*'s child. It is the situation that needs helper's help. There are three steps to make a help joining the tree. First step, peer *A* chooses helper *H* from helper candidates by the method described in previous section. Second step, peer *H* becomes *A*'s child and accepts *E*'s request, so *E* becomes *H*'s child. Third step, peer *A* transfers its child *D* to *H* and *D* becomes *H*'s child.

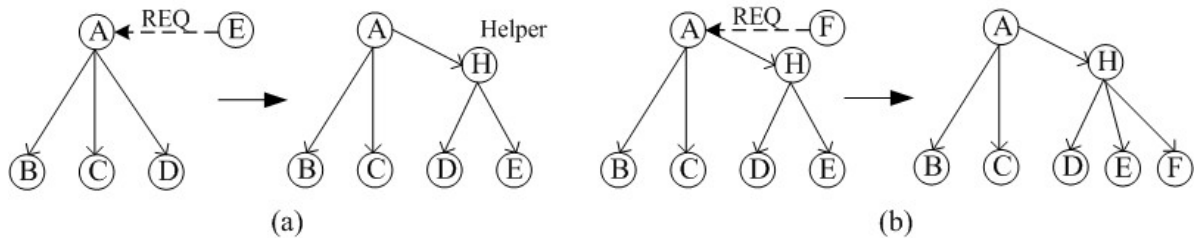


Figure 6: An example of helper join

The decision of transferring which one of *A*'s children to be *H*'s child is based on two rules. Rule one is leaf node has higher priority and rule two is choose the one closest to helper. If some of *A*'s children are leaf nodes, we choose the leaf node that is closest to *H*. Otherwise, we choose the one of *A*'s children that is closet to *H*. We choose leaf node first because it won't affect too much peers who will increase end to end delay.

In this approach, the ALM service can successfully accept peer *E*. It will bring two advantages. First, *E* request to be *A*'s child who is already reach its supporting upper bound must because it is *E*'s best choice or the only choice. Because we expect our infrastructure is decentralized control; usually, *E* can only obtain a small part of peers' information in the ALM service. Therefore, the parent choice of *E* is limited. Hence, base on this approach we could successfully accept *E* and also try our best to prevent to influence on peers that are already in the service. Second, because *H* is *A*'s neighbor, when *H* joins the service, the peers who might

choose A before could have another choice H . Therefore, peers can have more choices. Moreover, H still not reaches its upper bound of child number. It increases the number of peers the tree can support.

3.4.5 Enhancement

Enhancement 1 In finding helper step, we don't really need to choose the neighbors that are not in the service to be helper candidates in an exception. When the neighbor is already in the service, if the tree needs help is the same tree it already in, we still can request its help. In Figure 6(b), a new request from F , if the best choice of helper is still H , then A still can request H 's help if H doesn't reach its upper bound of child number. In addition, in this case, child transfer step can be omitted.

Enhancement 2 By enhancement 1, the peer may find a helper that is other peer's child. In this case, the helper can change parent if the distance from source peer to it will be shorter. We explain it by an example presented in Figure 7. d_{SA} is the distance between source peer and peer A , d_{SB} is the distance between source peer and peer B , d_{AH} is the distance between peer A and peer H , and d_{BH} is the distance between peer B and peer H . When peer B receives a request from Z , B detects the best helper is H and requests H 's help. There are two possible results. If $(d_{SA}+d_{AH}) \leq (d_{SB}+d_{BH})$, the result is presented in Figure 7(b). If $(d_{SA}+d_{AH}) > (d_{SB}+d_{BH})$, the result is presented in Figure 7(c). By this enhancement, not only helper will have lower end to end delay, but also all its children and descendants.

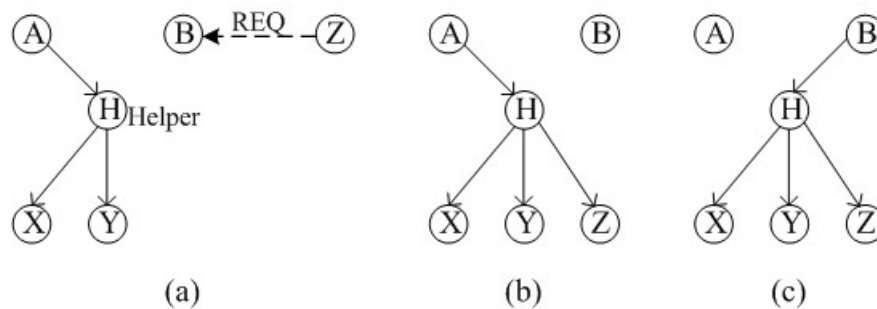


Figure 7: An example of helper approach enhancement

3.4.6 Difference between Helper and Waypoint

There is already a similar concept to helper called waypoint [9]. Both helper and waypoint are the peers with unnecessary resource, such as upstream bandwidth, and they provide it to help others. The most different point between waypoint and helper is that waypoint joins the service base on its own decision and helper joins the service when members in the service call for help.

The concept of waypoint is to treat waypoint as common peers. It means that the method of joining service and the data it receive is the same as common peers. Even though waypoint will join the service that need help base on its determination, its join method is the same as other peers. It may not be put in the urgent place that needs help. For example, there is one sub-tree in the service needs help. Waypoint may join this service, but may not be put in that sub-tree by the join mechanism.

Contrary, helper is requested by the members in the service who need help. And the helper joins the service using different joining mechanism. Helper will be put in the place that really needs help. Therefore, participation of helper will solve the issue immediately.

Chapter 4

Mechanism

In this chapter, we will introduce the mechanism of ALM service and the integration of our three approaches in the mechanism. We separate the mechanism into two parts. The first part is the start of an ALM service and the second part is the behavior of peers.

4.1 Service Startup

There are three things need to be done to start an ALM service in the ALM system. First, every peer should establish a neighbor table before join or start a service. Second, if a peer wants to establish an ALM service to share its streaming data, it must decide who the source peer of the service is. Third, the source peer must configure IDA before accepting subscribers. We describe the details in the following content.

4.1.1 Establish Neighbor Table

Before ALM service start, every peer who wants to join the ALM system, including source peers, subscribers, and even helpers, should establish their own neighbor table. The establishment of neighbor table is prerequisite for our helper approach.

The negotiation between peers and their neighbors is presented in Figure 8. We define a neighbor finding signal. In that signal, we set within how many hop count the peer can be called neighbor and the signal is sent by broadcast method. The signal should be drop when it travels farther than neighbor distance. When peers receive a neighbor finding signal, they should reply an ack signal to original sender.

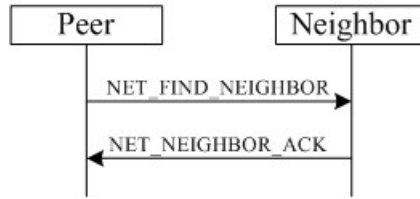


Figure 8: Negotiation between peers and neighbors

If a peer receives an ack signal from other peer, it means it find a neighbor. The peer should do the reaction presented in Table 1. If the neighbor table is not full, it just adds the sender of the ack signal into the table. If the neighbor table is full, when the sender is closer to it, it just adds it into table and removes the neighbor that is farthest to it. After repeatedly receive the ack signals and do the reaction described above, peer will obtain a neighbor table with closest neighbors inside. Be notice that the neighbor table should resize if there is no any available neighbor when peers need neighbor in Trickle.

<pre> //Receive neighbor n's ack message //Hop count between n and itself is h If neighbor table is not full Insert n into neighbor table (sorted by hop count) Else If h is smaller than any hop count of neighbor in neighbor table Insert n into neighbor table Remove the last element in neighbor table </pre>
--

Table 1: Reaction after receiving a neighbor ack message

4.1.2 Source of Service

Source peer is the tree roots of the service. It splits streaming data into IDA stripes and sends stripes to its children of each tree. But which peer is the source peer? Is the streaming data owner should be the source peer of the service? It is possible, but not necessary. There are two possible cases.

Source peer is the data owner When the data owner is a peer with sufficient upstream bandwidth and processing capability, such as multimedia service provider, it could be the source peer of the service.

Source peer is not the data owner When the data owner is an individual user, it

probably only can unicast the data to one child. All it can do is to send the streaming data to a proxy server (we call source proxy) with capability of being source peer and the IDA is done by source proxy. It is presented in Figure 9. The source proxy can be the e-home server. Users send data from their mobile devices to their e-home server and then the server multicasts data to numerous peers. Or the source proxy can be provided by ISP with commercial cost. There still has a lot of possibility about the source proxy. It can solve the issue that data owner is not powerful.

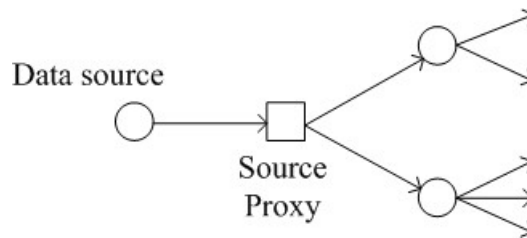


Figure 9: An illustration that source peer is a source proxy

4.1.3 IDA Configuration

4.1.3.1 (n, o, m) IDA

After deciding which peer is the source peer, source peer must decide the parameters of IDA. In the previous chapter, we mention we will combine multiple stripes and IDA. However, we don't adopt (n, m) IDA but (n, o, m) IDA. All we want to do is to decrease bandwidth consumption from the overhead of IDA.

First, n means in the ALM service, there will be n different trees. The message generated from source peer will be divided into n different stripes and each stripe will be transmitted in different tree. Second, m means every peer that join the service must receive at least m stripes of a message to restore to the original message. Third, o means that when every subscriber wants to join the ALM service, they should join o trees initially, $n \geq o > m$. We have to explain about the range of o , $n \geq o > m$. The IDA operation is decided when the ALM service starting, so n and m can't be changed. Hence, there will be totally n trees, so o must less or equal n . And o must bigger than m is because we don't want when any peer leave the service will cause any other peers that is still in the service can not receive m stripes of the message.

Due to we restrict every peer can be interior node in only one tree, so the lower bound of o is $m+1$. And if we want to decrease the consumption of the bandwidth, we must set o closer to m . Because $o > m$, it still can provide a little fault tolerance in normal situation.

4.1.3.2 Apply to Streaming Data

Now, the source peer can apply IDA to streaming data when subscribers join the service. The (n, o, m) parameters will influence on two things: the size overhead and the fault tolerance of the service. If the size of unit message needs to be encoded by IDA is B and we apply (n, o, m) IDA to it, the size of one stripe will be B/m and the total size of stripes is $n \times B/m$. Each subscriber need receive o stripes with size $o \times B/m$ in the beginning. The ratio of lost stripes subscribers can tolerate is between $(o-m)/o$ to $(n-m)/n$.

4.1.3.3 Discussion

The reason we add the parameter o is because when a message is processed by IDA, the size of the message will be (n/m) times bigger. If every subscriber joins n trees in the beginning, it will consume too much downstream bandwidth of itself. ($(n-m)/m$ of data is redundant.) In other words, it will consume too much upstream bandwidth of other peers. In order to prevent it happens, we must make every peer to receive redundant stripes as less as possible.

First, we must consider why peers need to receive redundant stripes. The peer receive more than m stripes of a message is because we promise when some stripes delay or lost, they still can restore to the original message without any delay (there are the upper bound of tolerance). Therefore, we need to receive more stripes (join more trees) of one message only when some stripes' transmission process occur problems. But we still need to receive a little part of redundant stripes ($o-m$ stripes) in normal situation. Otherwise, we will lose one or more message when we are doing remedial action. Oppositely, if the transmission process of stripes has no problem, we should receive stripes as less as possible (join less trees) to prevent the consumption of bandwidth. Hence, after the peer joins o trees in the beginning, every peer

should adjust the number of trees they join dynamically. The method about how to decide the adjustment will be described in later section.

4.2 Behavior of Peers

First, we describe the architecture of peer behaviors. And then we describe the detail of each behavior. The order of behaviors is organized along peer life cycle.

4.2.1 Architecture of Peer Behavior

The architecture of behaviors for peers that will be subscribers or helpers is presented in Figure 10. Arrow symbols mean the direction of data flow or control signals. It shows the interaction of each function and the detail of important behaviors will be described in following sections.

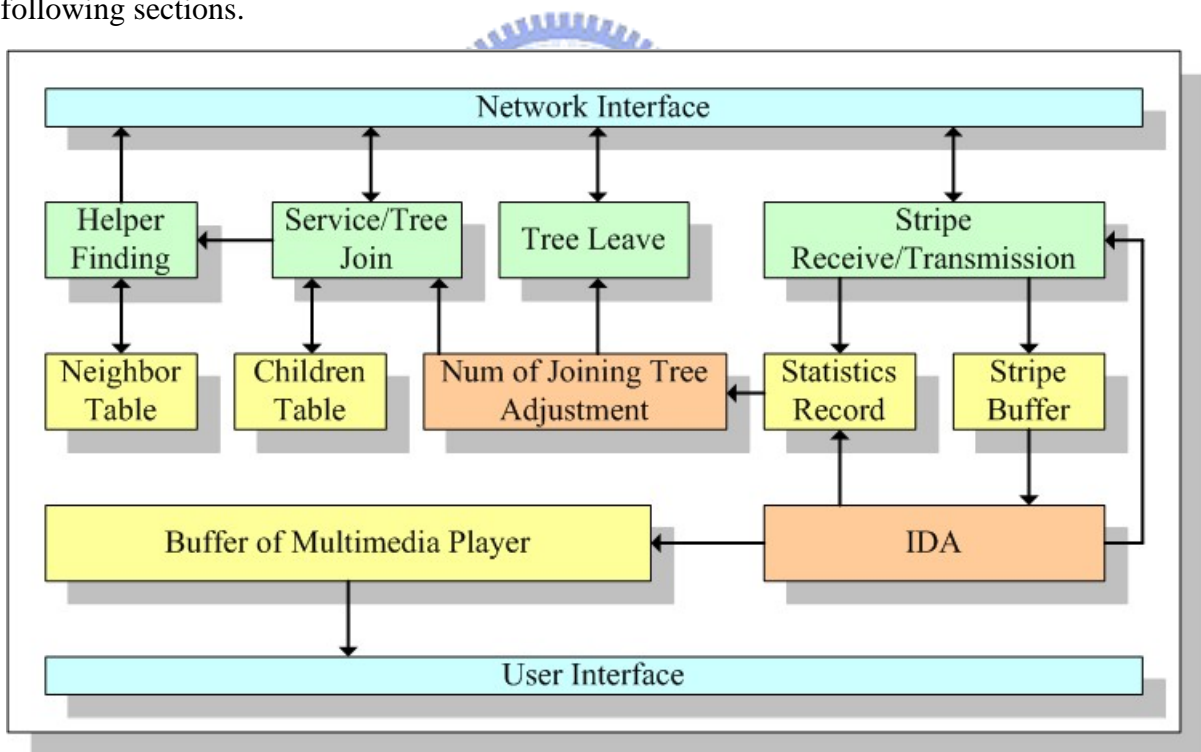


Figure 10: Architecture of peer behavior

4.2.2 Join

4.2.2.1 Randomly Join o Trees

When the peer decides to join an ALM service, it will know the n , o , and m parameters of the service. Hence, the peer knows it must join o trees of the service initially. In Trickle, the

method that peers decide they should join which trees is random selection by themselves. The reason is the same as randomly select a tree to be interior node. We don't want to add a load balance manager or request any peer, such as source peer, to be responsible for it. Management of network load will increase the load of peer and also waste bandwidth.

Let us consider the tradeoff of load balance. First, it will make every peer fairly consumes upstream bandwidth. It is not an advantage for our hypothesis. For our helper approach, because the lack of upstream bandwidth, if some peers don't spend all their upstream bandwidth in the service, they will possibly spend remained upstream bandwidth in other services. Therefore, there is no benefit in this point. Second, load balance will lead the peers distributed averagely. In other words, the height of each tree in the service will be average. It is really an advantage because the end to end delay of all peers will be average. However, in Trickle, we have done several steps to reach similar effect. One, we make every peer join o trees. In worst case that all subscribers join the same o trees, the load is still averagely distributed into o trees. Two, because of IDA, peers can ignore some worse stripes with high end to end delay in certain trees. Three, the trees and the number of trees each peer joins will change to have better performance. It will be mentioned in following section. Of course, load balance is the best choice, but by these three steps, the end to end delay of the peers will be lease and more average. Furthermore, we don't need to waste bandwidth to maintain load balance. Therefore, we decide that every peer randomly chooses o trees to join. And then they randomly choose one of o trees to be interior node.

4.2.2.2 Choose Parents

After each peer chooses which o trees they want to join, they must decide which peer should be their parent in each tree. The selection method of parent is based on tree building algorithm. Our purpose is to devise an approach that is able to suit any tree building algorithm for performance improvement. So any kind of tree building method, like distributed hash tables (DHT) [14], [16] or hierarchical clustering [4], [5], will be fine. The method only needs

to build one tree and other tree can also be built by the same method. Therefore, each new peer will find o parents in different o trees through tree building algorithm.

No matter what tree building algorithm the ALM system choose, we have to add two restrictions for the algorithm. The first restriction is it can't choose peer that plays the role of leaf node to be any peer's parent. The second restriction is every peer should have an upper bound of supporting children base on their upstream bandwidth for performance consideration.

4.2.2.3 Cases of Join Accept

There are two cases when peers receive a join request. The first case is presented in Figure 11. If the peer plays the role of interior node in the tree and still has quota to accept new child, then it accept the request and put the id of request sender into its children table.

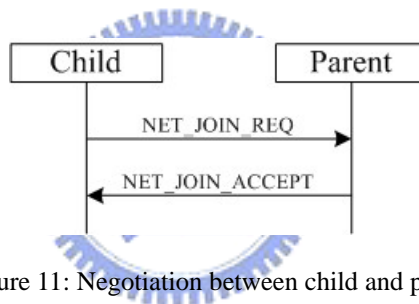


Figure 11: Negotiation between child and parent

The second case is presented in Figure 12. If the peer that receives a join request already reaches its upper bound of child number, it will request a helper for help. And it also needs to transfer one of its children to be the helper's child. The detail of using helper is already described in previous chapter. Sometimes, request helper's help will fail when helper receives the request and it already reaches its upper bound of child number or it already joins other tree. Figure 13 presents the failure of requesting helper.

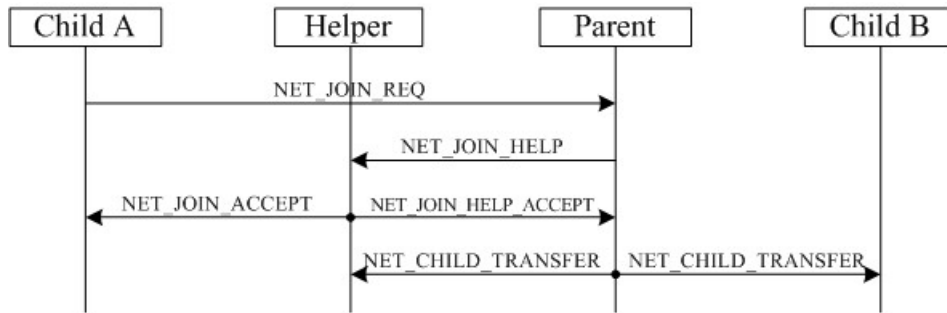


Figure 12: Negotiation between child, parent, and helper

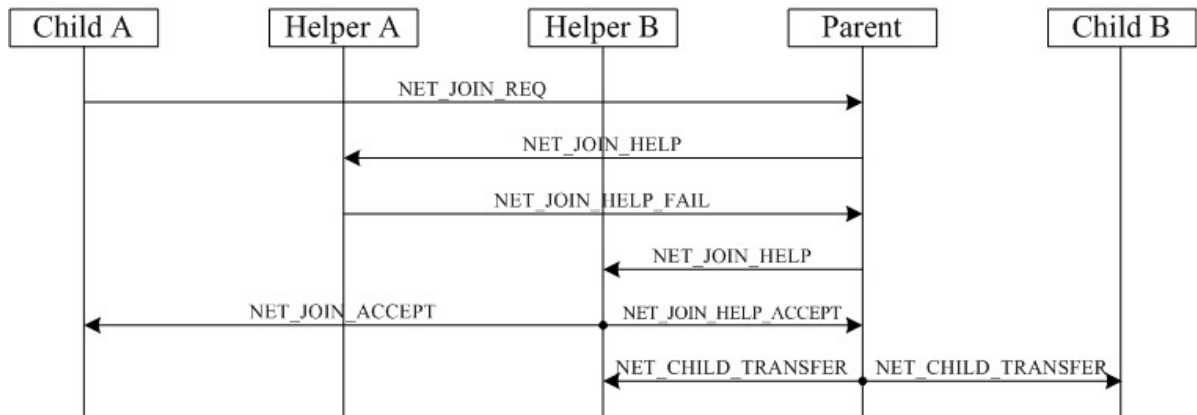


Figure 13: Negotiation after the failure of requesting helper

We combine the two cases into one flowchart and it is presented in Figure 14. And the detail reaction when parent receive the join request from a peer is presented in Table 2.

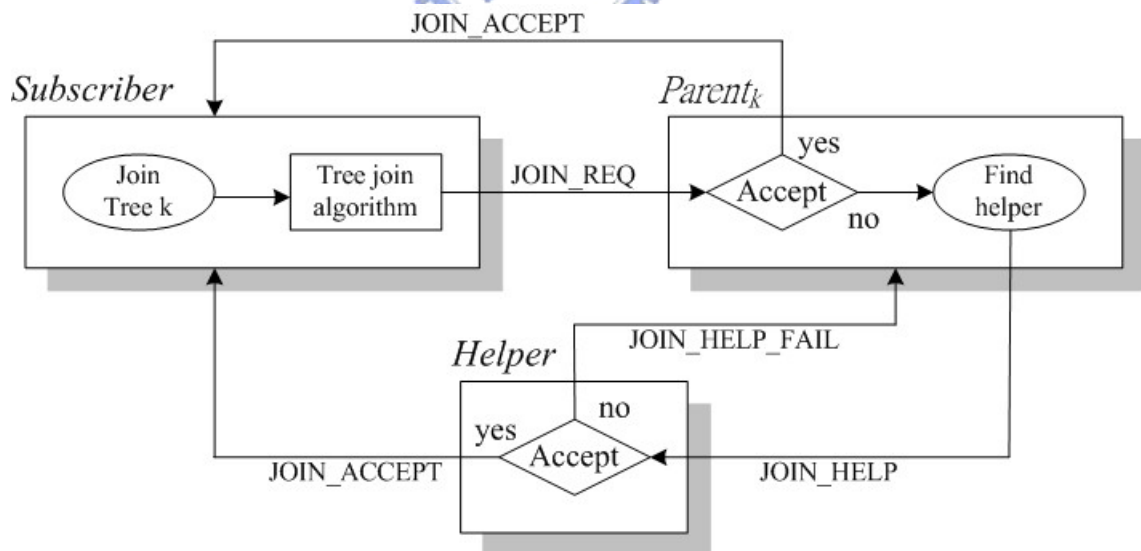


Figure 14: Flowchart of subscriber join

```

//The peer p receive join request from new peer r
If p is not reach upper bound of child number
  Accept r and put r into child table
  Child number++
Else
  Find a neighbor n in neighbor table that  $\overline{pn} + \overline{nr}$  is minimum
  Put n into child table
  Forward r's request to n
  If there is any leaf node in child table
    Find a child c that is leaf node and  $\overline{nc}$  is minimum
  Else
    Find a child c that  $\overline{nc}$  is minimum
  Inform c to change parent to n
  Remove c from child table

```

Table 2: Reaction after receiving join request

4.2.3 Receive and Retransmit

For source peer, transmission process is simple. Its responsibility is to generate n stripes using IDA from original data and transmit each different stripe to its corresponding children set in different trees. It is presented in Figure 15. And for peers that are leaf nodes in the tree, all they have to do is wait and receive stripes transmitted from parents. But for other peers that are interior nodes in the tree, there are two different ways to do the retransmission.

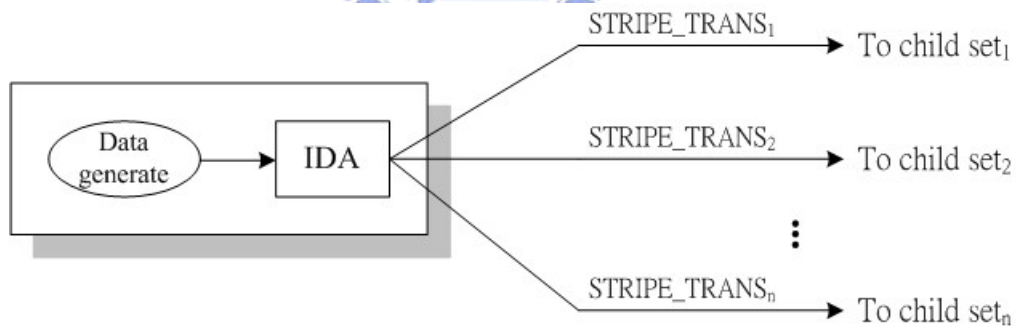


Figure 15: An illustration of stripe transmission for source peer

4.2.3.1 General Retransmission

When the peers play the role of interior node in the tree, they have responsibility to retransmit the corresponding stripe to their children. For example, if the peer is interior node in tree k , it must transmit stripe k of the message to its children in tree k . In general case of retransmission, when peers receive a stripe from parent in the tree that they serve as interior node, they just retransmit the stripe to their children. It is presented in Figure 16. So they have

done their responsibility. This is the basic concept of any ALM mechanism.

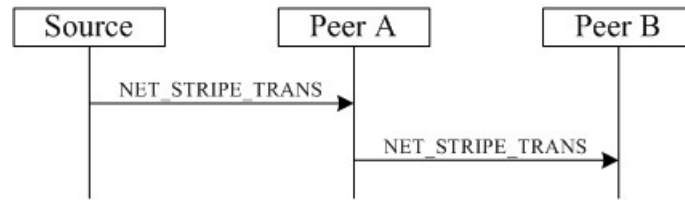


Figure 16: Peers retransmission

4.2.3.2 Stripe Regeneration

Because we use multiple stripes and IDA approaches, peers can regenerate any stripe they need to retransmit (even if they never receive it) and transmit to their child. The algorithm of stripe regeneration is presented in Table 3. When the peer already receives m stripes of the message from other trees but not including the stripe it need to retransmit, it can use these m stripes and matrix A of IDA to restore them to the original message. And then base on the original message and matrix A , it can generate any one of n stripes it needs.

<pre> //Being interior node in Tree k //Receive stripe t now If t=k If still not retransmit stripe k to children Retransmit stripe t to children Else if already exactly receive m stripes (including stripe t) without stripe k Generate stripe k and transmit to children </pre>

Table 3: Reaction after receiving one stripe

For example, in Figure 17, peer A join 5 trees to receive stripe S_1, S_2, S_3, S_4, S_5 from different transmission paths and the IDA parameter m is 4. Peer A serve as interior node in tree 3. Therefore, peer A need to transmit S_3 to its children. When A only receive $S_1, S_2, S_4,$ and S_5 but S_3 , it still can generate S_3 from other stripes and transmit S_3 to its children.

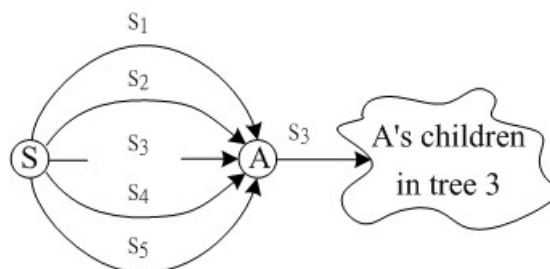


Figure 17: An example of stripe regeneration

Stripe regeneration has several advantages. First, if the stripe needed to retransmit is late or lost, parent still can generate it and transmit to its children. Therefore, parent will prevent the increase of children's end to end delay or prevent they can't receive the stripe. Second, when the peer want to change parent (because transmission path broken or performance consideration) in the tree that it serves as interior node, it still can serve its children during changing parent. It also prevents when a peer leave, all its descendants need to find a new parent to repair the transmission path. Only the children of leaving peer need to do the remedial action.

4.2.4 Adjust Number of Joining Trees

We have mentioned before that sometimes peers must decrease the number of joining trees (it means peers decrease the number of receiving stripes) to reduce the consumption of bandwidth. And sometimes peers must increase the number of joining trees when transmission quality is not stable to prevent end to end delay rising or even message loss. Therefore, we take the dynamic adjustment method to change the number of joining trees to adapt to various situations.

4.2.4.1 Foundation

We must define the rules about when peers need adjustment and how to adjust. Every peer will check every period of time that the success number of message restoration (as long as receive any m stripes of one message, it means a success) in the interval and the interval is predefined by the ALM system. And then we set the upper bound and the lower bound of success number. When the success number of restoring message is not between upper bound and lower bound, the peer will adjust the number of joining trees. Every adjustment will increase or decrease only one tree because transmission issues might be impermanent. We can not predict the transmission issues will continue or not, so we adjust the number of joining trees gradually.

Besides, the decision of upper bound and lower bound of success number will influence

on the message lost rate we can accept. If we don't want any message loss, we should set the upper bound and lower bound higher. It means peers will join trees easier and leave trees harder.

4.2.4.2 Increase and Decrease

Increase When success number is lower than lower bound, the peer will randomly (we will explain about "random" later) join one more tree. If in next check, the situation is still the same, then the peer will join one more tree until the situation is better or the number of joining trees reaches the upper bound n .

Decrease When success number is equal or higher that upper bound, the peer will leave one tree that behaves worst. If in next check, the situation is still the same then the peer will leave one more tree until the situation is worse or the number of joining trees reaches the lower bound o . The tree that behaves worst means in the interval the peer received stripes that transmitted in the tree is fewest.

Figure 18 present the increase and decrease number of joining trees. We can know the quality of receiving streaming data of each peer by observing their number of joining trees. If the peer's number of joining trees keeps in the lower bound, it means the peer can receive the streaming data smoothly. On the contrary, if the peer's number of joining trees keeps in the upper bound, it means the peer receive the streaming data intermittently.

Subscriber

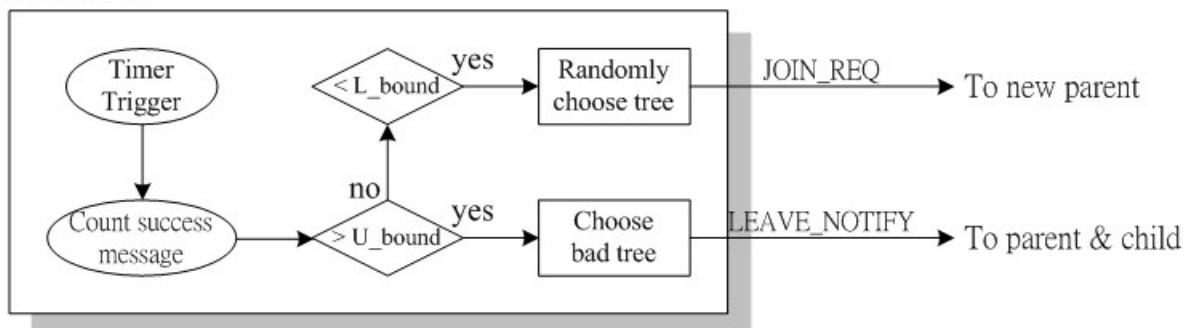


Figure 18: Flowchart for adjusting the number of joining tree

Base on the adjustment method we describe above, every peer will gradually join or leave trees and the trees every peer join will change slowly. Finally, the trees of peer joining

will be stable. It means that every peer will join the suitable trees. The suitable trees mean that in the positions of the peer in these trees will receive the stripes with close end to end delay and the delay will be lower than before adjustment. As long as the end to end delay of each stripe for a message is close, the peer can restore the original message from the buffer that stores the stripes. Otherwise, because of the limitation of buffer size, some stripes that are still not restored to message will be discarded.

4.2.4.3 Priority of Joining Tree Selection

When peers leave the tree that behaves worst, they will put it in the last choice when choosing the tree to join. Therefore, randomly join one tree with low success number is not really to randomize the choices every time. Every peer will firstly randomly sort the tree id that is not joining and put them into a join queue. Every time peers need to join one more tree, they will base on queue and join the first tree in the queue. And the tree they leave will be put in the last of the join queue. Therefore, the selection of joining tree has priority. It is presented in Figure 19.

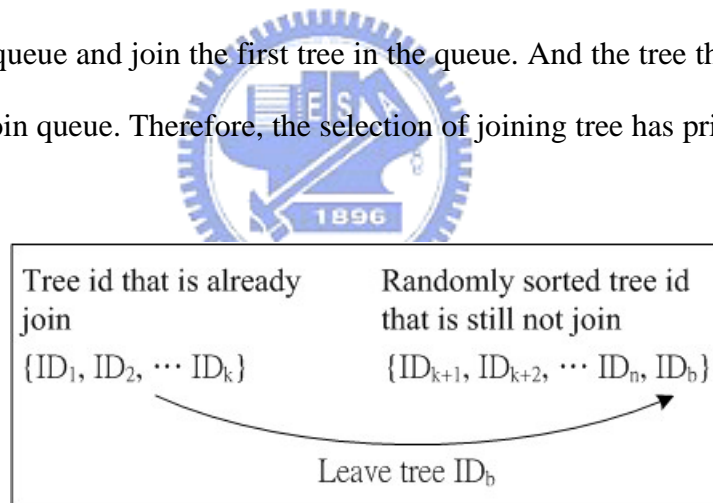


Figure 19: An illustration of priority queue for tree id

4.2.5 Leave

When some peers leave one tree or even the ALM service, the reaction of their children is to repair the transmission paths. And only the children of leaving peers will repair the transmission paths because these children can temporarily use the method of stripe regeneration to serve their descendants and their descendants won't detect any wrong.

The repair method is the same as tree building algorithm that is not included in Trickle. If the tree building algorithm includes the path repair mechanism, we adopt the mechanism.

Otherwise, we can treat these children peers as new peers and use the tree building algorithm to find a new parent. No matter what mechanism we choose, the mechanism will help these children peers find new parents. After finding new parents, the step will be the same as tree joining step we mention before. Hence, these children peers can repair their transmission paths.

You may be doubt that since the service has n trees, why these children peers choose to repair the transmission paths in original tree but not joining other trees. Because these children peers might have their descendants, if they join other trees that they still not joining, their descendants might be already in the new tree. This method will cause some peers join the same tree twice. It doesn't make sense for a peer to receive the same stripe twice. Therefore, we choose the repair action for the children of leaving peers.



Chapter 5

Performance Analysis

5.1 Purpose

The purpose of the simulation in this chapter is to observe the properties of Trickle. The three properties include:

- **Message delay** We will measure the delay of each stripe and delay of message restoration for each subscriber. It will show the perturbation of message restoration delay. Moreover, we will compare with IP multicast to show the difference.
- **Network resource usage** We will analysis the bandwidth consumption and the amount of links it demands.
- **Successful restoration rate** It will show the robust performance with links/nodes failure condition.



5.2 System Configuration

We use two existent tools to do our simulation. First one is Georgia Tech Internetwork Topology Models (GT-ITM) [10]. It is a random graph generator. It can be used to generate transit-stub style network topology. Second one is a discrete event simulation system called OMNeT++ [17]. We use it to simulate the behavior of peers based on the mechanisms we proposed in the previous chapter. We use GT-ITM to generate the network topology and import it into OMNeT++. In the following content, we describe the configuration of simulation system.

5.2.1 Network Topology

First, we generate a network that contains 380 routers. There are 4 transit domains and

with an average of 5 transit nodes in each transit domain. For each transit node, it is attached an average of 3 stub domains. In each stub domain, it contains an average of 6 stub nodes. Therefore, there are 20 transit nodes and 360 stub nodes. Second, we randomly attach hosts to stub nodes and the total number of host is 1024. The result of network topology is presented in Figure 20. In these 1024 hosts, we randomly choose 50 hosts to be the subscribers of the ALM service. Because we will simulate a lot of characteristics of network links, it will cost a lot of time to run the simulation. For feasibility, we don't simulate on a large-scale network.

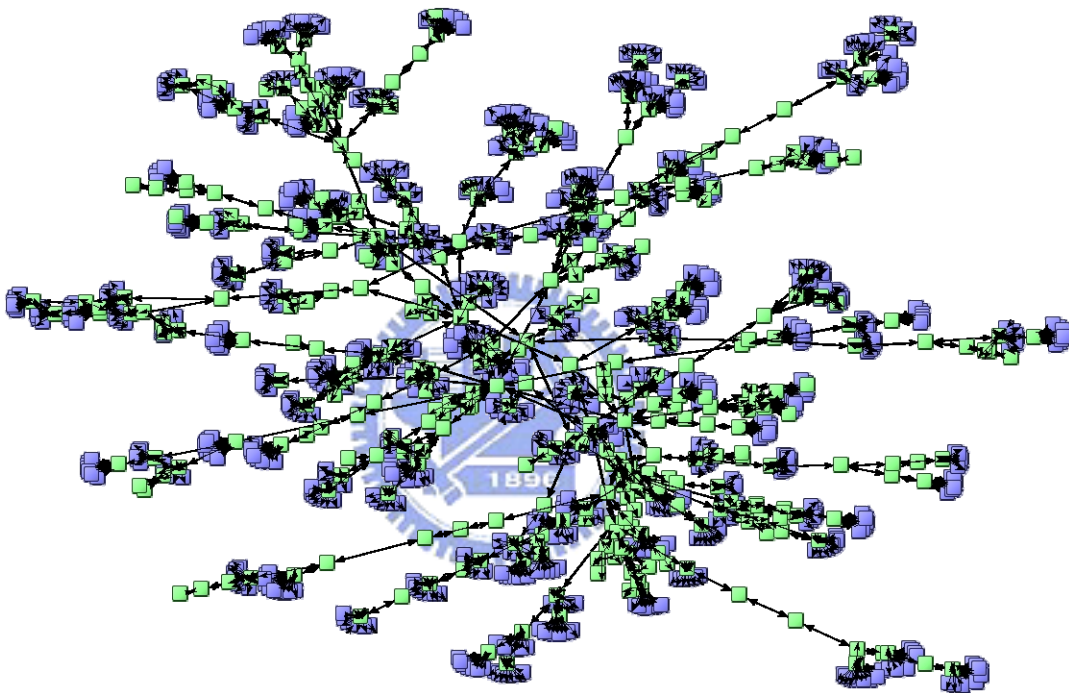


Figure 20: Network topology

5.2.2 Link and Traffic Condition

After importing the network topology into the simulator, we still need to set several link properties to make the simulation more realistic. They are:

- **Propagation delay** Propagation delay = length / (2×10^5 km/s). We set the propagation delay base on the length parameter of links which is generated from GT-ITM.
- **Bit error rate** We set the bit error rate to be 10^{-10} .
- **Bandwidth** In order to simplify the simulation, we assume there are only two different kinds of connectivity between routers and hosts. Because of our hypothesis, we set 60%

of links between router and host to be asymmetric connectivity. It means the downstream bandwidth and upstream bandwidth of these links are asymmetric. We set the downstream bandwidth to be 2 Mbps and the upstream bandwidth to be 512 Kbps. For the other links between routers and hosts, we assume they are LAN connectivity.

Moreover, in order to make the simulation more realistic, there will be other network traffic to consume network bandwidth during our simulation process. The simulation of bandwidth consumption is called self-similar. We use the existent Fractional Gaussian Noise (FGN) [18] implementation to simulate self-similar. The parameters of FGN is $H=0.7$. The range of queuing delay is between 0 to 20 ms.

5.2.3 Trickle Setup

Because we lack tree building algorithm in Trickle, we implement our approaches based on DHT tree building method of SplitStream. Because of the number of hosts, the parameters of routing table are set as $b=4$ and row number=3. Most steps of tree building are the same as SplitStream. The only difference is that when a peer reaches its degree constraint and receives a new join request, it will use helper approach instead of SplitStream's.

There are several parameters need to be configured. For (n, o, m) parameters, we set $n=20$, $o=17$, and $m=16$. Therefore, there will be 20 different trees in the service and each new subscriber will randomly join 17 trees. When subscribers receive any 16 stripes of the message, they can restore to the original message. By this setting, the size overhead of one message subscribers received is between 6.25% and 25%. If most transmission processes of stripes are successful, the overhead should be close to lower bound. For the parameters of adjusting the number of joining trees, we set the time period be 20 second. It means a subscriber will check if it needs to adjust the number of joining tree every 20 second. And then we set upper bound of successful rate to be 100% and the lower bound to be 90%. It means if the number of messages that subscribers successfully restored are equal or more than 100% of message number that source peer sends within 20 second, they will leave one tree if

needed by the rule described in previous chapter. If the subscribers restore less than 90% of messages within 20 second, they will join one more tree.

5.2.4 Streaming Data Setup

For the setting of streaming data, we assume the average bit rate of streaming data sent from source peer is 300 Kbps and it includes 30 frames per second (the size of streaming data packet sent from source peer is variable [23]). For simplicity and without considering video codec, each frame will be fragmented by IDA before sending out from source peer. The average size of one frame is 10Kb. One frame will be divided into 20 stripes and the size of each stripe is $(10 \text{ Kb}/16) = 640$ bits. Because the stripe size is too small, we must consider the overhead of packet header. We set the size of packet header to be 320 bits. Therefore, the size of one stripe will be 960 bits.

Hence, each interior node of trees should transmit $(960 \text{ bits} * 30) = 28.125 \text{ Kb}$ per second to one child. By the setting of stripe size, we set the maximum number of children one subscriber or helper can support is 8. The peers only need to contribute upstream bandwidth at most $(28.125 \text{ Kb} * 8) = 225 \text{ Kb}$ per second. Even a peer with asymmetric connectivity, it only needs to contribute less than half of their upstream bandwidth. Moreover, in order to do comparison, we also run IP multicast in the same condition described above. We assume the average size of one packet in IP multicast is 8 Kbit.

5.3 Experiment

5.3.1 Metrics

In the simulation process, we will measure several metrics to stand for our performance. The four metrics are restoration delay, relative delay penalty (RDP), link stress, and probability of success restoration. RDP and link stress need the statistics of IP multicast.

Restoration delay It represents the end to end delay from source to subscribers. It is

important for real-time streaming service. Delay will influence on the time subscribers need to wait to receive the first message of the streaming data and as well as the following messages. We define two symbols: T_1 means the time source peer send out the first stripe of the message. T_2 means the time subscriber receives the m -th different stripe of the message. Subscribers only need to receive m different stripes to restore to the original message. Hence, the definition of restoration delay is $T_2 - T_1$. It shows that the time of receiving stripes after m -th stripe will not influence on the restoration delay.

Relative delay penalty The definition of RDP is the ratio of delay between Trickle and IP multicast. We will show two kinds of RDP. Relative maximum delay (RMD) is the ratio of maximum delay between Trickle and IP multicast and relative average delay (RAD) is the ratio of average delay between the two approaches.

Link stress We will count the number of network links used in Trickle and the number of links used in IP multicast. And then we calculate the amount of packets going through each link. Thus, we can obtain total bandwidth consumption of these two approaches and also the ratio of average amount of traffic in one link between Trickle and IP multicast. Therefore, we can have a comparison of bandwidth consumption between Trickle and IP multicast. It will also show the overhead of using IDA approach.

Probability of success restoration It is the ratio that during the evaluation period, the number of message the subscriber successfully restored divided by the number of message the subscriber should restore. The reasons that cause subscribers can't restore all messages are bad stripes and lost stripes. Bad stripes are because of bit error rate of links and lost stripes are because of peer failure. It will show how resilience of Trickle.

5.3.2 Procedure

We will run the simulation several times with different procedures. The procedures of the first round are:

1. All subscribers will randomly join the service within 3 minutes (simulation time). The helper and multiple stripes approaches are used here.
2. After first subscriber joining the service, source peer starts to transmit the streaming data. IDA and multiple stripes approaches are used here.
3. Retransmission and adjustment of joining tree are according to the mechanism we described in previous chapter.
4. After 5 minutes, we starts to record the statistics of the simulation.

The procedures of the second round are:

1. All subscribers will randomly join the service within 3 minutes (simulation time).
2. After first subscriber joining the service, source peer starts to transmit the streaming data using IP multicast.
3. After 5 minutes, we starts to record the statistics of the simulation.

The procedures of the remaining rounds are the same as the procedure 1 to 4 in the first round.

The difference is peers have failure probability. These rounds are with different peer failure probability. The probability of peer failure will influence on probability of success receiving one stripe and message restoration. Suppose the probability of peer failure is P and the path from source peer to the subscriber in tree k will pass through D peers. Then the subscriber's probability of success receiving stripe k is $(1 - P)^{D_k}$.

5.4 Results and Analysis

In this chapter, we will analysis our results in the order of simulation purpose. In 5.4.1, we observe the message delay from three different viewpoints. They are delay of stripe and message restoration for each subscriber, delay of message restoration for all subscribers, and

relative delay penalty which is compared to IP multicast. In 5.4.2, we observe the link stress to find out the usage of network resource. In 5.4.3, we observe the probability of success restoration to stand for resilience.

5.4.1 Restoration Delay and Stripe Delay

5.4.1.1 Delay for Individual Subscriber

We plot 4 histograms to show the delay of stripes and the delay of message restoration for each subscriber. Because the parameter $n=20$, if we plot all stripe delay, the histogram will be confused. Therefore, in the histogram for one subscriber, we only plot 6 curves of stripe delay that is closest to the curve of message restoration delay. It is because the curve of message restoration delay is only influenced by the curves of stripe delay that is close to it.

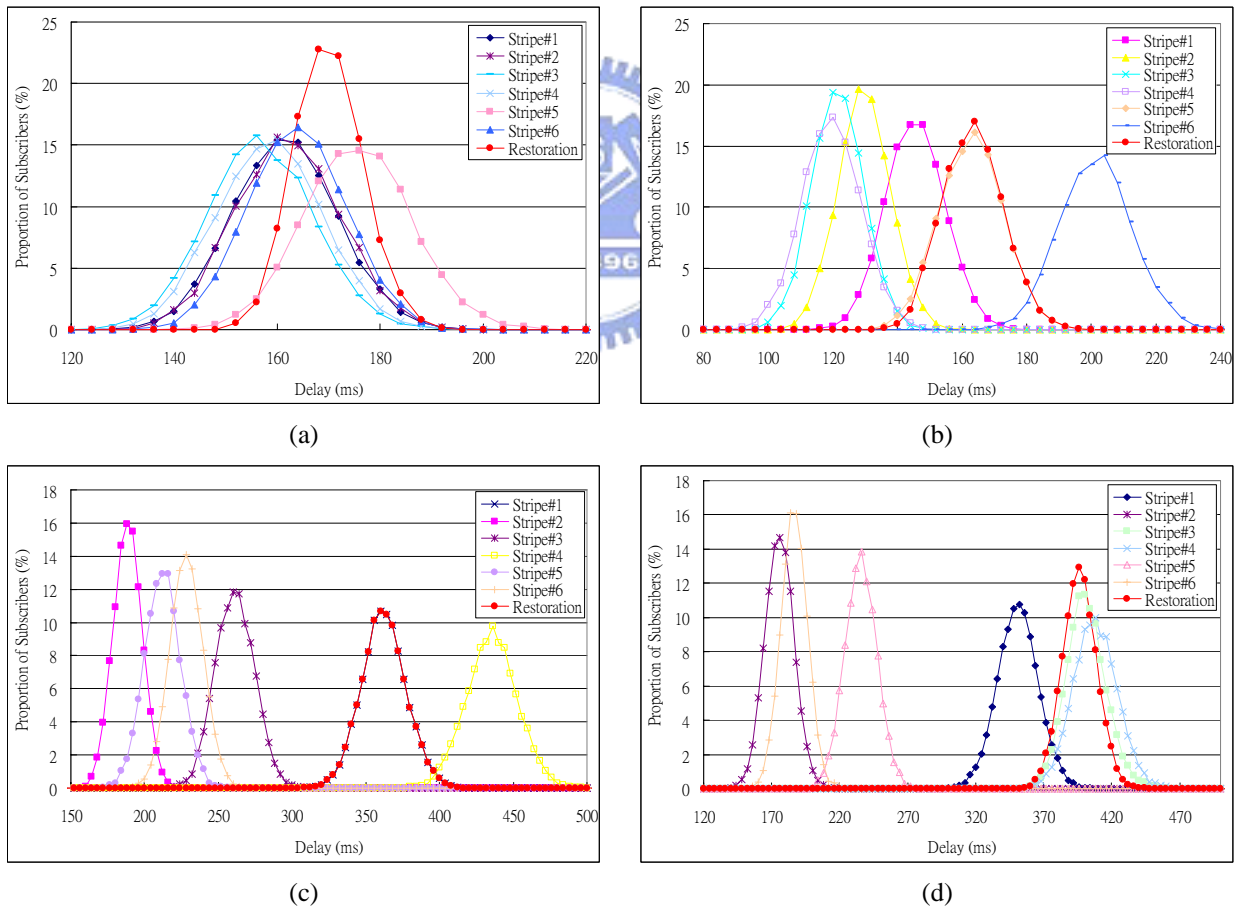


Figure 21: Histogram of stripe and restoration delay for different subscriber

The histograms are presented in Figure 21. (a) is the subscriber with the minimum max and stddev (standard deviation) of delay. (b) is the subscriber with minimum mean and min of

delay. (c) is the subscriber with maximum max and stddev of delay. (d) is the subscriber with maximum mean and min of delay.

Observation

- In Figure 21(a), it shows that when the curves of stripe delay overlap more intricately, the curve of message restoration delay will be much narrower.
- In Figure 21(b), it shows that stripe#6 almost didn't influence on the delay of message restoration. It is the advantage of receive more than m stripes.
- In Figure 21(c), the curve of message restoration delay almost overlaps the curve of stripe#1. It is because stripe#1 is nearly always the 16th stripe that arrives to the subscriber. We can see the overlap of the curve of stripe#1 and other curves is very few. Therefore, the delay of message restoration is almost fully influenced by the delay of stripe#1. That's why the subscriber's variance and stddev of delay is the maximum.
- We obtain a conclusion that the curve of message restoration is narrower than other curves of stripe delay. It is because we use IDA and multiple stripes/trees approaches, the delay of message restoration is based on delay of several stripes. Therefore, the delay of message restoration will be more stable.

5.4.1.2 Delay for Trickle

We have measured the end to end delay of message restoration for each subscriber. We use the statistics to calculate the average delay of all subscribers. It is presented in Table 4.

Max	Mean	Min	Variance	Stddev
314.91	244.90	207.64	123.36	10.81

Table 4: Average delay, variance, and stddev for all subscribers with Trickle

There are 5 arguments in the table. Max means during evaluation period, the average of maximum delay for all subscribers. Mean means the average of mean delay for all subscribers. Min means the average of minimum delay for all subscribers. Variance and standard deviation (stddev) mean how spread out the distribution of each message delay is.

The bar chart histogram of mean delay is presented in Figure 22. And the bar chart

histogram for coefficient of variation (standard deviation divided by mean delay) is presented in Figure 23.

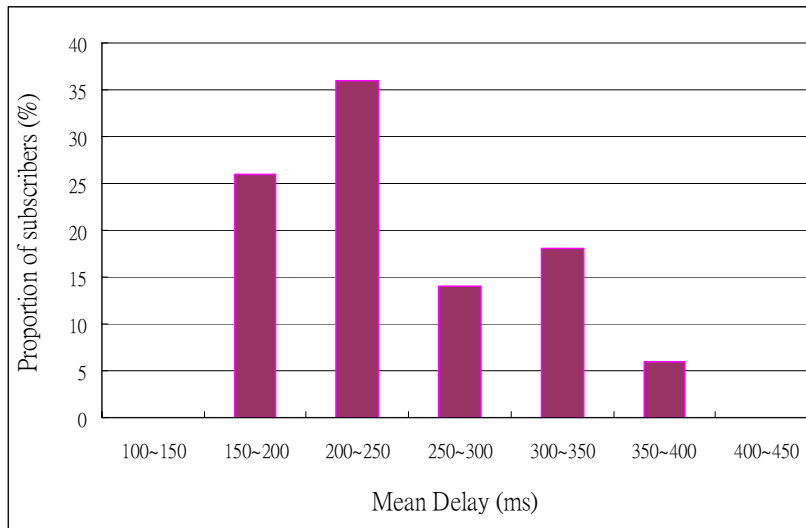


Figure 22: Histogram of mean delay for all subscribers

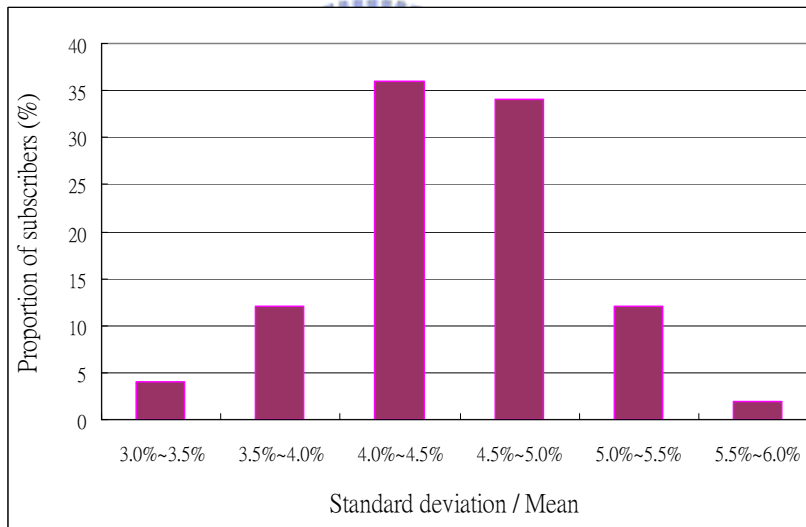


Figure 23: Histogram of the coefficient of variation for all subscribers

Moreover, we show the relationship between restoration delay and path length from source peer to subscribers in Figure 24. Length of each link is generated from GT-ITM. Each transmission path from source to subscriber will pass through many links. We sum up the length of these links to be path length. We choose the path length that is 16th shortest for each subscriber because it will possibly affect restoration delay most. In Figure 24, each straight line means the restoration delay of one subscriber. The top of straight line means the max

delay, the middle circle of straight line means the mean delay, and the bottom of straight line means the min delay for the subscriber.

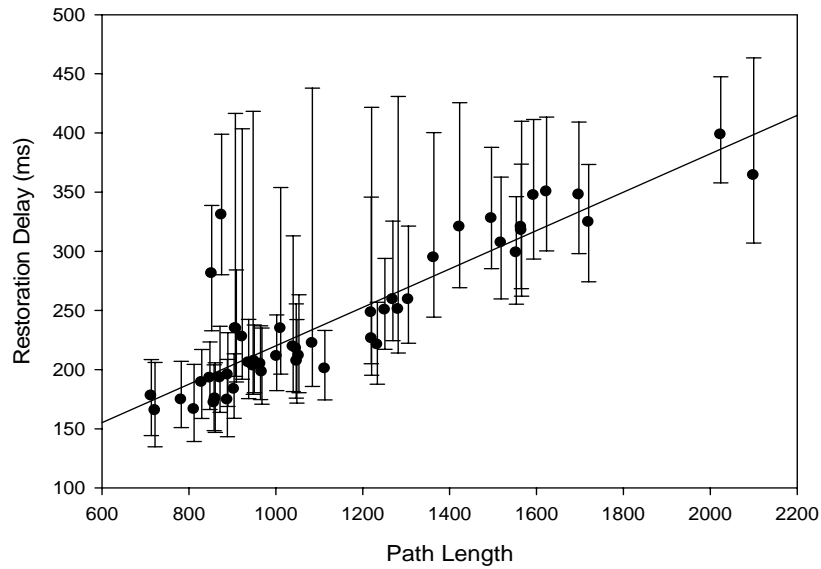


Figure 24: Restoration delay versus path length for each subscriber

Observation

- In Table 4, we can observe the difference between max delay, mean delay, and min delay is small. It is because we use IDA and multiple stripes approaches.
- In Figure 22, we can observe most mean delay of subscribers is within 150 to 350 ms. The standard deviation for mean delay of each subscriber is only 61 ms.
- In Figure 23, we can observe that the coefficient of variation for all subscribers is within 3% to 6%. It means the dispersion of mean delay is less and the delay of message restoration is stable.
- In Figure 24, it shows restoration delay and 16th shortest path length are in direct proportion. It also shows mean delay for each subscriber are close to regression line. Slope = 0.16, intercept = 57.80, and correlation coefficient = 0.78.
- We obtain a conclusion that the delay of message restoration for each subscriber is stable. And the difference of delay between subscribers is small.

5.4.1.3 Relative Delay Penalty (RDP)

In order to calculate RDP, we must measure the delay of message restoration by using IP multicast first. We run IP multicast in the same condition as Trickle. The result of IP multicast delay is presented in Table 5. The table also contains the same arguments as Table 4 for Trickle. Now, we can calculate RDP with these statistics.

Max	Mean	Min	Variance	Stddev
96.31	70.79	45.82	42.69	6.46

Table 5: Average delay, variance, and stddev for all subscribers with IP multicast

We calculate RMD, RAD, and also the relative minimum delay (RMND) of each subscriber. The statistics of RDP for each subscriber is attached in appendix. The average is presented in Table 6. And the histogram of subscriber's RDP is presented in Figure 25.

RMD	RAD	RMND
3.537	3.831	5.191

Table 6: Average relative delay penalty for all subscribers

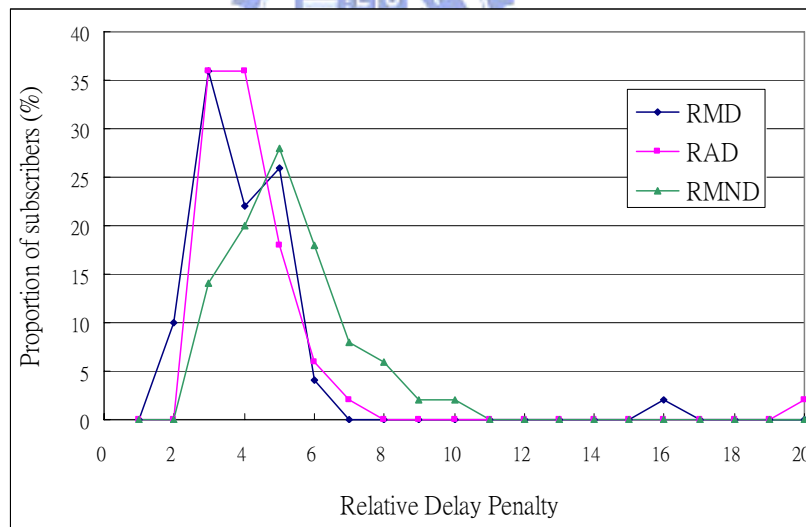


Figure 25: Histogram of relative delay penalty

Observation

- It shows that most RDP of subscribers is within 3 to 5. It is good that every subscriber has approximately the same delay penalty relative to IP multicast. (By using IP multicast, subscribers will have least delay of message restoration)
- There is a subscriber that its RDP is much bigger than other subscribers. It is because the

distance between the subscriber and source peer is only 2 hops. Hence, the delay of message restoration for the subscriber by using IP multicast is very small and the delay by using Trickle is similar to other subscribers. It causes the RDP much bigger than other subscribers.

5.4.2 Link Stress

The network resource usage of Trickle and IP multicast is presented in Table 7. There are three different values in the table. First, link count means the number of link being used in the approach. Second, bandwidth consumption means the average bandwidth of one link used the approach. Third, packet overhead means the ratio of the sum of packet header size and the total amount of network traffic.

	Link Count	Bandwidth Consumption (per link)	Packet Overhead
Trickle	1319	175.5 Kbps	33%
IP Multicast	176	312.5 Kbps	4%

Table 7: Link stress

Observation

- It shows the link count of Trickle is much bigger than the link count of IP multicast. It is because we use multiple stripes/trees approach.
- There are 20 trees ($n=20$) in Trickle. The average number of links used in each tree is about 66 ($1319/20$).
- Because we use multiple stripes/trees approach, bandwidth consumption of one link in Trickle is smaller than in IP multicast.
- In Trickle, it only consumes with an average of 175.5 Kbps per link and the maximum upstream bandwidth one peer can provide is 225 Kbps. For asymmetric connectivity whose upstream bandwidth is 512 Kbps, it only costs 34.2% ($175.5/512$) of total upstream bandwidth.
- The total bandwidth consumption can be calculated by the link count multiplies by bandwidth consumption. Trickle is $(1319 \times 175.5) / (176 \times 312.5) = 4.2$ times larger than

IP multicast

- Link stress is the average number of stripes of one frame pass through a link. The link stress in Trickle is $8232 / 1319 = 6.2$.
- Trickle has large packet overhead because of our poor IDA utilization.

5.4.3 Probability of Success Restoration

We run the simulation several rounds with different peer failure probability to make comparison. We got the statistics of the probability of success restoration for each subscriber in different cases of peer failure probability and it is attached in appendix. The average value is presented in Table 8. For easy observation, we plot the average values by a curve presented in Figure 26.

Peer Failure	1%	2%	3%	4%	5%	6%	7%
(20, 17, 16)	97.67	95.82	93.28	88.43	81.09	79.95	72.50
(16, 16, 16)	67.76	49.07	35.22	24.16	X		

Table 8: Average probability of success restoration

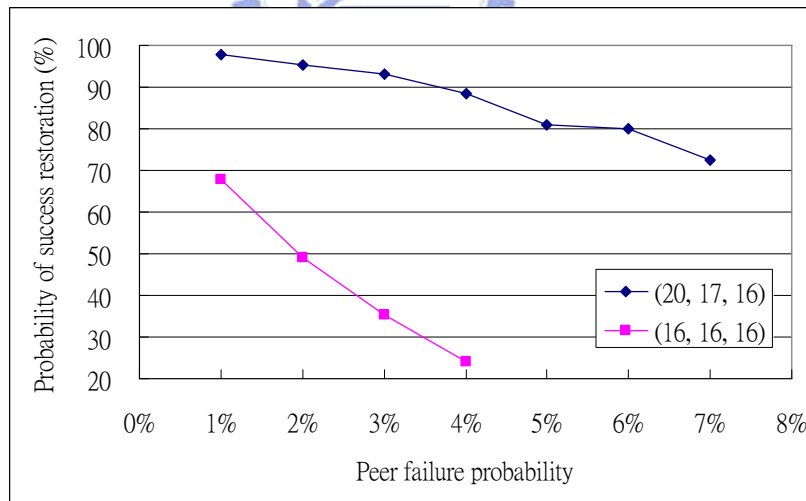


Figure 26: Probability of success restoration versus different peer failure probability

We also plot a histogram presented in Figure 27 to show the probability of success restoration versus different peer failure probability.

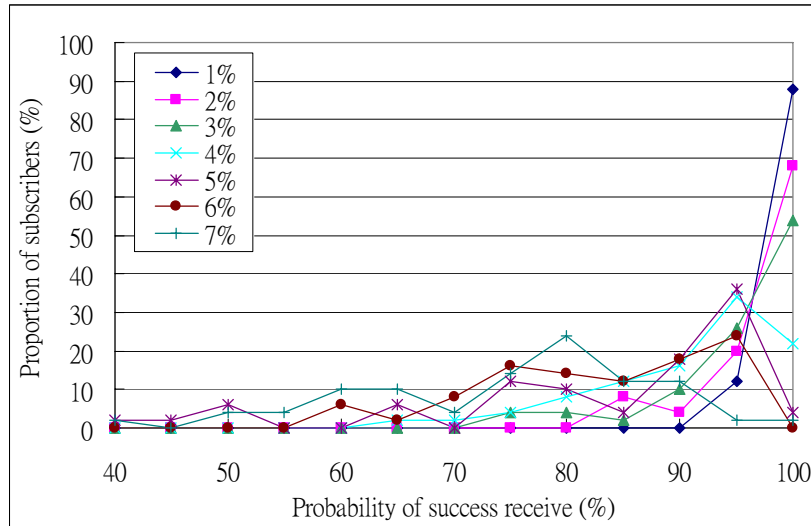


Figure 27: Histogram of the probability of success restoration

Observation

- In Table 8 and Figure 26, it shows the probability of success restoration is good (the average is more than 93%) when the peer failure probability is less or equal 3%. The probability of success restoration drops when the peer failure probability is more than 3%.
- In Figure 26, the probability of success restoration is much better when it uses IDA.
- In Figure 27, when the peer failure probability is less or equal 2%, all subscribers' probability of success receives are greater than 80%.
- In Figure 27, we can observe most subscribers' probability of success receive is greater than 60% even if peer failure probability is 7%.
- The result can give us a basis of setting the ratio of n and m . If we could predict the peer failure probability of the service, we can decide (n, o, m) more suitably to tolerate the peer failure and reduce the overhead of IDA.

Chapter 6

Conclusion

6.1 Accomplishment

There are two issues in ALM we focus on. First issue is peers that form the ALM service change dynamically. It means peers may join or leave the service at anytime. It will also cause links between peers to change dynamically. Therefore, transmission of streaming data is not reliable. The service can not promise all subscribers receive all packets sent from source peer. Second issue is narrow upstream bandwidth of peers will be the bottleneck of ALM service. When the bit rate of streaming data is higher, for peers with asymmetric connectivity, the number of children they can retransmit data to is fewer. It will cause the end to end delay to be higher because of larger tree depth and also limit the number of subscribers can subscribe the service.

The accomplishments we have done are:

- We propose the combination of three approaches to provide a resilient application layer multicast mechanism and solve the issue of peers with insufficient upstream bandwidth.
 - **Information dispersal algorithm** Subscribers can tolerate some packets lose without losing data completeness. And also IDA provides partial security protection to the streaming data.
 - **Multiple stripes/trees** With multiple small stripes, each peer can fully utilize their upstream bandwidth and also support more children. With multiple trees, when any links of one tree broken, subscribers can still receive other stripes in other trees.
 - **Helper** With helpers contributing their upstream bandwidth, the total amount of upstream bandwidth in the service will increase and make the service to accept

more subscribers. Furthermore, with the join of helper, subscribers will have more choices of tree parent to have lower end to end delay.

- We have implement Trickle mechanism with SplitStream tree building algorithm on the discrete event simulator by C++. Parts of implementation code are attached in appendix.
- We have run the simulation several rounds for three purposes. And we have three conclusions described below.
 - **Message delay** We obtain a conclusion that the delay of message restoration for each subscriber is stable (coefficient of variation is within 3% to 6%). And the difference of delay between subscribers is small (standard deviation is 61 ms).
 - **Network resource usage** The average bandwidth consumption of one link in Trickle is low. The average bandwidth consumption per link is less than half of upstream bandwidth for asymmetric connectivity (only cost 34.2% of total upstream bandwidth).
 - **Successful restoration rate** We have good tolerance for packet loss. From the experiment round with peer failure probability, it shows subscribers still have good successful probability of message restoration (the average is more than 93%) when peer failure probability is less or equal 3%.

6.2 Future Work

There are several works will be done or need to be resolved.

- Devise a tree building algorithm. Although SplitStream is a decentralized tree building method, it doesn't provide tree adaptive method to locally change links between peers based on some criteria after forming a tree. We want to provide an ALM tree building algorithm that can build tree by decentralized method and optimize tree structure by local information after tree built.

- Combine IDA with existent video coding technology. From the assumption of our simulation, it shows that after one frame divided into stripes, the overhead of packet header is very large. We need to solve it by developing a better fragmentation principle based on the combination of IDA and the video codec. It will help to save more network bandwidth.
- Usage of source proxy. There are still a lot of issues of using source proxy. Who is the source proxy, how to find a source proxy, and how many source proxies should one streaming data service use are all still need to be investigated.
- Simulation on real network. After we have done the works described before, they will be combined with Trickle. We will implement it and run the simulation on real network to measure its performance.



Reference

- [1] C. Diot, B. Levine, J. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the IP multicast service and architecture," in *IEEE Network*, Jan. 2000.
- [2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *SOSP*, Oct. 2003.
- [3] V. N. Padmanabhan, H. J. Wang, and P. A. Chou, "Resilient peer-to-peer streaming," in *Proc. of IEEE ICNP*, 2003.
- [4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc. of ACM SIGCOMM*, Aug. 2002.
- [5] D. Tran, K. Hua, and T. Do, "ZIGZAG: An efficient peer-to-peer scheme for media streaming," in *Proc. of IEEE INFOCOM*, Mar. 2003.
- [6] 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.
- [7] A Ganjam and H Zhang, "Internet multicast video delivery," in *Proc. of IEEE*, Jan. 2005.
- [8] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," in *JACM*, Apr. 1989.
- [9] Y. Chu, A. Ganjam, T. S. E. Ng, S. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early experience with an internet broadcast system based on overlay multiast," in *Proc. USENIX Ann. Tech. Conf.*, June 2004.
- [10] K. Calvert, E. Zegura, and S. Bhattacharjee, "How to model an Internetwork," in *Proc. of IEEE INFOCOM*, March 1996.
- [11] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient multicast using overlays," in *Proc. of ACM SIGMETRICS*, June 2003.
- [12] M. Yang and Z. Fei, "A proactive approach to reconstructing overlay multicast trees," in *Proc. of IEEE INFOCOM*, March 2004.
- [13] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM Int. Conf. Distributed Systems Platforms (Middleware)*, Nov. 2001.
- [14] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE JSAC*, vol. 20, no. 8, October 2002.
- [15] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-resilient wide-area location and routing," U. C. Berkeley, Tech. Rep. UCB//CSD-01-1141, Apr. 2001.
- [16] S. Zhuang, B. Zhao, J. Kubiatowicz, and A. Joseph, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proc. of NOSSDAV*, June 2001.
- [17] A. Varga et al. The OMNeT++ discrete event simulation system. <http://www.omnetpp.org>

- [18] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the selfsimilar nature of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, pp. 1–15, Feb. 1994.
- [19] Taiwan Network Information Center. <http://www.twnic.net.tw/>
- [20] Chunghwa Telecom. <http://www.cht.com.tw/>
- [21] L. Lao, J.-H. Cui, M. Gerla, and D. Maggiorini, "A comparative study of multicast protocols: Top, bottom, or in the middle?" in *Proc. of IEEE INFOCOM*, Mar. 2005.
- [22] V.K. Goyal, "Multiple description coding: Compression meets the network," *IEEE Signal Processing Magazine*, vol. 18, pp. 74-93, Sept. 2001.
- [23] A. Drigas, S. Kouremenos, Y. Bakopoulos, V. Loumos, "A study of H. 263 traffic modeling in multipoint videoconference sessions over IP networks," *Computer Communications*, vol. 29, issue 3, Feb, 2006.
- [24] VC-1, SMPTE. <http://www.microsoft.com/windows/windowsmedia/forpros/events/NAB2005/VC-1.aspx>
- [25] RealProducer, RealNetworks. <http://docs.real.com/docs/RealProducer10/Producer.pdf>



Appendix

A. Code of Implementation

We show some codes of functions used in the simulation. All these codes have been modified and simplified for easy understanding.

A.1 Join and Accept

```
void Host::send_join_req(int treenum) {
    do_parent_find(treenum);

    packetp=do_packet_create(index(),dest,NET_JOIN_REQ);
    send(packetp,"out");
}

void Host::receive_join_req(Net_packet *packetp) {
    treenum=packetp->getIntpar(0);

    if(inService==false) {
        send_join_accept(packetp);
        send_join_req(treenum);
    }
    else {
        if(isSource==true) {
            if(c_child_num<max_child_num) {
                if(hasChildInTree[treenum]==0)
                    send_join_accept(packetp);
                else if((n_IDA-hasChildInTreeNum) < (max_child_num-child_num))
                    send_join_accept(packetp);
                else
                    send_join_help(packetp);
            }
            else
                send_join_help(packetp);
        }
        else if(interiorNum==treenum) {
            if(child_num<max_child_num)
                send_join_accept(packetp);
            else
                send_join_help(packetp);
        }
        else
            do_join_req_forword(packetp);
    }
}

void Host::send_join_accept(Net_packet *packetp) {
    do_child_add(childid,treenum);
}
```



```

    newpacketp=do_packet_create(index(),childid,NET_JOIN_ACCEPT);
    send(newpacketp,"out");
}

void Host::receive_join_accept(Net_packet *packetp) {
    tree_num=packetp->getIntpar(0);

    parent[tree_num]=packetp->getSrc();

    if(tree_num==default_interiorNum && c_child_num==0)
        interiorNum=tree_num;
    else if(interiorNum==-1)
        interiorNum=tree_num;
}

```

A.2 Helper Approach

```

void Host::send_join_help(Net_packet *packetp) {
    helper=do_helper_find(childid,treenum);

    newpacketp=do_packet_create(index(),helper,NET_JOIN_HELP);
    send(newpacketp,"out");
}

void Host::receive_join_help(Net_packet *packetp) {
    treenum=packetp->getIntpar(0);

    if(child_num>=max_child_num)
        send_join_help_fail(packetp);
    else
    {
        if(inService==false)
            send_join_help_accept(packetp);
        else {
            if(close_to_new_parent()==true) {
                send_leave_notify(parent[treenum],treenum);
                change_parent(packetp);
            }
            send_join_help_accept(packetp);
        }

        packetp->setSrc(childid);
        send_join_accept(packetp);
    }
}

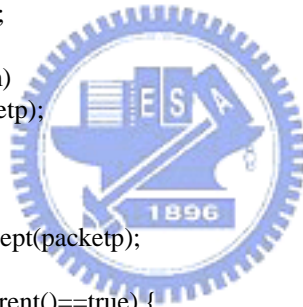
void Host::send_join_help_accept(Net_packet *packetp) {
    dest=packetp->getSrc();

    if(isChangeParent==true) {
        interiorNum=treenum;
        parent[treenum]=dest;
    }

    newpacketp=do_packet_create(index(),dest,NET_JOIN_HELP_ACCEPT);
    send(newpacketp,"out");
}

void Host::receive_join_help_accept(Net_packet *packetp) {

```



```

        if(isChangeParent==true) {
            do_child_add(childid,treenum);
            send_child_transfer(packetp);
        }
    }

void Host::receive_join_help_fail(Net_packet *packetp) {
    send_join_help(packetp);
}

void Host::send_child_transfer(Net_packet *packetp) {
    childid=do_transfer_child_find(treenum,helper);
    do_child_remove(childid,treenum);

    newpacketp=do_packet_create(index(),helper,NET_CHILD_TRANSFER);
    send((cMessage *)newpacketp->dup(),"out");
    newpacketp->setDest(childid);
    send(newpacketp,"out");
}

void Host::receive_child_transfer(Net_packet *packetp) {
    if(isHelper==true)
        do_child_add(childid,treenum);
    else {
        if(inService==false || inTree[treenum]==false)
            send_leave_notify_p(parent[treenum],treenum);
        else
            parent[treenum]=helper;
    }
}

```

A.3 Receive and Retransmit

```

void Host::receive_stripe_trans(Net_packet *packetp) {
    if(isSubscriber==false || (packetp=do_stripe_insert(packetp))!=NULL) {
        if(bit_error()==false && peer_failure()==false)
            send_stripe_forward(packetp);
    }
}

void Host::send_stripe_forward(Net_packet *packetp) {
    childp=child_table.begin();
    for(i=0;i<child_table.size();i++) {
        packetp->setDest(childp->id);
        send((cMessage *)packetp->dup(),"out");
        childp++;
    }
}

Net_packet *Host::do_stripe_insert(Net_packet *packetp) {
    seq_diff=stripe_seq-current_seq;
    if(seq_diff<0) //the stripe is out of date
        return NULL;

    newptr=(stripeptr+seq_diff)%stripe_buffer_size;
    if(seq_diff<stripe_buffer_size) {
        if(stripe_table[newptr][stripenum]==true)//already have the stripe
            return NULL;
    }
}

```

```

do_stripe_buffer_insert(newptr,stripenum,stripe);

//stripe regeneration
if(stripe_sum[newptr]>=m_IDA && stripe_table[newptr][interiorNum]==false) {
    do_ida_regen(newptr,interiorNum);
    packetp->setStrpar(stripe_buffer[newptr][interiorNum]);
    packetp->setIntpar(0,interiorNum);
}

while(stripe_sum[stripeptr]>=m_IDA)
    do_stripe_pop(stripeptr++);
}
else { //some stripes in buffer is out of date
    pop_num=min(seq_diff-stripe_buffer_size+1,stripe_buffer_size);
    for(i=0;i<pop_num;i++)
        do_stripe_pop(stripeptr++);

    do_stripe_buffer_insert(newptr,stripenum,stripe);
}
if(packetp->getIntpar(0)==interiorNum)
    return packetp;
else
    return NULL;
}

void Host::do_stripe_pop(int p) {
    if(stripe_sum[p]>=m_IDA)
        do_ida_restore();
    do_stripe_table_column_clean(p);
}

```



B. Statistics of Simulation

B.1 Statistics for Relative Delay Penalty

No.	ID	RMD	RAD	RMND
1	28	2.807	2.975	3.640
2	52	1.961	2.176	2.787
3	78	3.979	4.557	5.941
4	102	2.236	2.517	3.157
5	120	4.983	6.067	9.053
6	125	3.130	3.953	5.902
7	131	2.822	3.137	4.292
8	161	2.168	2.549	3.479
9	164	5.631	3.741	5.266
10	193	3.224	2.388	2.972
11	194	4.063	3.839	5.166
12	213	2.544	2.988	4.346
13	217	2.650	3.378	4.875
14	224	2.162	2.561	3.288
15	287	2.999	2.852	3.463
16	294	5.457	5.916	8.042

17	308	3.167	3.367	4.273
18	311	4.031	4.949	6.920
19	369	2.164	2.367	2.902
20	376	4.588	3.562	4.694
21	413	1.958	2.280	3.400
22	432	4.101	4.644	5.794
23	460	2.426	2.771	3.738
24	469	4.584	5.395	7.613
25	487	3.281	3.275	4.063
26	491	4.464	4.920	6.125
27	519	3.360	4.201	5.920
28	535	4.427	3.185	4.134
29	551	2.738	3.268	4.321
30	578	1.856	2.023	2.480
31	631	2.183	2.602	3.315
32	652	2.704	3.067	3.987
33	659	2.089	2.380	2.861
34	671	2.743	3.482	4.843
35	707	2.633	2.926	4.131
36	722	1.835	2.059	2.471
37	743	4.076	4.947	6.766
38	748	1.866	2.094	2.539
39	794	3.761	4.332	5.427
40	807	3.984	4.670	6.202
41	809	4.061	3.152	4.012
42	810	4.147	4.946	7.602
43	828	4.882	3.327	4.491
44	852	15.039	19.389	28.714
45	871	3.771	3.875	5.179
46	874	3.770	3.624	5.477
47	916	3.841	3.828	4.921
48	944	2.686	3.110	4.089
49	1001	4.576	5.418	7.393
50	1008	2.237	2.545	3.082

B.2 Statistics for Probability of Success Restoration

No.	ID	Peer Failure Probability						
		1%	2%	3%	4%	5%	6%	7%
1	28	96.618	92.887	94.943	83.861	60.356	55.619	69.265
2	52	95.465	97.447	94.406	94.358	94.500	91.776	88.229
3	78	94.988	94.971	98.642	96.575	75.763	81.348	78.740
4	102	98.368	90.115	93.970	84.519	88.905	79.568	45.256
5	120	95.213	96.440	78.882	76.446	70.933	70.663	84.220
6	125	98.147	95.099	97.907	74.155	87.071	83.444	78.751
7	131	98.442	98.660	98.758	95.734	86.071	74.750	70.247
8	161	98.943	84.558	96.687	80.945	92.790	86.258	58.265
9	164	99.717	97.090	95.689	78.072	92.729	72.577	62.053
10	193	97.993	99.257	99.032	93.190	63.414	73.008	58.105
11	194	97.637	98.083	89.556	92.892	48.174	76.773	72.989

12	213	97.994	97.296	96.300	90.709	85.964	75.734	49.638
13	217	95.495	96.289	93.805	70.813	89.140	77.693	58.503
14	224	95.443	95.908	94.437	93.136	91.046	82.809	78.475
15	287	98.767	94.578	93.813	96.068	74.648	82.694	73.038
16	294	96.368	98.341	91.868	90.906	73.399	55.084	37.632
17	308	99.703	96.704	91.174	61.509	61.604	67.247	56.555
18	311	99.344	98.521	97.171	83.910	39.922	69.822	72.017
19	369	94.029	85.424	95.056	94.132	94.838	87.537	83.035
20	376	98.263	99.157	89.193	94.712	81.806	91.804	62.319
21	413	99.718	83.089	92.012	89.595	93.535	85.627	71.840
22	432	94.208	97.885	77.883	86.239	74.907	66.291	78.194
23	460	99.623	99.073	91.886	94.408	94.441	86.712	78.204
24	469	96.971	99.478	88.862	90.843	46.445	55.041	60.267
25	487	99.871	97.328	73.285	86.619	79.918	84.014	78.679
26	491	99.580	99.169	97.377	85.291	73.798	79.406	62.835
27	519	99.852	97.660	84.803	87.690	89.998	85.249	90.904
28	535	93.743	83.210	89.347	79.793	87.104	73.476	50.009
29	551	97.978	98.546	96.613	96.968	91.512	89.056	77.197
30	578	94.647	92.262	96.271	95.758	94.459	92.276	86.253
31	631	99.425	95.506	97.705	92.620	92.964	85.587	77.077
32	652	95.843	92.876	97.814	94.832	96.060	92.451	87.917
33	659	97.889	99.275	97.247	95.681	95.633	92.047	89.146
34	671	98.125	95.865	93.017	90.961	92.406	91.804	84.645
35	707	98.670	98.428	97.833	92.055	93.249	90.475	82.480
36	722	93.359	95.381	98.312	94.230	86.111	83.271	77.083
37	743	97.497	99.268	96.814	97.347	92.433	85.847	77.545
38	748	96.859	92.598	96.922	96.602	91.874	90.078	86.140
39	794	95.670	97.522	87.929	85.879	77.146	93.515	84.385
40	807	99.849	95.886	91.247	96.699	76.055	72.071	73.088
41	809	98.597	80.544	95.231	67.934	74.148	67.334	71.577
42	810	99.705	98.729	90.746	82.310	92.045	75.144	65.784
43	828	98.338	96.281	95.329	78.390	41.714	60.233	54.781
44	852	99.926	93.130	97.635	94.530	90.390	91.728	97.138
45	871	99.870	92.817	73.386	93.903	92.261	74.283	77.249
46	874	100.000	99.832	97.723	96.525	49.556	85.246	62.817
47	916	99.017	88.531	96.647	81.325	79.926	71.113	57.444
48	944	97.710	96.550	97.516	88.585	80.426	92.640	85.505
49	1001	98.846	99.741	99.005	85.207	86.755	77.889	82.694
50	1008	95.082	94.668	96.181	96.238	94.262	91.562	78.927

B.3 Distance between Source and Subscribers

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Distance	13	12	12	15	8	8	10	10	8	15	9	10	9	11	13	8	12	12	13	10	12	11	10	9	14

No.	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
Distance	10	8	11	10	15	11	9	13	8	9	15	9	15	12	11	12	7	10	2	14	9	13	8	9	13

B.4 Distance between Each Subscribers

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50					
	10	11	13	13	11	10	10	9	13	12	9	12	10	11	13	3	10	11	12	10	14	15	12	17	8	10	9	13	13	10	12	11	11	12	13	10	12	11	11	12	13	12	15	15	6	9	12	9	13	12	12	11	14	11
		10	11	12	10	6	8	5	11	8	11	9	9	12	9	8	7	8	6	13	14	9	16	8	10	7	10	5	9	10	9	11	10	11	13	11	15	14	9	10	11	11	11	11	11	11	10	13	5					
			13	12	10	10	10	8	13	11	8	11	9	11	12	10	10	11	12	10	13	14	11	16	8	10	9	12	13	9	11	10	11	13	11	15	14	9	10	11	8	12	6	11	7	10	13	11						
				15	13	11	11	14	14	11	14	12	10	15	12	11	12	13	10	12	11	16	17	14	19	11	13	10	15	14	12	13	14	2	14	16	17	11	12	14	11	15	13	14	12	13	16	12						
					8	10	10	8	15	9	10	9	11	13	8	12	12	13	10	12	11	10	9	14	10	8	11	10	15	11	9	13	8	9	15	12	11	12	7	10	8	14	9	13	8	9	13							
						8	8	6	13	7	8	7	9	11	8	10	10	11	8	10	9	10	7	14	8	8	9	13	9	7	11	6	7	13	9	13	10	9	10	7	8	8	10	8	11	7	10	11	7					
							2	8	9	9	8	9	9	10	9	8	7	6	6	11	12	7	16	8	10	7	8	8	9	7	7	8	7	11	11	9	10	8	9	9	8	10	8	11	7	10	11	7						
								8	9	9	8	9	9	9	10	8	8	7	6	11	12	7	16	8	10	7	8	8	9	7	7	8	7	11	11	9	10	8	9	9	8	10	8	11	7	10	11	7						
									8	9	9	8	9	9	10	8	8	7	6	11	12	7	16	8	10	7	8	8	9	7	7	8	7	11	11	9	10	8	9	9	8	10	8	11	7	10	11	7						
										11	7	6	7	7	9	8	8	9	8	9	10	7	12	6	6	7	8	11	7	7	9	6	7	11	13	10	7	8	7	6	8	10	7	6	8	10	7	9	6	9				
											14	11	14	12	15	12	11	10	11	9	16	17	12	19	11	13	10	13	2	12	12	8	13	12	14	14	15	11	12	14	11	15	9	14	10	13	16	8						
												9	8	10	12	9	11	11	12	9	11	10	11	8	15	9	10	9	14	10	8	12	5	8	14	10	14	11	10	11	8	9	9	13	10	12	9	10	12					
													9	7	9	10	8	8	9	10	8	11	12	9	14	6	8	7	10	11	7	9	8	9	11	13	12	7	8	9	6	10	9	9	8	11	9							
														10	12	9	11	11	12	9	11	10	11	8	15	9	10	9	14	10	8	12	7	8	14	10	14	11	10	11	8	9	9	13	10	12	9	10	12					
															10	11	9	9	10	11	9	12	13	10	15	5	9	8	11	12	2	10	10	9	12	10	12	14	15	9	10	12	9	13	11	12	10	11	14	10				
																13	10	9	10	11	9	14	15	12	17	9	11	8	13	12	10	12	10	11	12	10	12	14	15	9	10	12	9	13	11	12	10	11	14	10				
																	12	12	13	10	12	11	9	9	14	10	8	11	10	15	11	9	13	8	9	15	12	11	12	6	10	8	14	9	13	8	7	13						
																		9	10	11	9	13	14	11	16	8	10	8	12	12	9	11	10	10	11	12	11	14	14	5	8	11	8	12	11	11	10	13	10					
																			9	10	8	13	14	11	16	8	10	7	12	11	9	11	10	11	11	13	14	8	9	11	8	12	10	11	9	10	13	9						
																				9	10	15	10	17	9	11	8	11	10	10	8	11	10	10	8	11	10	12	13	9	10	12	9	13	9	12	2	11	14	8				
																					8	11	12	7	16	10	10	9	8	11	11	7	9	8	7	13	11	11	12	8	9	11	8	12	8	11	7	10	13	7				
																						13	14	9	16	8	10	7	10	9	9	7	10	9	11	11	12	8	9	11	8	12	8	11	7	10	13	7						
																							13	10	16	11	10	12	11	16	12	8	14	9	10	16	11	16	11	12	13	10	11	11	15	11	14	10	12	14				
																								11	16	12	10	13	12	17	13	11	15	10	11	17	14	13	14	5	12	10	16	11	15	10	10	15						
																									15	9	9	10	7	12	10	6	10	7	6	14	10	12	9	10	11	8	9	9	11	10	10	9	10	10				
																										14	10	15	16	19	15	13	17	14	15	19	13	21	16	15	16	13	14	14	18	13	17	8	15	17	9			
																											8	7	10	11	5	9	9	8	9	11	9	13	12	7	8	9	2	10	10	9	9	8	11	9				
																												9	10	13	9	7	11	8	9	13	7	15	10	9	10	7	8	8	12	7	11	4	9	11				
																													11	10	8	10	8	9	10	10	12	13	7	8	10	7	11	9	10	8	9	12	8					
																														13	11	7	11	8	7	15	11	13	10	11	12	9	10	10	12	11	11	10	11	11				
																															12	12	8	13	12	14	14	15	11	12	14	11	15	9	14	10	13	16	8					
																																10	10	9	10	12	10	14	13	8	9	10	5	11	11	10	9	12	10					
																																10	7	6	14	8	12	7	10	11	8	9	9	11	8	10	7	10	10					
																																	11	10	12	12	13	9	10	12	9	13	7	12	8	11	14	6						
																																	7	13	9	13	10	9	10	7	8	8	12	9	11	8	9	11						
																																	14	10	12	9	10	11	8	9	9	11	10	10	9	10	10							
																																		14	16	17	11	12	14	11	15	13	14	12	13	16	12							
																																		16	11	10	11	8	9	9	13	2	12	7	10	12								
																																		15	13	14	14	13	15	13	16	12	15	16	12									
																																		13	14	11	12	14	11	13	10	13	13											
																																		7	10	7	11	10	9	9</														