

國立交通大學

多媒體工程研究所

碩士論文

點對點影音串流系統的開發與研究



Study and Implementation of Reliable Peer-to-Peer Streaming System

研究生：劉家銘

指導教授：蕭旭峰 教授

中華民國 一〇一 年 二 月

點對點影音串流系統的開發與研究

Study and Implementation of Reliable Peer-to-Peer Streaming System

研究生：劉家銘

Student : Chia-Ming Liou

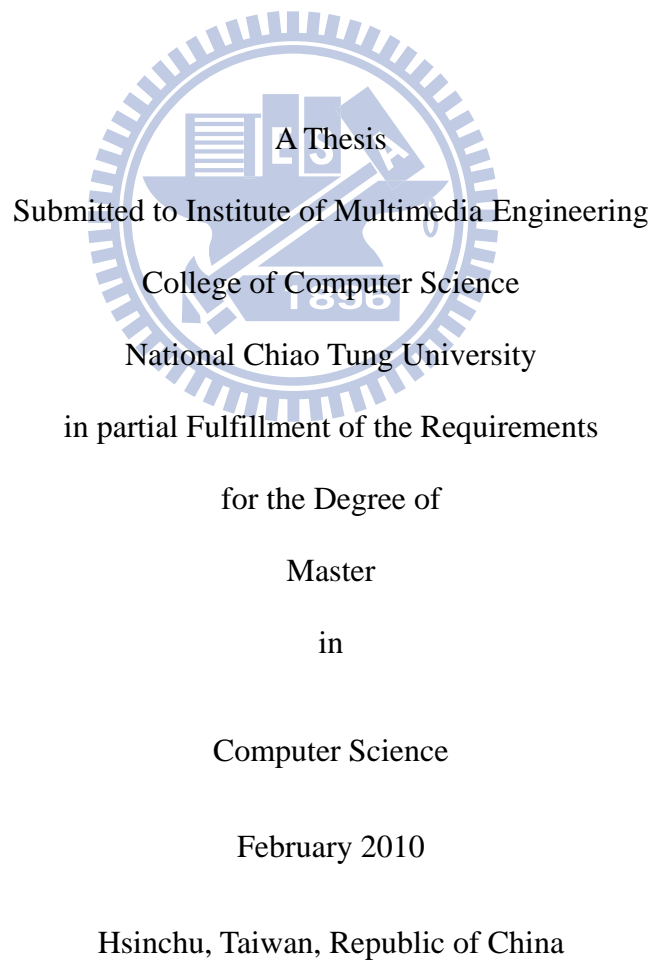
指導教授：蕭旭峰

Advisor : Hsu-Feng Hsiao

國立交通大學

多媒體工程研究所

碩士論文



中華民國一百年二月

點對點影音串流系統的開發與研究

研究生: 劉家銘

指導教授: 蕭旭峯

國立交通大學多媒體工程研究所

摘要

目前有越來越多影音服務架構在 peer-to-peer(P2P)系統中。然而，建立一個即時點對點影音串流系統充滿了許多挑戰。在本論文中，我們企圖建立一個穩定的點對點影音串流系統。首先，我們選擇了網狀結構當作網路拓撲的結構。因為有了網狀結構為基礎，使用者的高流動率對系統的影響較小；接下來，我們動態地選擇適合自己的父節點，挑選的標準由四個參數決定。此外在編碼方面，從原始資料依序進行影音編碼和信道編碼。藉由影音編碼，我們節省了全部要發送的總資料量；而經由信道編碼，讓我們節省了同儕之間資料溝通的訊息並加強了資料的保護。在我們實作系統的過程中，我們發現了一些實作上的議題，藉由這些議題，使我們的即時串流系統更加穩定。最後，在 NS2 模擬也顯示了我們方法的穩定性。


Study and implementation of reliable peer-to-peer streaming system

Student: Chia-Ming Liou

Advisor: Hsu-Feng Hsiao

Institute of Multimedia Engineering
National Chiao Tung University

Abstracts



Nowadays, there are more and more video streaming services based on peer-to-peer architecture. There are many challenges to build a real time peer-to-peer streaming system. In this paper, we attempt to construct a stable peer-to-peer streaming system. At the beginning, we choose the mesh-based structure to construct the peer-to-peer overlay network. The whole system is more robust when dealing with peer churn because of the mesh-based structure. Next, we dynamically choose the parents that are suitable to us by four factors. Furthermore, the original data from source is encoded by video coding and channel coding sequentially. With video coding, we save the total size of data to send. With channel coding, we save a lot of data coordinate messages for peers and enhance the protection of data. During the process of building our system, we find some important implementation issues to make our peer-to-peer streaming system more stable. Finally, the NS2 simulation also shows the stability of the method we proposed.

Acknowledgement

能夠完成這篇論文，首先必須要感謝蕭老師的指導。在無數次的挫折中，老師總是循序漸進地帶領我慢慢找出問題的癥結，提醒應該要注意的方向。老師也教導了我許多人生中正確的態度，讓我不僅在學業上有所突破，對於人生的方向也更能掌握。同時也要感謝實驗室的好伙伴們：聖舜學長、智為學長、新嘉學長、玉書學長、伯恆、峻豪、群志、忠穎、彥暉、旻璟、晨洲、德沛以及昀修，不僅讓我在研究上有共同討論的對象，你們在我研究和生活遇到瓶頸的時候，也能適時地給予支持與鼓勵。此外，要特別感謝我的家人，在我快要支持不下去的時候，拉了我一把並給予莫大的支持。最後，謹將這篇論文獻給我最摯愛的母親以及家人。



Contents

摘要.....	I
ABSTRACTS.....	II
ACKNOWLEDGEMENT	III
CONTENTS.....	IV
CHAPTER 1. INTRODUCTION.....	- 1 -
1.1 PREFACE.....	- 1 -
1.2 MOTIVATION	- 1 -
1.3 SYSTEM OPERATING ENVIRONMENT	- 2 -
1.4 THESIS ORGANIZATION	- 3 -
CHAPTER 2. RELATED WORK.....	- 4 -
2.1 PREFACE.....	- 4 -
2.2 PEER-TO-PEER STREAMING SYSTEM.....	- 4 -
2.3 PEER-TO-PEER OVERLAY NETWORK	- 5 -
2.4 FORWARD ERROR CORRECTION CODES.....	- 6 -
2.5 DATA PROTOCOL AND PEER MANAGEMENT	- 7 -
CHAPTER 3. PROPOSED METHOD.....	- 9 -
3.1 SYSTEM ARCHITECTURE	- 9 -
3.1.1 Video Compression Codec.....	- 11 -
3.1.2 Channel Coding.....	- 12 -
3.2 CONNECTING PROTOCOL.....	- 15 -
3.3 PEER MANAGEMENT	- 17 -
3.4 DATA PROTOCOL	- 20 -
3.4.1 Adaptive index.....	- 22 -
3.4.2 How to allocate the bandwidth of children and parents	- 24 -
3.4.3 Communication Protocol	- 25 -
3.4.4 Packet Protection.....	- 26 -
CHAPTER 4. SIMULATION RESULT	- 28 -
4.1 SIMULATION ENVIRONMENT	- 28 -
4.2 WEIGHT SETTING	- 29 -
4.3 THE ENVIRONMENT UNDER BACKGROUND TRAFFIC.....	- 31 -
4.4 THE REAL-WORLD IMPLEMENTATION	- 36 -
CHAPTER 5. CONCLUSIONS	- 41 -

5.1 CONCLUSIONS - 41 -

REFERENCE - 42 -



List of Tables

Table 4-1 The parameter setting..... - 29 -
Table 4-2 The parameter setting..... - 38 -



List of Figures

<i>Figure 1-1: The upper left: Sopcast [18] , the upper right PPStream [19] and TVUPlayer [20]</i>	- 2 -
<i>Figure 2-1 Reed-Solomon code [15]</i>	- 7 -
<i>Figure 3-1 System Architecture</i>	- 9 -
<i>Figure 3-2 The component of peer</i>	- 11 -
<i>Figure 3-3 The process of video compression</i>	- 11 -
<i>Figure 3-4 LT codes Encoder</i>	- 13 -
<i>Figure 3-5 The decoding process of LT codes</i>	- 14 -
<i>Figure 3-6 The Complete LT Encoding process of Root</i>	- 15 -
<i>Figure 3-7 The Connecting Protocol</i>	- 16 -
<i>Figure 3-8 The Format of Communication Packet</i>	- 17 -
<i>Figure 3-9 The peer pool</i>	- 17 -
<i>Figure 3-10 The data protocol</i>	- 21 -
<i>Figure 3-11 The condition of the sending buffer</i>	- 23 -
<i>Figure 3-12 The process of allocating children's bandwidth</i>	- 24 -
<i>Figure 3-13 The process of passing the NAT environment</i>	- 25 -
<i>Figure 3-14 Data protection for regular communication packets</i>	- 26 -
<i>Figure 3-15 Data protection for important communication packets</i>	- 27 -
<i>Figure 4-1 The network topology</i>	- 29 -
<i>Figure 4-2 The searching process of weights</i>	- 31 -
<i>Figure 4-3 The average jitter under background traffic</i>	- 32 -
<i>Figure 4-4 The continuous playback index under background traffic</i>	- 33 -
<i>Figure 4-5 The changing level of parent under background traffic</i>	- 34 -
<i>Figure 4-6 The effective throughput under background traffic</i>	- 34 -
<i>Figure 4-7 The source delay under background traffic</i>	- 35 -
<i>Figure 4-8 The ratio of the parents with background traffic</i>	- 36 -
<i>Figure 4-9 The system interface description</i>	- 37 -
<i>Figure 4-10 The network topology in real world</i>	- 37 -
<i>Figure 4-11 The average throughput</i>	- 39 -
<i>Figure 4-12 The average source delay</i>	- 39 -
<i>Figure 4-13 The PSNR for each decoded frame</i>	- 40 -

Chapter 1. Introduction

1.1 Preface

In the recent years, multimedia video streaming is considered to be one of the killer applications. The quality and quantity of the applications also increase day by day. The simplest way to do video streaming service is to make the source unicast the video streaming to each receiver. But since the video streaming services are basically high bandwidth consuming applications, this way will consume a tremendous amount of bandwidth and encounter a bottleneck at source. So its scale has a limitation. IP multicast could be the best way to overcome this drawback because it can serve many receivers from a single stream. However, it is still not widely deployed on the internet. On second thoughts, the peer-to-peer architecture provides a more feasible solution.

1.2 Motivation

Nowadays, there are more and more video streaming services [Figure 1.1] [18] [19] [20] based on peer-to-peer architecture. However, there are still a lot of topics need to be discussed in peer-to-peer architecture. For instance, a peer's behavior is unpredictable. They may leave or join at any time. Besides, the network condition between adjacent two peers may change with time. These drawbacks probably make the whole peer-to-peer video streaming system unstable.

In order to provide more stable video streaming system, we proceed with three aspects (video

streaming encoding, network topology construction and peer dynamic management) to build our system.



Figure 1-1: The upper left: Sopcast [18] , the upper right PPStream [19] and TVUPlayer [20]

1.3 System Operating Environment

[Figure 1.3] shows the environment our system operates in. It is a mesh based peer-to-peer network. There is one root and a large number of servers in the system. Furthermore, we need a patch server to record every peer in this system. The source attempts to spread the video streaming data to each server in the system. Similarly, each server wants to receive the newest video streaming data steadily. A server stands for a peer in the peer-to-peer network. It means that a server has to receive data from servers and upload data to servers simultaneously. Servers provide their available bandwidth to the system. The more servers are in the system, the more stable service the servers will have. [Figure 1.2] shows the screenshot of our system.

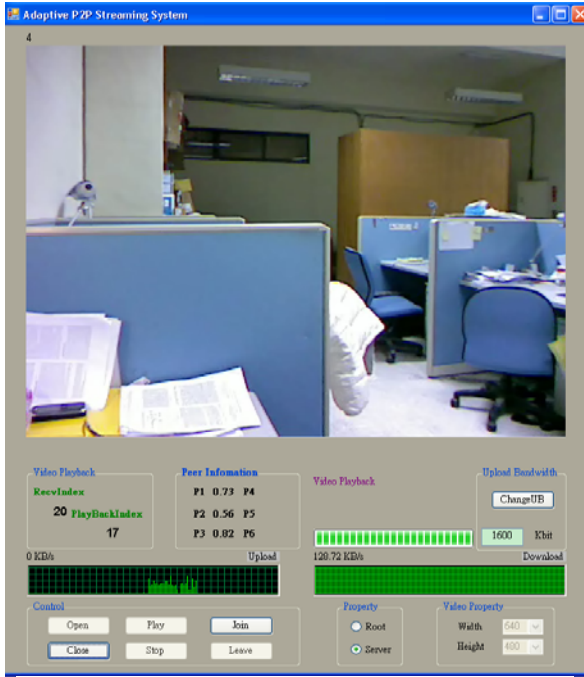


Figure 1-2 Screenshot of the system

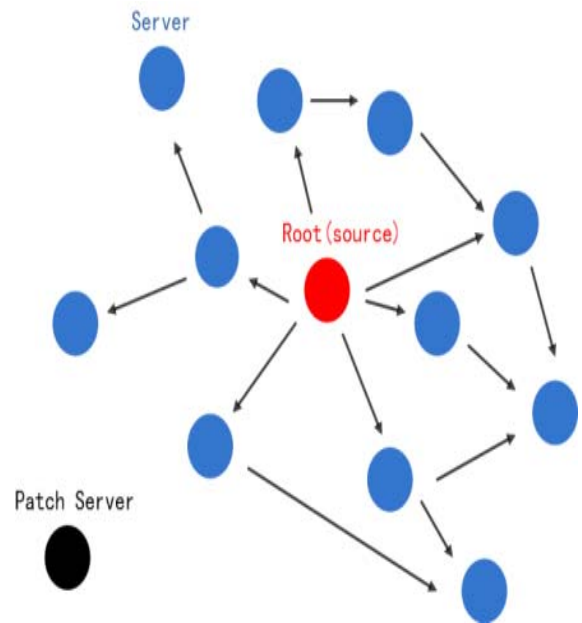


Figure 1-3: System operating environment

1.4 Thesis Organization

In the remainder of this paper, we review the related works in Chapter 2. The peer-to-peer streaming architecture and the practical implementation issues are presented in Chapter 3. In Chapter 4, the simulated experiment results are presented. Finally, the conclusions are given in Chapter 5.

Chapter 2. Related Work

2.1 Preface

At beginning, we would like to study a few cases about peer-to-peer live streaming. That will help us to build our system.

2.2 Peer-to-Peer Streaming System

The AnySee [17] is a tree-based peer-to-peer live streaming system. It uses a multiple tree protocol and lets each peer join multiple overlays. Each peer in the system maintains one active path and one backup path, in case the active path fails to deliver. Besides, it is the first to introduce the concept of inter-overlay optimization. In other words, it uses the remaining bandwidth of the peers that are watching channel A to help the peers that are watching channel B [26].

Prime [3] is a receiver-driven mesh-based streaming system and its streaming data is encoded with multiple description coding (MDC). The peers in Prime receiving more descriptions have better video quality. The parents push the information about their available data to their children periodically. On the other hand, every child requests data by pulling it from parents. PPLive [23] is the most popular video streaming system up to now. It is built by the proprietary company. By the work from [23] [24], we know it is a mesh-based peer-to-peer live streaming system and its parent selection mechanism is to pick the peers with high upload bandwidth.

At next part, in order to create a stable peer-to-peer streaming system, we have to choose which kind of peer-to-peer overlay network that is feasible for our system. And then we need to pick the appropriate peers to be our parents for enjoying stable video streaming. Besides, since each peer's status in peer-to-peer system is unstable, we want to enhance the protection of data streaming.

2.3 Peer-to-Peer Overlay Network

The Peer-to-peer overlay network can be generally classified to two classes: centralized and decentralized [21].

The centralized peer-to-peer network means that each peer in the system is assigned to a fixed connection with some peers by a central server. The central server organizes all peers into a predetermined graph like a single tree or multiple trees. The architecture of single tree [1] [6] is simple and easy to be deployed. The peer just follows the order from central server and starts to receive data from parents. However, peers differ in survival time, bandwidth and CPU power. [2] [5] try to divide peers into several groups by exploiting the heterogeneity of peers. And then each group will pick up a leader. All group leaders are formed by a single tree based structure. The centralized architecture has low management overhead and is suitable for multicast scheme. Nevertheless, it is vulnerable to disrupt under high rate of peer churn. Because each peer has one parent in a single tree, as long as the peer leaves suddenly, the peers at its downstream might suffer low downloading bandwidth.

The decentralized overlay network organizes peers in a random graph. It is a distributed system and also called mesh-based overlay network [3] [17]. Because there is no centralized server, peers in this kind of architecture need to transfer information with adjacent neighbors frequently. But it means that constructing this type of overlay network costs lots of control packets. Furthermore, since peers obtain information from neighbors, it is hard to get the

real-time situation of all peers in the system. Nevertheless, we think the decentralized architecture is suitable for our peer-to-peer streaming system. It could conquer the problem of peer churn. The reason is each peer has many peers to be its parents and has its own peer list. In addition, peers in the system could pick the parents that are suitable to them.

2.4 Forward Error Correction Codes

Peer-to-peer systems are mostly known from file sharing application. Why is peer-to-peer architecture widely used in file sharing services? This is because peers in file sharing only care whether it can get complete data. A large number of peers can alleviate the influence of unstable peers. Therefore you got more peer, you have higher chance to get complete data. However, the stability of a peer in video streaming system is more important than in file-sharing system. This is because video streaming service is time-dependent and each peer in system needs a stable streaming bit rate. High fluctuation of peer's survival time and heterogeneous uploading bandwidth could make the video streaming system unstable. Here, we use streaming data protection to solve peer fluctuation problem. Forward Error-Correction coding (FEC) makes sure that receiving some number of different encoded blocks can restore the original data. The main principle is to send out the extra data, making the wrong place can be identified and corrected. Next, let us introduce some channel codes with Forward Error-Correction characteristics

In channel coding, Reed Solomon codes [15] is one of the forward error correction codes, as shown in [figure 2.1] which is trying to explain the principle of operation. At beginning peer has k copies of message symbols, and then produces $n-k$ copies of the parity symbols by the RS coding. This coding guarantees to restore the original data if receiving k copies of

symbols. By the extra $n-k$ copies protection of data, this could achieve the feature of forward error correction coding.

Reed-Solomon code can have the ability to nearly achieve rateless digital fountain codes. But the R-S code has time-consuming problem and the decoding process has asymmetric computational complexity, if the amount of processing data is small, the decoding speed is acceptable, but if the amount of processing data is large, the decoding speed is very slow. Therefore we go toward a much faster error correction code such as: LT code [11], Raptor code [16] that are exclusive-or as the main calculation.

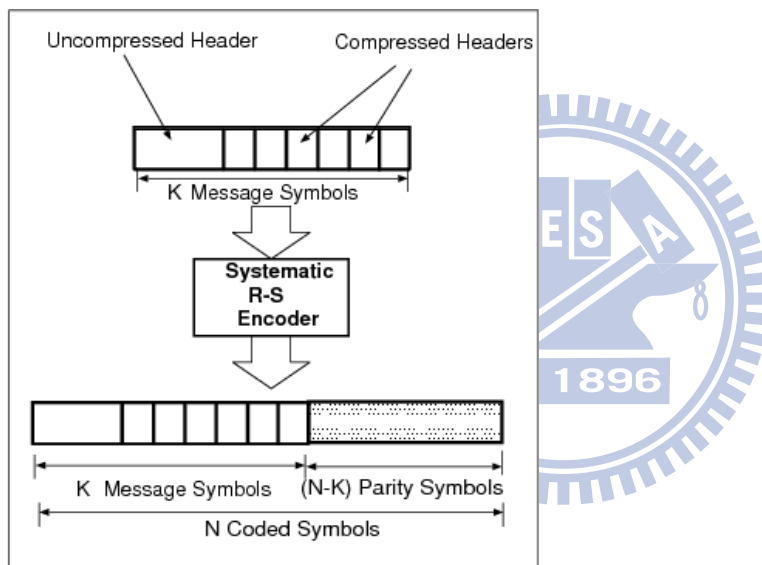


Figure 2-1 Reed-Solomon code [15]

2.5 Data Protocol and Peer Management

There are mainly two different ways in peer-to-peer streaming data protocol [21]. They are push based [7] [8] and pull based [4] [9]. The type of push based is that source decides to deliver what kind of data and when to deliver it. The type of push based is naturally associated with tree based architecture. Nevertheless, it always needs a super node to be source. When the users increase rapidly, the streaming service will be influenced. As a result,

[2] attempts to decrease the depth of tree for lowering the communication delay and overhead [26].

The other is pull based type [4] [9]. On the contrary, every peer periodically notifies its neighbors (parents) of what data it has. And the peer needs to select the data it wants and informs the parents of that. The type of pull based is normally connected with mesh based architecture. The weakness is that it costs a lot of control packets to get neighbors' information. And how to coordinate with parents is also a problem. In our system, we use the channel code to alleviate this problem. We will describe it in detail in next chapter.

The architecture we implement refers to [9]. [9] proposed an architecture that attempts to do application routing in peer-to-peer network. The application routing in peer-to-peer network means to select the parents that are good for you. [9] makes the decision of parent-selection dynamically by utilizing the residual lifetime of peers and other factors. [9] exploits a phenomenon in a peer-to-peer system. That is "the longer a peer lives in the system, the less probability it will leave later." And this peer lifetime distribution can be modeled by heavy-tailed distribution [12] [13]. In the process of research, we found some different issues and factors. Therefore, we attempt to enhance the method proposed in [9] and hope to create a stable peer-to-peer streaming system in real world.

Chapter 3. Proposed Method

3.1 System Architecture

In this section, we will introduce the system architecture [figure 3.1]. At first, we start with the view of the root (source).

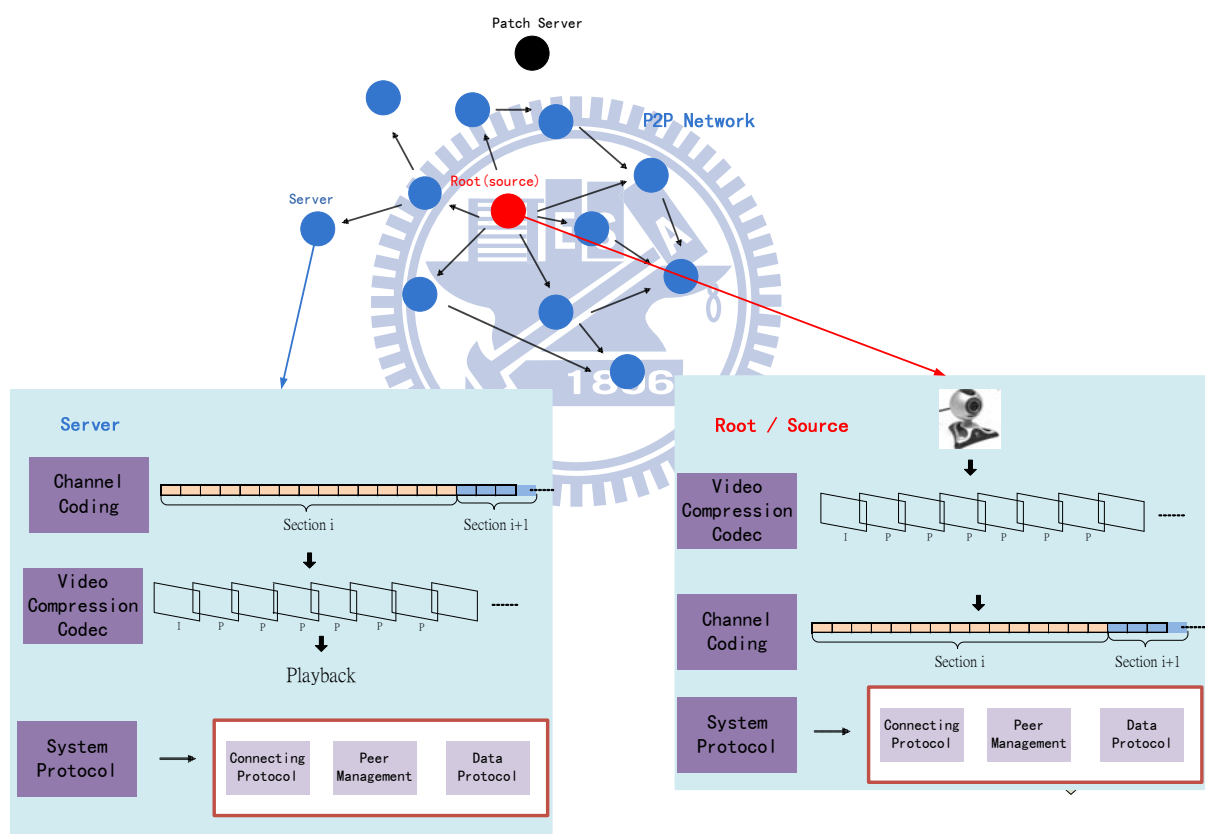


Figure 3-1 System Architecture

Root:

Step1: Initially, the root enters this system and sends a notification to the patch server.

Step2: The root needs to get the raw video data. The input could be video file or webcam.

Step3: Before sending the data, the raw video data needs to be compressed by video encoder.

Step4: After the process of video encoding, we put the encoded video data into channel encoder.

Step5: After encoding a section of data, root keeps sending data to its downstream nodes in peer-to-peer network. The root will go back to **step2** to get the next raw video data and keep going.

Server:

Step1: Initially, the server enters this system and sends a notification to the patch server. It will get the reply and follow the connecting protocol to get its parents.

Step2: The server will ask the video data by the data protocol.

Step3: The server starts to decode the data by channel decoding.

Step4: After the process of channel decoding, the server gets the encoded video streaming data. It needs to be decoded by video decoder.

Step5: The server starts to play the raw video data and keeps sending the data to its downstream nodes in peer-to-peer network. The server will go back to **step2** to get next video data.

The steps described above attempt to simplify the whole process in our streaming system.

Next, we will explain each component in the peer [figure 3.2].

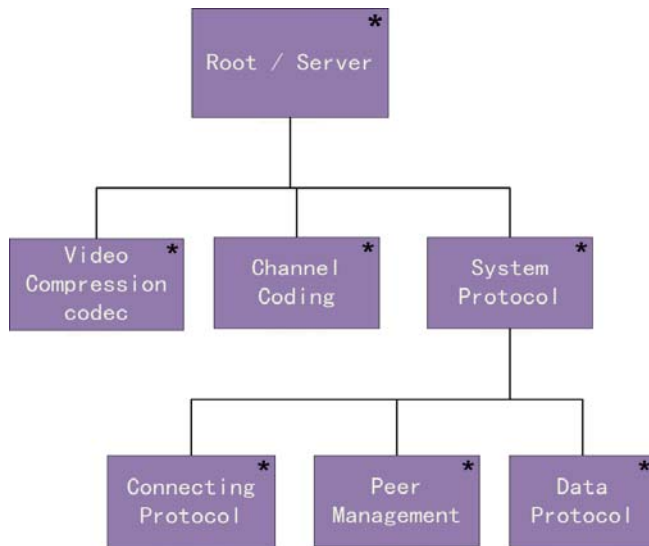


Figure 3-2 The component of peer.

3.1.1 Video Compression Codec

This component is used by the root. We attempt to decrease the total data bits of transmission by video compression codec. This will help each peer in our system. The video compression codec we used in our system is Windows Media Video 9(VCM) [10]. [10] is the Microsoft implementation of the VC-1 SMPTE standard. It provides high-quality video for streaming. This codec supports constant bit rate (CBR) and variable bit rate (VBR) encoding. And the video compression picture type we used is I-frame and P-frame. For twenty continuous P-frame, we will insert an I-frame [figure 3.3].

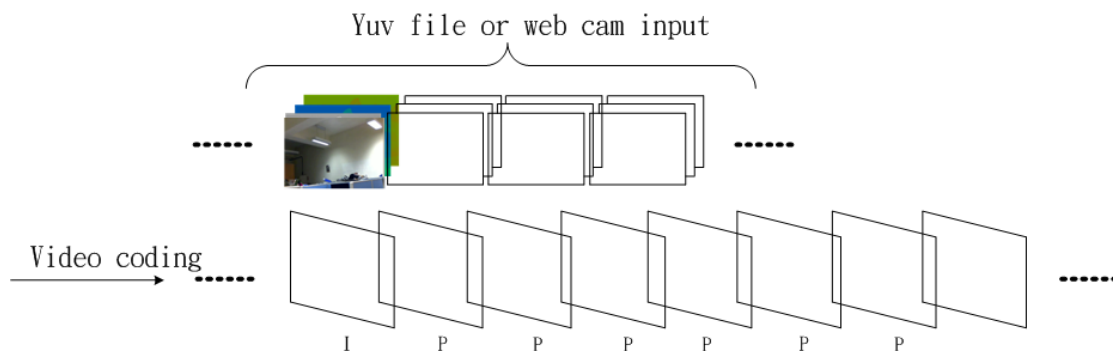


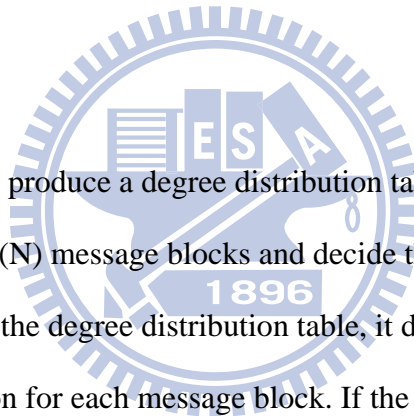
Figure 3-3 The process of video compression

3.1.2 Channel Coding

The channel coding we used is Luby Transform Codes (LT codes) [11]. LT codes makes sure that receiving fixed number of different encoded blocks could restore the original data with high probability. Furthermore, it uses simple XOR to encode data so it has low complexity on encoder and decoder. These properties make that the LT codes is suitable for peer-to-peer streaming system. Peer does not need to coordinate with its parents frequently. It greatly reduces the number of control packets and simplifies the data transmission protocol. Moreover, the overhead of using LT codes is low. Next, we will introduce the process of encoder and decoder.

The encoder:

First of all, we need to produce a degree distribution table. After that, we have to cut original file into fixed-size (N) message blocks and decide the degree (d) of each block by a random seed. According to the degree distribution table, it decides that how many different blocks doing XOR operation for each message block. If the degree is equal to one, and then the message block is equal to the encoded block. [Figure 3.4] is the process of producing encoded blocks. Each encoded block also needs to add the corresponding random seed for decoder.



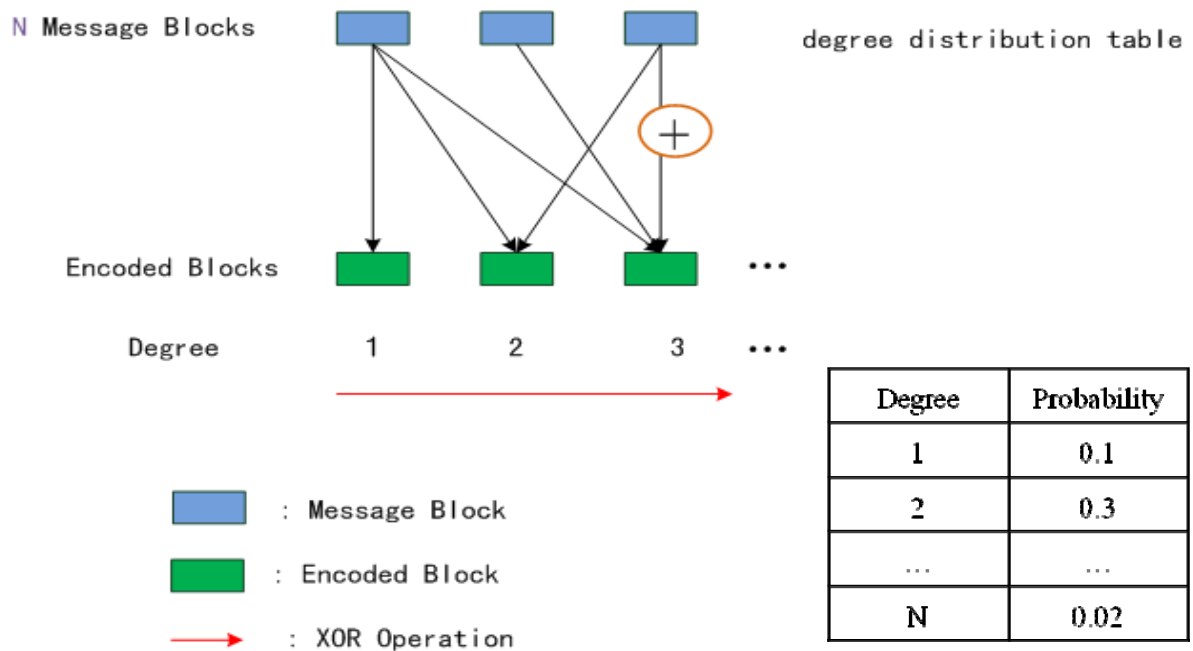


Figure 3-4 LT codes Encoder

The Decoder:

We can get the same relationship between message blocks and encoded blocks by random seed the encoder provided. Decoding procedure [Figure 3.5] keeps finding the degree one encoded blocks. Because the message block is equal to the encoded block. The message block with degree one is restored immediately. After restoring some blocks with degree one, we can start to restore blocks with degree two. After XOR operation, the encoded block with degree two decreases its degree to one. Doing above procedure repeatedly, we can restore all the message blocks.

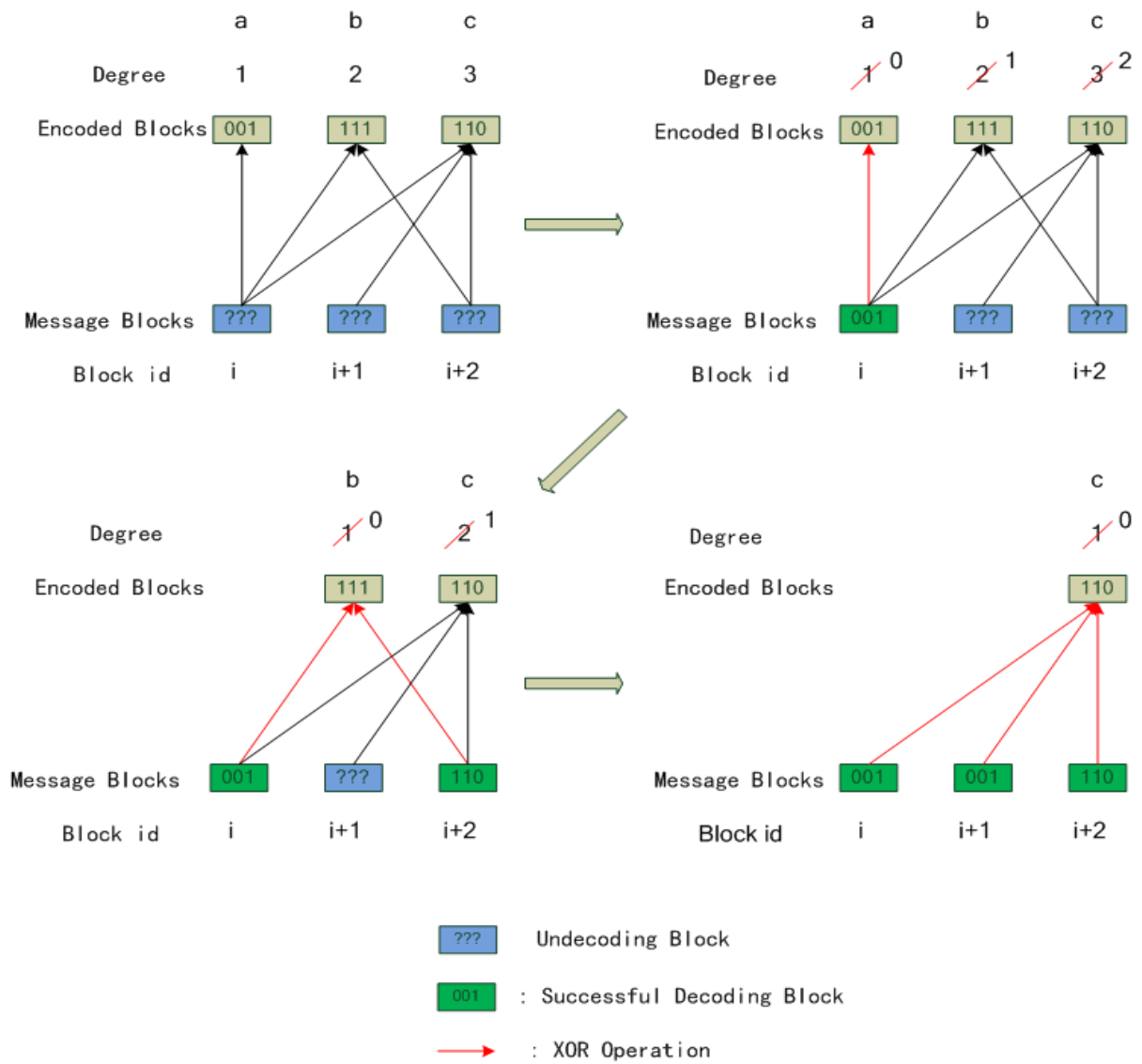


Figure 3-5 The decoding process of LT codes

[Figure 3.6] shows the complete LT encoding process of root. After getting frame from webcam or file, we put the frame into video encoder. And the size of output frame is different every time. No matter what, we accumulate each VCM9 encoded frame into buffer until the total size is equal to or over the section size of LT code. Because it is nearly impossible that the size of accumulated VCM9 encoded frame is equal to a section size of LT code, there always exists wasted space [Figure 3.6]. Our solution is to cut the VCM9 encoded frame and put the remaining data to the next section.

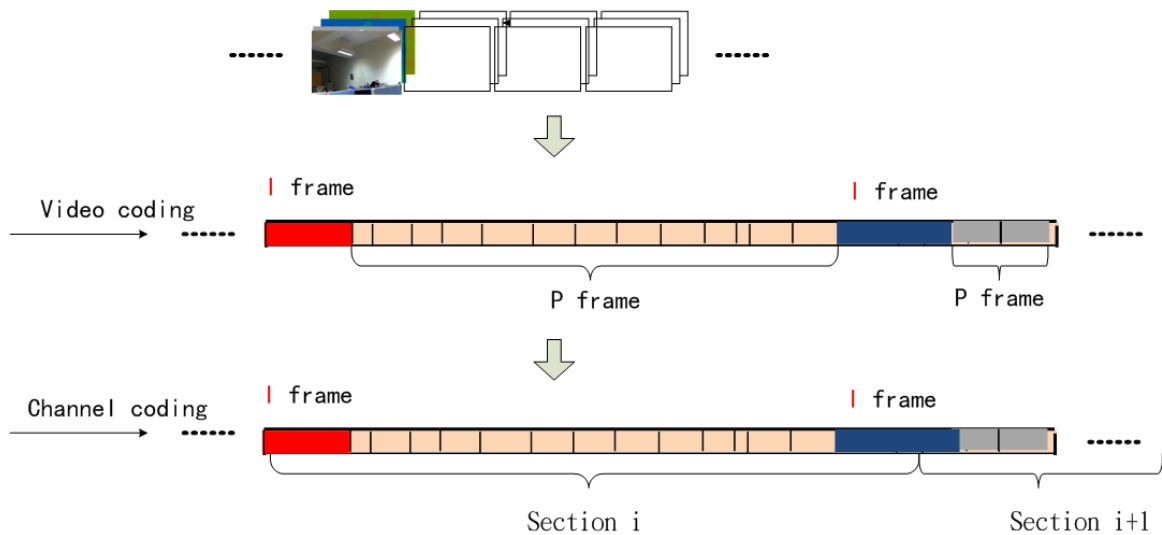


Figure 3-6 The Complete LT Encoding process of Root

3.2 Connecting Protocol

Before every peer in the system starts to receive data, they need to follow the connecting protocol of this system. [Figure 3.7] shows the flow of connecting protocol. We will describe each step in detail.

Step1: Initially, the peer (server) enters this system and requests a peer list from patch server.

Step2: The patch server records every peer in this system. After receiving a request, it will reply a random peer list.

Step3: After getting the peer list, the server will parse each peer's information in the peer list. Next, the server sends the request packets for joining to every peer in the list. We call these peers in the list candidates.

Step4: These candidates have to decide the reply of the request by their ability. If their number of children exceeds the threshold, they have to reject the request.

Step5: After receiving the replies of these candidates, we have enough information to calculate the score for each candidate. We will define the calculation of score later. Finally, we

choose the candidates with higher scores to be our parents.

Step6: In this step, we send the packets for confirming the joining action. When the total number of parents exceeds the threshold, the server will end this connecting procedure. Or the server goes back to setp4 and picks the other candidates with higher scores.

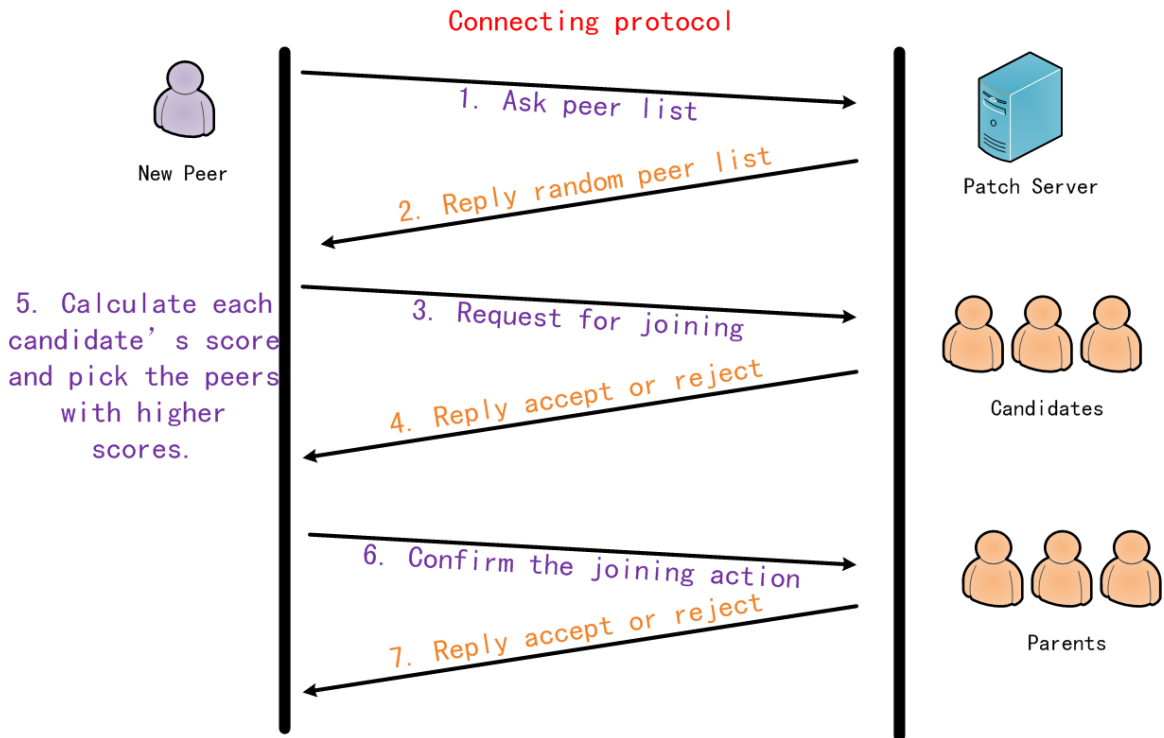


Figure 3-7 The Connecting Protocol

[Figure 3.8] shows the format of communication packet. Peer can obtain basic information of some peers from this packet. Initially, a new peer receives a peer list from patch server. The header of the packet contains a new ID for this peer and the rest has some peers' information. We call these peers candidates. The peer saves all of these candidates' data and starts to contact them.

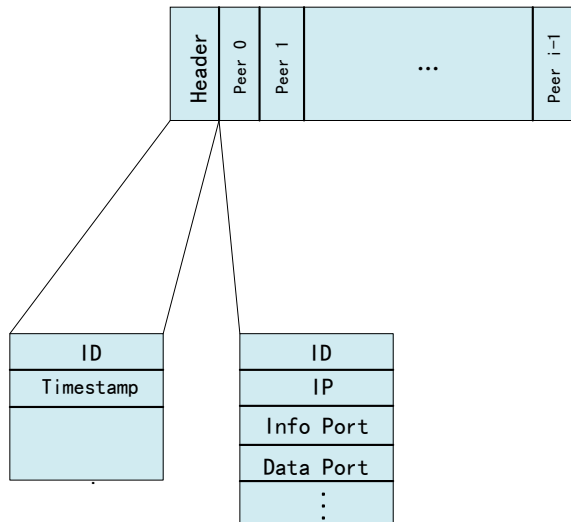


Figure 3-8 The Format of Communication Packet

3.3 Peer Management

In peer-to-peer network, how to manage your peers is very important. The reason is the peers you choose affect your application's performance. Therefore, how to save and manage the peers in the system becomes a big issue. Next, we will introduce our peer management thoroughly.

Initially, we can get some peers by connecting protocol. The information of peers contain peer's IP, peer's port, peer's score, and ... etc. We save our peers in a peer pool [figure 3.9]. We need to keep updating the data in the peer pool until we leave this system.

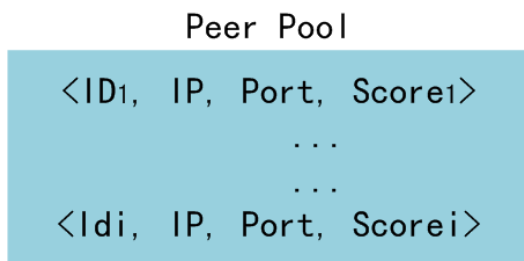


Figure 3-9 The peer pool

Secondly, we explain how to calculate the score of a peer and how to implement them in our system. We use four parameters to measure the capability of a node. These four parameters are the available bandwidth (ABW), and source delay (Fresh Level), the usability of data (effective ratio), and the variance of packet delay (jitter).

Available Bandwidth (ABW):

This parameter is used to know the available bandwidth the parent can provide. When peer wants to know available bandwidth of its own, it needs to cumulate the available bandwidth of all parents. Before cumulating the available bandwidth for each parent, we take the bandwidth multiplied by the reliable probability. Based on the phenomenon in a peer-to-peer system (“the longer a peer lives in the system, the less probability it will leave later.”), [9] uses the Shifted Pareto Distribution [25] to point out the reliable level $R_i(t)$ for peer i with its lifetime t .

Finally, we compare the cumulated bandwidth with own uploading bandwidth and take the smaller one [equation 1]. The one we get is our available bandwidth. As a result, this factor represents the minimum supportable bandwidth of a peer.

$ABW(i) = \min\left(\sum_{j \in i'sParent} (ABW(j) * R_i(t_j)), UBW_{self} \right)$	(1)
--	-----

Effective Ratio:

This parameter wants to measure the usable data ratio of the parent after finishing a section of data. It could express the real contribution of this parent node. With this parameter, you can discount the scores of the parents that send repeated or overdue data. Therefore, we measure this parameter for each parent after decoding a section of data. At first, we take the repeated or overdue data the parent (i) sent divided by the total data parent (i) sent [equation

2]. And then, we take one minus previous result. In the end, the result is the effective ratio of parent (i).

$\text{effective ratio}(i) = 1 - \frac{\text{repeated or overdue data parent } i \text{ sent}}{\text{the total data parent } i \text{ sent}}$	(2)
---	-----

Jitter:

In the real world, the peer under high background traffic may influence its own uploading quality. So we hope to add jitter evaluation to peer’s score calculation. The definition of jitter is the variance of packet delay. [equation 3] shows the calculation of jitter. The setting of weights refers to [14]. When we implement this parameter in our system, the time of packet delay can be regarded as the relative time difference. As a result, although the time in the different computers is always not the same, the implementation is much easier.

$jitter_{new}(i) = jitter_{new}(i) * 0.0625 + jitter_{old}(i) * 0.9375$	(3)
---	-----

Fresh Level:

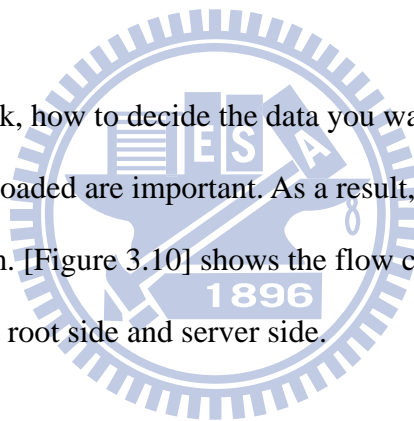
Setting this parameter is for measuring the source delay of a peer. It can show the peer’s fresh level of data. Each peer in the system always wants to play the newest video streaming data. This is why we set this parameter. But how to implement in our system is a problem. In general, the time in the different computers is always not the same. We can only calculate the relative time difference of the adjacent two peers. Every data packet records the sender’s source delay and the receiver is able to get own source delay. And peer takes the longest source delay as its own source delay from its parents.

$Score(i) = w_1 * FL(i) + w_2 * ABW(i) + w_3 * ER(i) + w_4 * Jitter(i)$ <p>where w_1, w_2, w_3 and w_4, are weighting factors.</p>	(4)
--	-----

Finally, we have to calculate the final score of a peer by these four factors [equation 4]. After that, we will pick the peers with higher scores. Furthermore, we need to choose the weighting of each factor. We will explain this part in next chapter. Nevertheless, peers' condition may change with the time. We have to adjust the peers' score periodically. Our coding is the section based of LT codes. Therefore, we adjust each parent's score after finishing a session.

3.4 Data Protocol

In peer-to-peer network, how to decide the data you want to download and how to manage the data you downloaded are important. As a result, we attempt to design a data protocol that fits our system. [Figure 3.10] shows the flow chart of our data protocol. Next, we describe this protocol at root side and server side.



Encoding Procedure for Root

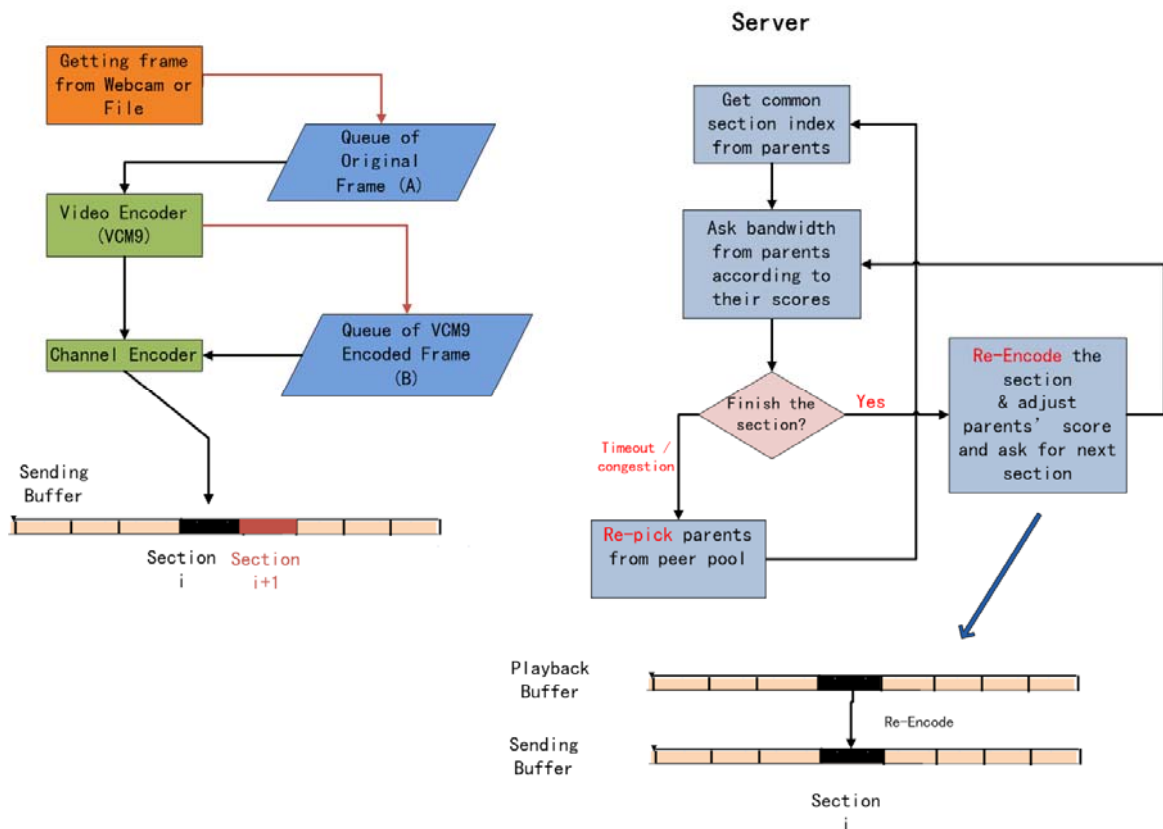


Figure 3-10 The data protocol

Root:

The data protocol in root is intuitive. The job of root is keeping encoding the data and sending the encoded data to its downstream nodes. At first, we need to keep obtaining original frame from the webcam or file. And we save these frames into the queue. Next, there is a thread keeping encoding the original frame from the queue (A) by VCM9 codec. After encoding a frame, it puts the encoded frame into queue (B). When the queue (B) accumulates enough size for channel coding, it will send a notification to channel encoder. Finally, we have a sending buffer storing the encoded data. The size of a sending buffer represents the maximal number of section you can keep. This buffer is also a circular buffer. When root puts the encoded data into the corresponding position which is filled with old data, the newer data will cover the older one. When the downstream nodes ask the section of data root has, root

starts to send data from the sending buffer according to their allocated bandwidth. We explain how to allocate the bandwidth of children later.

Server:

The data protocol in server is much complicated than root. There are two buffers in each server. One is the sending buffer which is same as root. The other is playback buffer storing the decoded data. First of all, after choosing parents, server needs to decide the initial section index among the parents. We will discuss how to decide the index in the next part. When we finish picking the initial index, it is time to allocate the requested bandwidth from parents. We believe that the parents with higher scores are more suitable for us. As a result, we allocate the bandwidth according to their scores [equation 5]. The next step is waiting for the completion of this section. However, we can't keep on waiting for infinite time. If the time we wait exceeds the expected time or no more streaming data can be played, we start with the timeout mechanism. The expected time is the total bits of a section divided by the target bit rate. Because real time streaming data is time-dependent, we hope to have a stable throughput. The timeout mechanism helps us to detect the condition of our streaming service. If we decode this session within our expected time, we put the decoded data into playback buffer. Furthermore, we re-encode the decoded session by channel encoder and put it into the sending buffer. This action increases the data diversity of whole system. When the peer receives data from upstream nodes, the chance of receiving repeat data (encoded block) is much lower.

$\text{AskingBandwidth}(i) = \frac{\text{Score}(i)}{\sum_{j \in \text{Parents}} \text{Score}(j)} \times \text{TargetRate}$	(5)
--	-----

3.4.1 Adaptive index

After choosing our parents, we need to decide which section of data to download. In the same time, our parents have to reply the section of data they download now. Therefore, we discuss the issue on the two sides.

First of all, we have to choose the section replying to the children according to our sending buffer. [Figure 3.11] shows a peer's condition of the sending buffer. The playback index is the section index of the data peer watches. And the decoding index is the section index of the data peer receives. We can see the difference between playback index and decoding index is D . At the beginning of playing the video stream data, D should be the same as the initial buffer size. A peer's initial buffer size is the amount of data prepared for playing. We set the value to be two sections. Nevertheless, with the change of peer's downloading ability, D will vary over time. If the throughput of the peer drops off, D will be smaller. If we send the decoding index to our children, we may not be able to transmit the section of data to them. Because of this condition, we send the decoding index minus one to our child. If D is the same as initial buffer size or bigger, the throughput of the peer might increase or keeps stable. It means we have high probability to decode the section we receive successfully. As a result, we reply the decoding index to our child.

After obtaining each parent's decoding index, we have to decide the initial section index. The strategy we use is to choose the newest and maximum section index the parents have. We want the newest data and get help from more parents. We think it is the best way for our system.

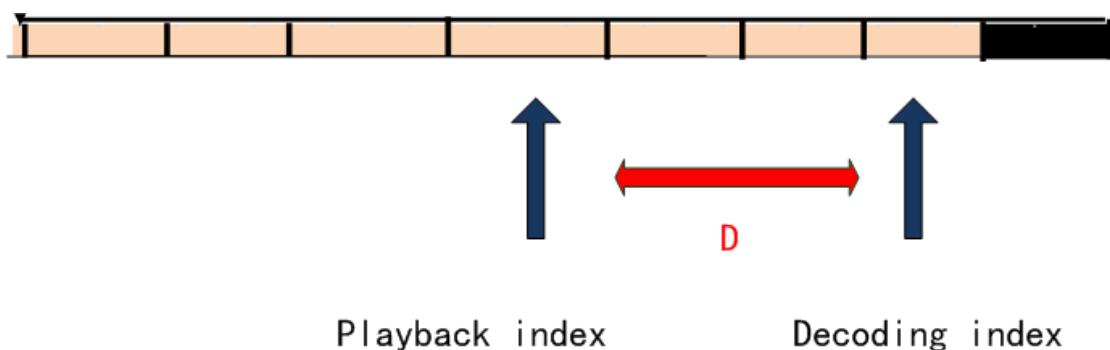


Figure 3-11 The condition of the sending buffer

3.4.2 How to allocate the bandwidth of children and parents

How to manage the download and upload bandwidth in peer-to-peer network is important. Because we discussed how to allocate download bandwidth, now we describe how to allocate children's bandwidth. The strategy we take is to allocate bandwidth fairly for all the children [figure 3.12]. At first, we need to find out the smallest upload bandwidth for all children. The spirit of strategy is that we attempt to satisfy each child with the smallest upload bandwidth. If we still have uploading bandwidth left, we keep satisfying the remaining children with the smallest amount of bandwidth until the total bandwidth is allocated.

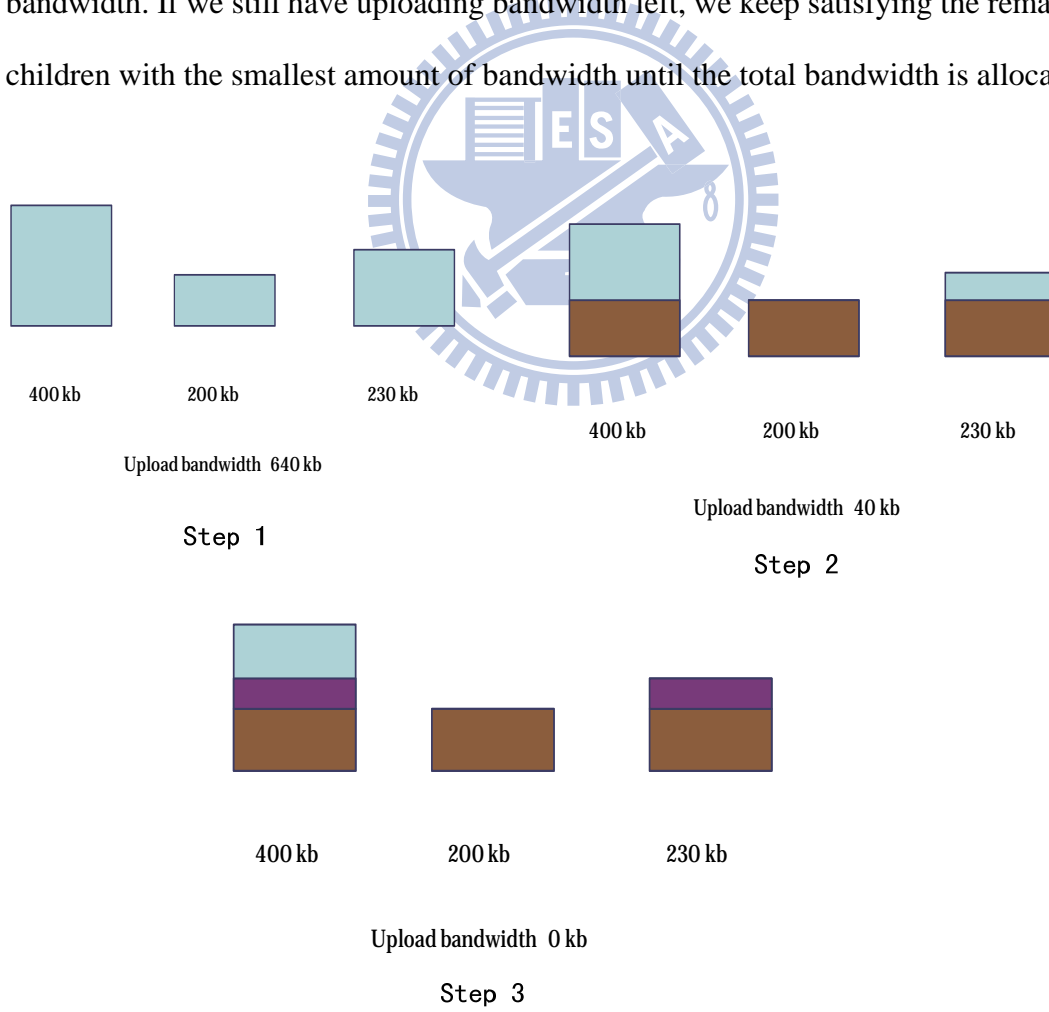


Figure 3-12 The process of allocating children's bandwidth

3.4.3 Communication Protocol

In the real world, it is hard to let every computer obtain a public IP. It means there are a lot of peers under the NAT environment. We use the user datagram protocol (UDP) to solve NAT traversal problem. [Figure 3.13] shows how our system passes the NAT environment by UDP. We open two ports for our system. The one (info port) is for sending the communication packets. The other (data port) is for sending the data packets. When we ask peer list from patch server, we need to send the request packet through the two ports. The patch server records every new coming peer's public IP and ports. Before we ask the data packet from parents, we need to send the test packet to parent's data port. That is because we want to create the mapping from our internal IP and ports to the parents' external IP and ports in the router. Therefore, the data packets from parents can be transferred to us by the mapping in the router.

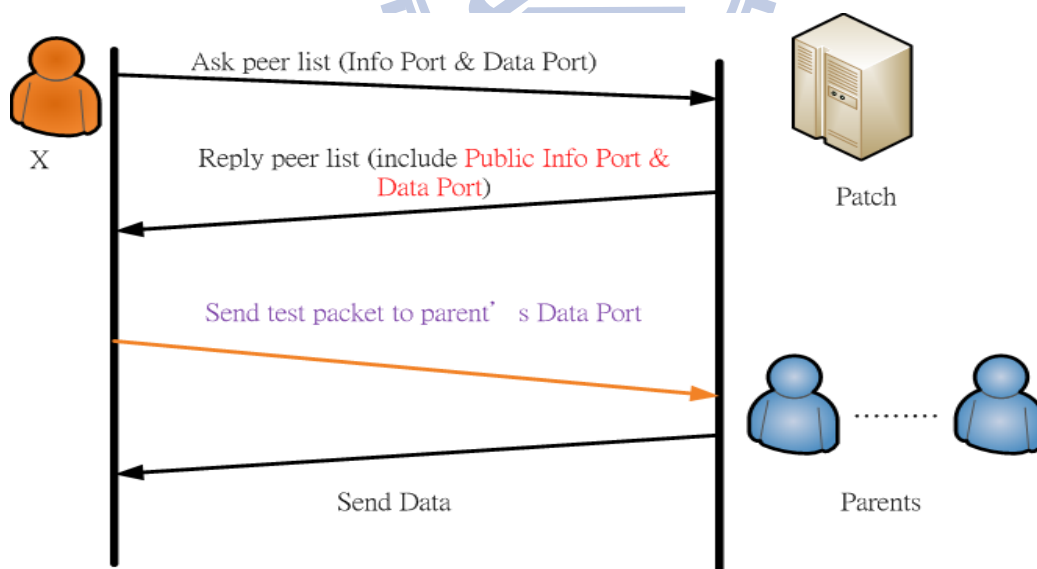


Figure 3-13 The process of passing the NAT environment

3.4.4 Packet Protection

Because UDP is a message based connectionless protocol, the packet may be lost in the process of transmission. Therefore, we provide the data protection for our system. We use different level of protection according to the importance of packets. At first, we use ACK mechanism for the regular communication packets (for example: the packets for asking peer list, joining request, leaving notification ...etc). If a peer receives this kind of packets, it needs to reply an ACK packet for notifying the sender [Figure 3.14]. If the sender does not receive the ACK packet after a limited time and the number of resending times for the parent is smaller than the MAX, we will resend the packet. The MAX is the threshold of resending times. If the number of sending times is bigger than MAX, we suppose that the peer leaves this system and remove this peer from peer list.

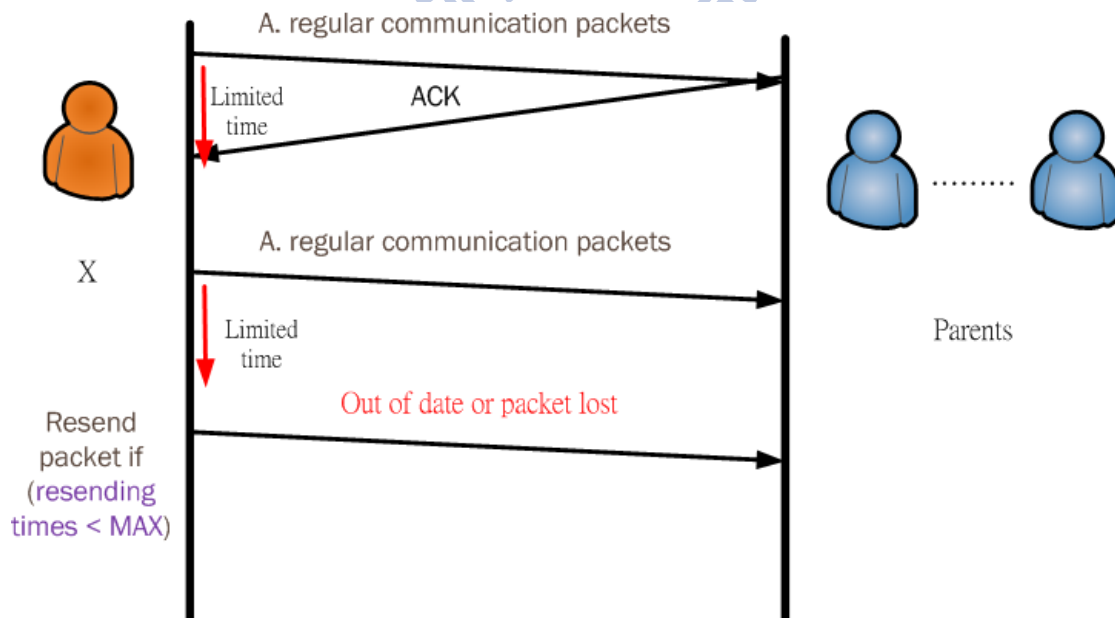


Figure 3-14 Data protection for regular communication packets

When peer decodes the section of data successfully, it needs to request the next section of data from its parents. If a peer sends this kind of communication packet, it will send the same communication packet twice [Figure 3.15]. The reason is if the receivers lose this communication packet, we will receive a bunch of overdue data packets. To prevent this situation, we send this communication packet twice.

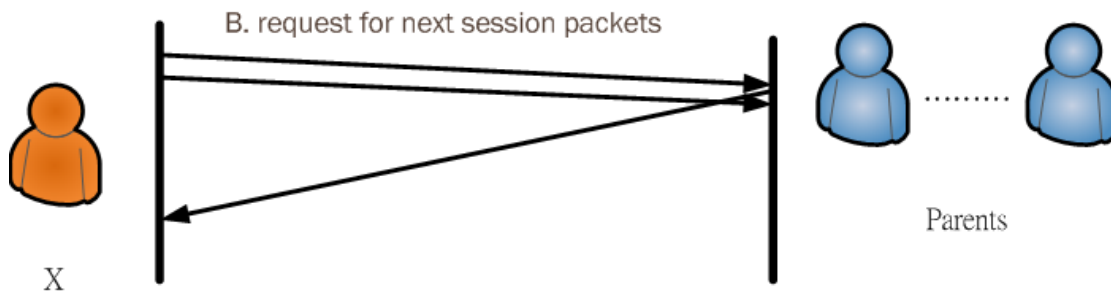


Figure 3-15 Data protection for important communication packets

For the data packet, we do not provide any protection mechanism. That is because we use the rateless code to encode our data. The encoded packets in the same section are time-independent. For the receiver, as long as it receives enough packets, it has the ability to retrieve the original data. Even if we lose a few packets, the decoding will be affected slightly.

Chapter 4. Simulation Result

4.1 Simulation Environment

In this chapter, we attempt to observe the experiment result in the simulated peer-to-peer environment. Based on the source code [9] provided, we modify the code according to our proposed method. Furthermore, we compare the original method with our method in our designed environment. The simulation platform is NS-2 2.34 and the network topology is generated with BRITE topology generator.

The topology is generated by some rules. [Figure 4.1] shows one of the topologies. There are two hundred nodes in the system and we try to let some nodes have background traffic. In real word, people who are watching the video streaming probably upload some files to someone at the same time. That is the reason we set our simulated environment like this.

The core nodes (blue) are generated by topology generator. And we put the peers on the leaf. We set the background traffic as UDP/CBR. The nodes with red color and circular shape are the senders with background traffic. And the nodes with red color and square shape are the receivers. We will describe the experiment in detail later.

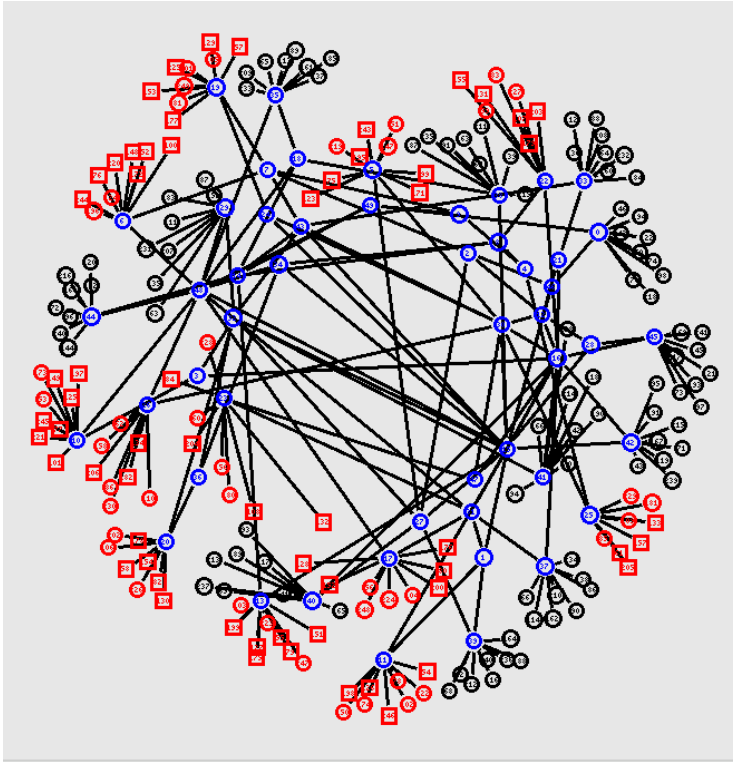


Figure 4-1 The network topology

Peer upload/download capacity	1024 Kbps
Source upload capacity	10 Mbps
LT block size	1024 Bytes
LT section size	128 K Bytes
Target bit rate	512 Kbps
Buffer size	2 sections
Simulation time	600 seconds
Total peers	200

Table 4-1 The parameter setting

4.2 Weight Setting

Recalling the [formula 3], we need to decide each factor's weight. Our goal is to find a good combination of weights. We let the summation of these four weights equal to one. Thus, we lock our searching range on the plane of the equation $w_1+w_2+w_3+w_4=1$. [Figure 4-2] shows the searching process. The coordinate is the weights combination of each factor. They are fresh level, available bandwidth, jitter and effective ratio in order.

At first, we cut the original square shown in Figure 4-2 into four small squares. Each black node in the original square is the gravity of each small square. We do experiments to evaluate these four black nodes. After choosing the best black node in the square, we cut the corresponding small square into four smaller squares and obtain four red nodes. And we do the same evaluation to get the best red node. But how to pick the best node from these four nodes is important. We use four benchmarks to evaluate these four nodes. We separately rank these four nodes by the four benchmarks. If the node ranks first at one of the benchmarks, it gets one score. From the average result of many experiments, the score of the best black node is three and the score of the best red node is four as shown in Figure 4-2.

The standards of evaluation are the average effective throughput, average of playback continuity, average of jitter and average of source difference in decoding index. The effective throughput of a peer is computed by non-repeat and non-overdue data packets (encoded blocks) for every two seconds. The playback continuity of a peer is the ratio of successful playback before deadlines to the total sections. A peer's average jitter is the average variance in the delay of all data packets. The average of source difference is the average of the decoding section difference of all peers from source for every 10 seconds. In order to observe the influence of jitter, we add background traffic to half of total nodes in our experiment.

After a series experiments, we get the better combination of weights (3/64, 21/64, 35/64, 3/64). The jitter and the available bandwidth are much more important than other factors. Because the peers with smaller jitter do not have background traffic to influence their uploading capacity, we think they have high probability to supply more stable bandwidth for

their downstream peers. And the peers with high available bandwidth could make more contribution to downstream peers than the peers with low available bandwidth. Having sufficient and stable download bandwidth is important for peers in the system.

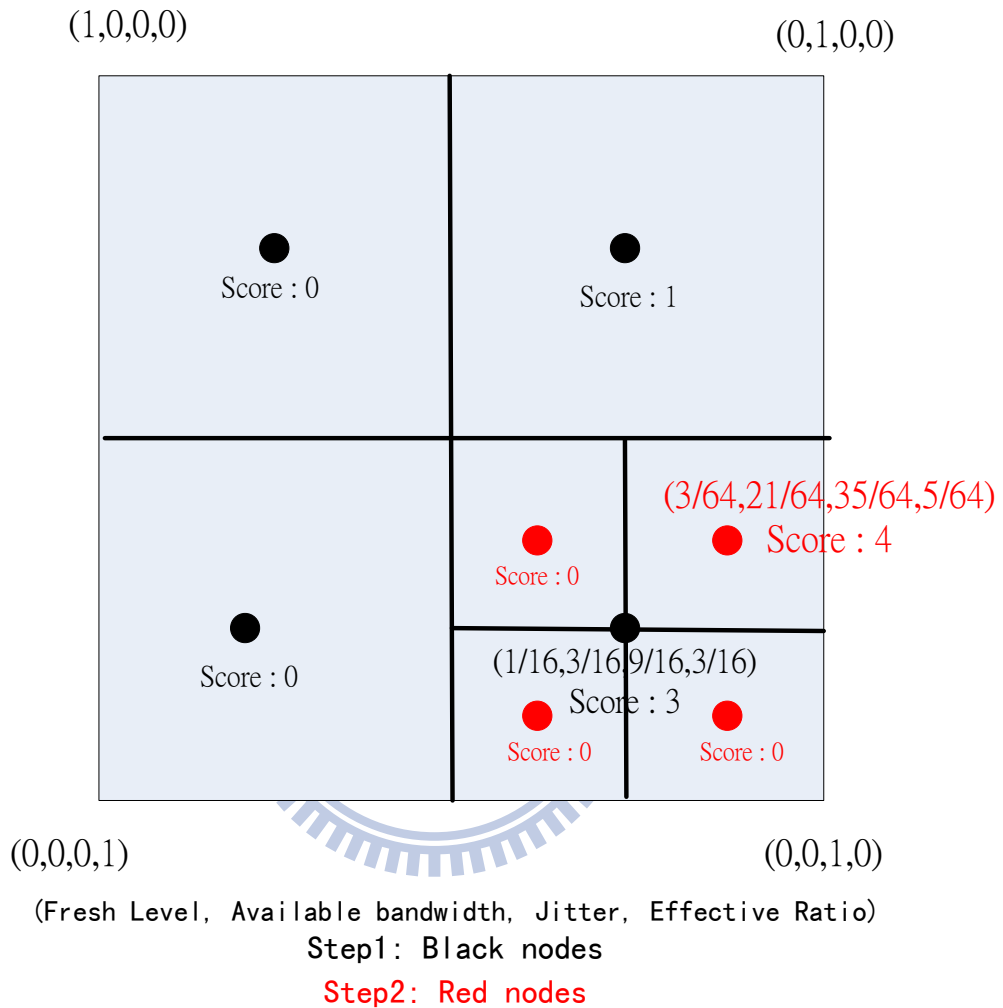


Figure 4-2 The searching process of weights

4.3 The Environment under background traffic

In this section, we want to compare the result to the original method [9]. The method [9]

proposed did not concern about jitter. As a result, in order to observe the influence of jitter, we add background traffic to the half of total nodes. The background traffic we use is CBR/UDP. It is running during the entire simulated time.

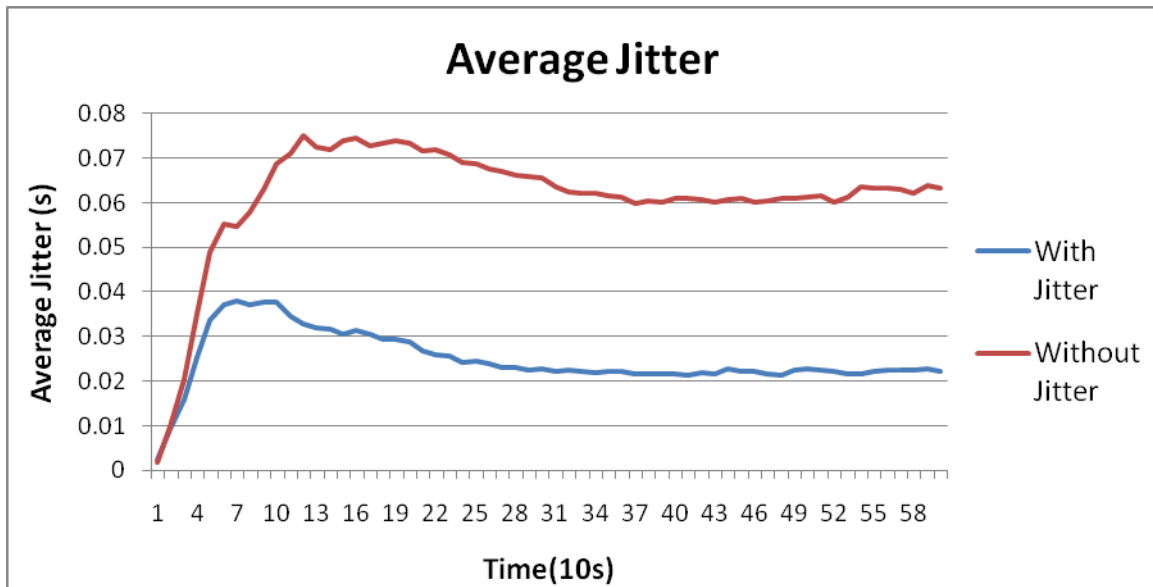


Figure 4-3 The average jitter under background traffic

[Figure 4.3] shows the average jitter under background traffic. The interval in the horizontal axis represents 10 seconds. The blue line is the method considering jitter and red line is the method not considering jitter. We can see difference of jitter when the CBR traffic is running. We set the rate of CBR is 512Kbps and the outside link capacity of peer is 1024Kbps. The whole system suffers from the damage of background traffic. And we can observe the method considering jitter has smaller average jitter than the compared one.

Next, we want to see the continuous playback index of these two methods [Figure 4.4]. Because we make the background traffic have influence on uploading capacity about 50%, we can see the great impact on the continuity of playing streaming video. The method with jitter has smoother streaming quality than without jitter. We believe that our method selects fewer peers with background traffic as parents. And we will verify it at next part. However, we can see the method without jitter [9] performs a little bit better after 210s. The reason is shown

below. At first, the peers may select the nodes with background traffic as parents. After a period of time, these peers probably do not have smooth video streaming quality. In the method [9], it adjusts the score of parents by the effective ratio after finishing a section of data. It means the peer with discontinuous video streaming quality may start to update their parents. In order to verify our thought, we do an experiment for observing each peer's change level of parents [Figure 4.5]. We calculate this parameter for every 2 seconds. Each peer recorded the parents' ID two seconds ago and compared with the parents' ID for now [equation 6]. For example, node A had the ID vector of parents (1, 5, 25, 33) to be its parents two seconds ago and had (1, 4, 5, 33, 55) to be its parents for now. The changing level of parent for node A is $2 / 5$. As a result, if the node does not change its parents, its changing level of parent is zero.

From [Figure 4.5], we can see the method without jitter updates their parents frequently between 110s and 210s. Therefore, that is why the method [9] proposed performs a little bit better after 210s. On the other hand, the method with jitter picks much fewer nodes with background traffic in the process of parent selection. As a result, our method enjoys the stable video streaming quality.

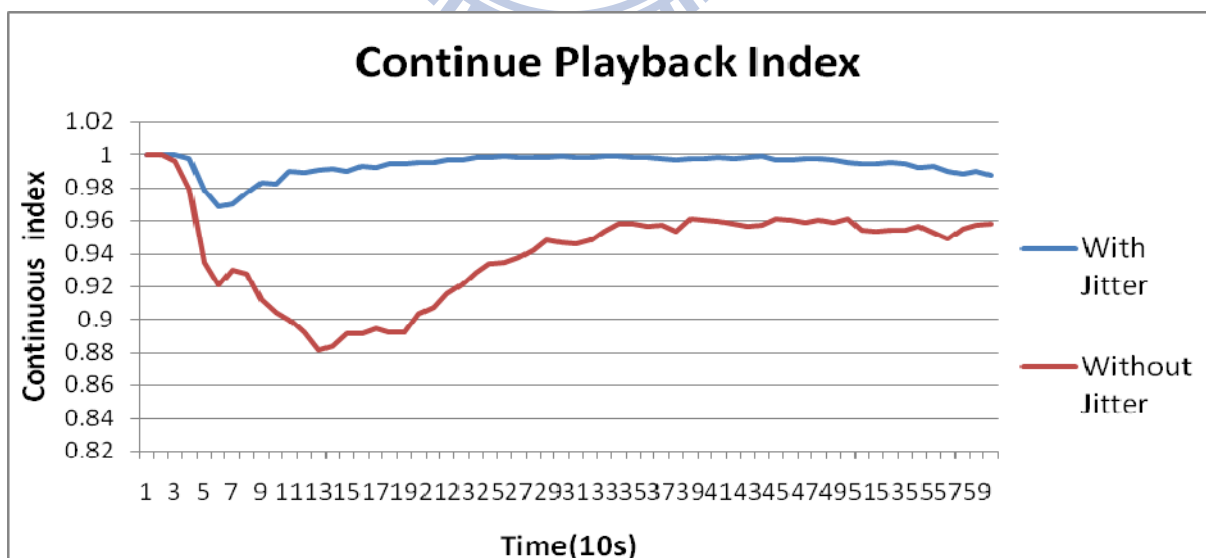


Figure 4-4 The continuous playback index under background traffic

$\text{changing level } (i) = \frac{\text{the count of different parents}}{\text{Max}(\text{total number of parents (now)}, \text{total number of parents (2s ago)})}$	(6)
--	-----

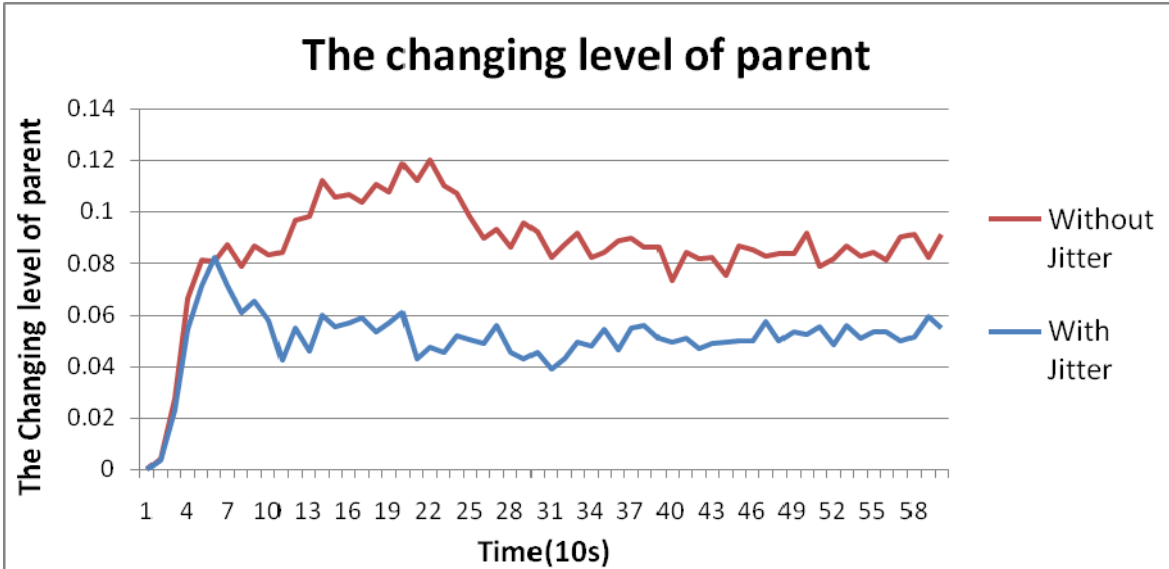


Figure 4-5 The changing level of parent under background traffic

Next, we observe the effective throughput of these two methods [Figure 4.6]. The blue line is our method and the average is 640.45 kbps. The average of red line is 629.33 kbps. We can see our method has higher effective throughput than the method [9].

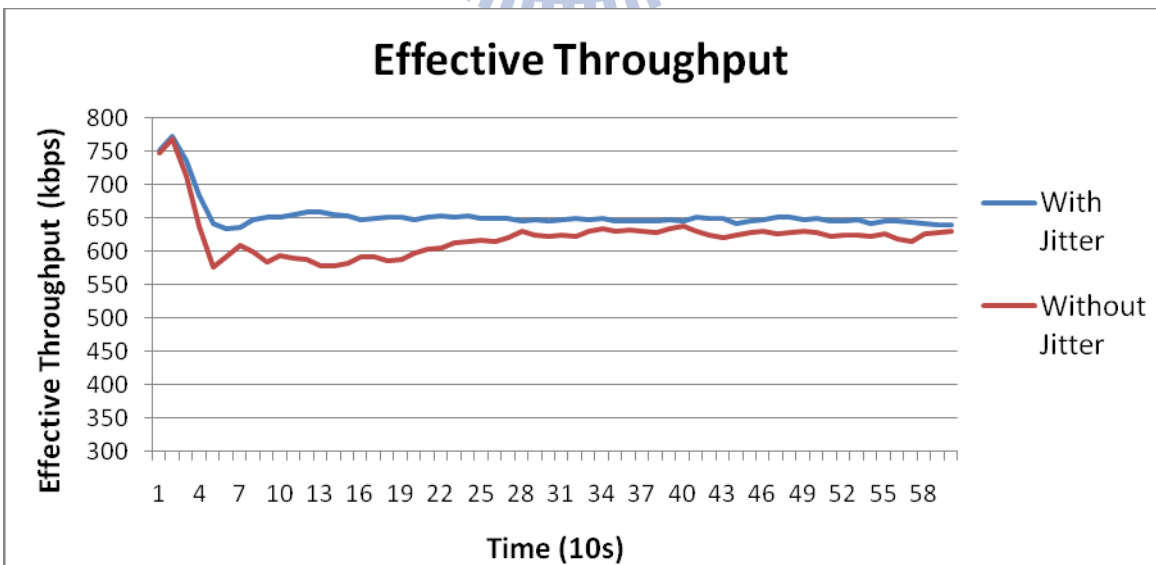


Figure 4-6 The effective throughput under background traffic

[Figure 4.7] shows the source delay of these two methods. Because our encoded data is

section-based, we calculate the source delay by section difference. The average of our method is 1.312 sections. The average of red line is 3.313 sections. We believe our method enjoys newer video data than the method [9].

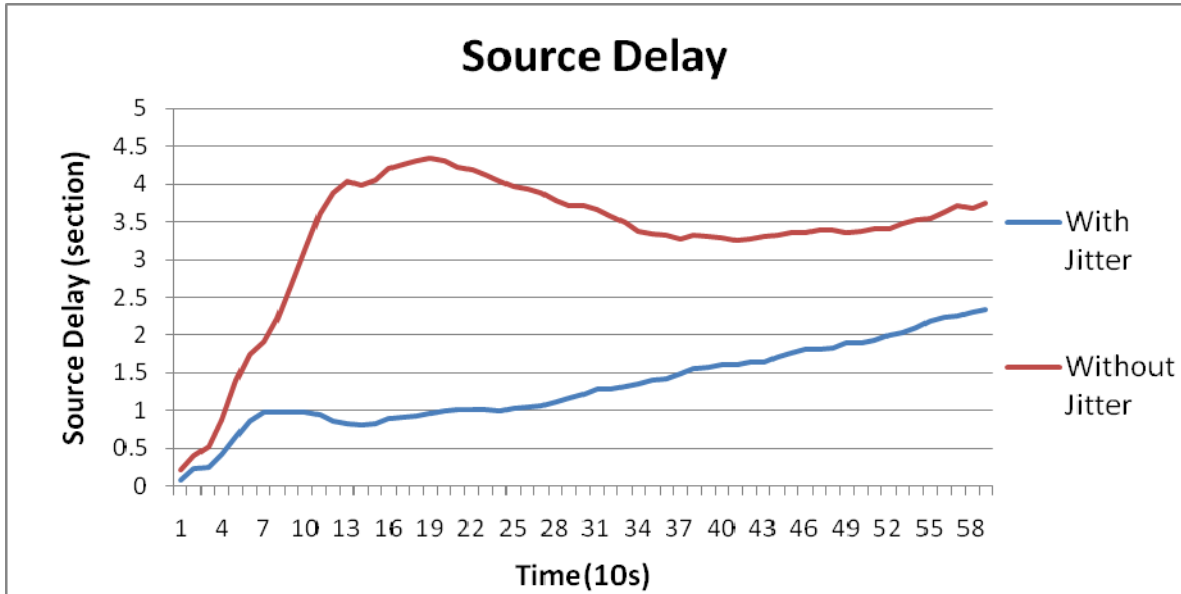


Figure 4-7 The source delay under background traffic

Finally, we check the ratio of the parents with background traffic for the two methods [Figure 4.8]. In this experiment, we want to check that our method takes effect on parent selection in the environment with background traffic. The blue line is our method and the average is 0.0832. The average of red line is 0.2053. Each peer in the system has 5 parents in average. In other words, the method [9] proposed selects about one peer with background for their parent in average. We believe that this is the main reason to explain why our method is better than original method [9]. Consequently, we consider our method is good for the system under background traffic.

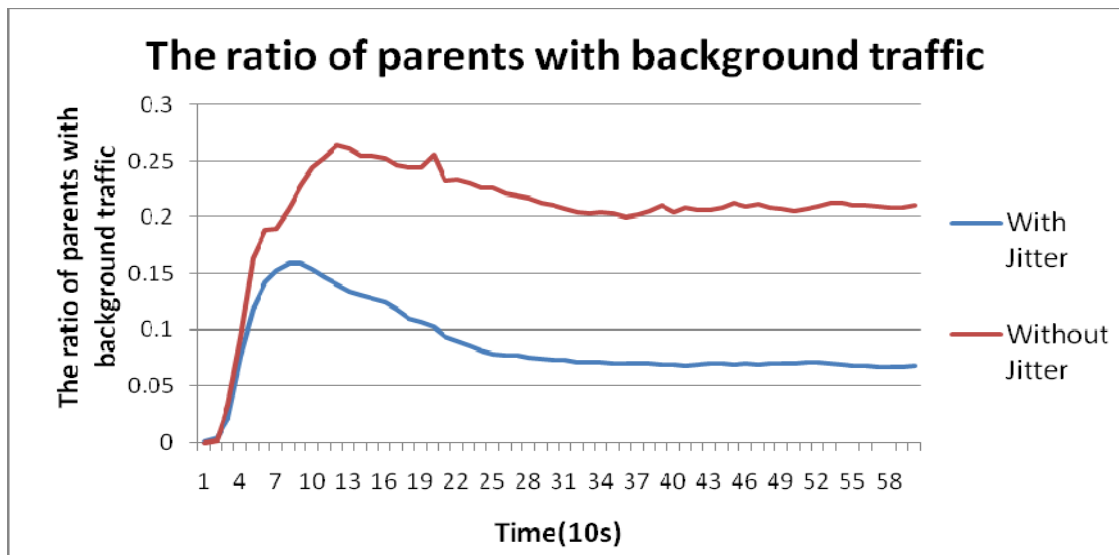


Figure 4-8 The ratio of the parents with background traffic

4.4 The real-world implementation

We want to introduce our system interface [Figure 4.9] at this part. At first, the resolution of display is 640x480 and the frame rate (E) is shown in the upper right corner. Each peer in the system has a unique identification (A) and it is shown in the upper left corner. Next, we can see the section index we are receiving (B) and playing (C). We also have information about the parents and it contains the identification and score for each parent (D).

We design a simple environment in order to review our system in the real world. The connection between two peers may change over time. [Figure 4.10] shows one of the network topologies in the real world.

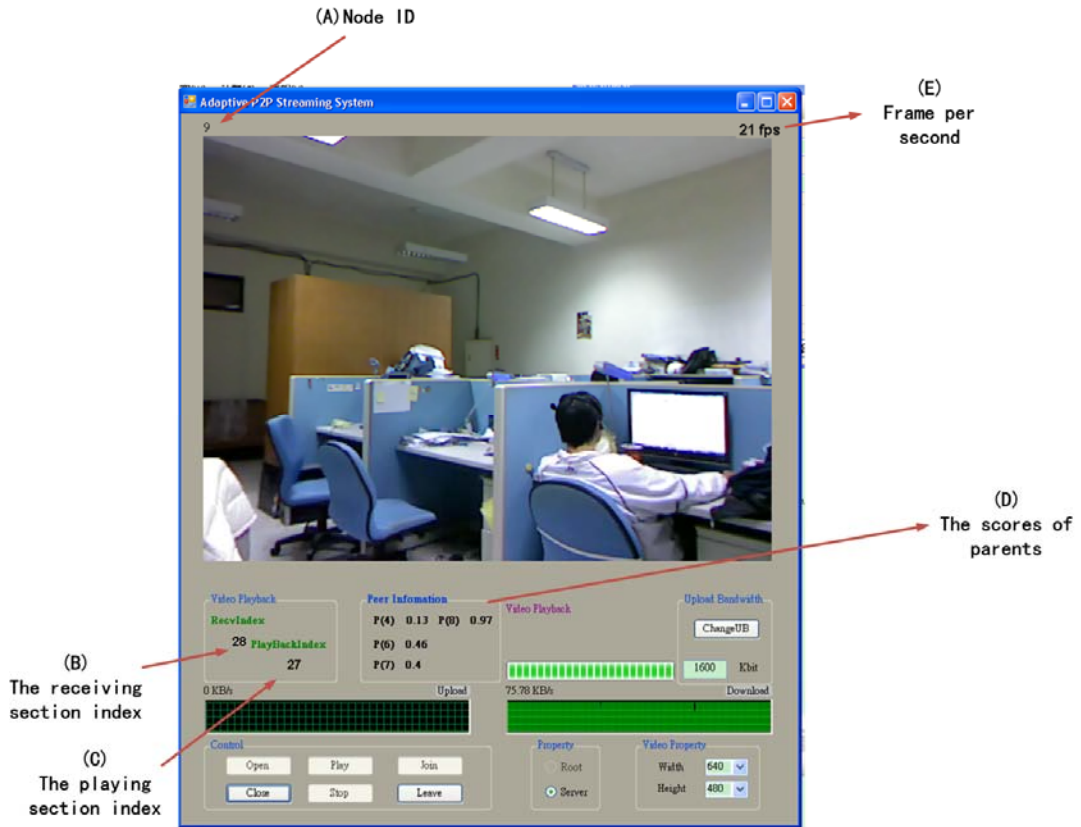


Figure 4-9 The system interface description

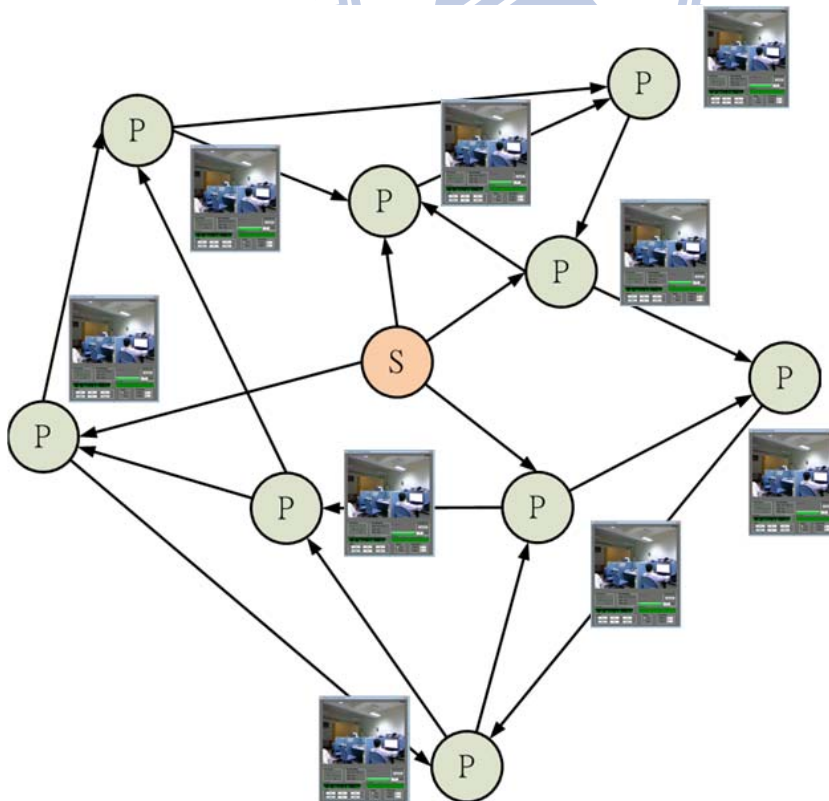


Figure 4-10 The network topology in real world

Peer upload/download capacity	1024 Kbps
Source upload capacity	4096 Kbps
LT block size	1024 Bytes
LT section size	80 K Bytes
Target bit rate	800 Kbps
Buffer size	2 sections
Simulation time	900 seconds
Total peers	10

Table 4-2 The parameter setting

At this experiment, we want to test the stability of our system. Therefore, we reduce half of the uploading bandwidth for one third of peers at 430s.

[Figure 4.11] shows the average effective throughput of all peers in the system. The interval in the horizontal axis represents 10 seconds. The blue line represents the effective throughput and the red line represents the goodput. We can clearly see that each peer has a stable throughput before 430s. When we reduced some peers' uploading capacity, these peers' children might suffer from insufficient download bandwidth. Nevertheless, these peers in our system adjust the scores of their parents dynamically and try to find better peers to be their parents. That is why the throughput becomes stable again after a period of time.

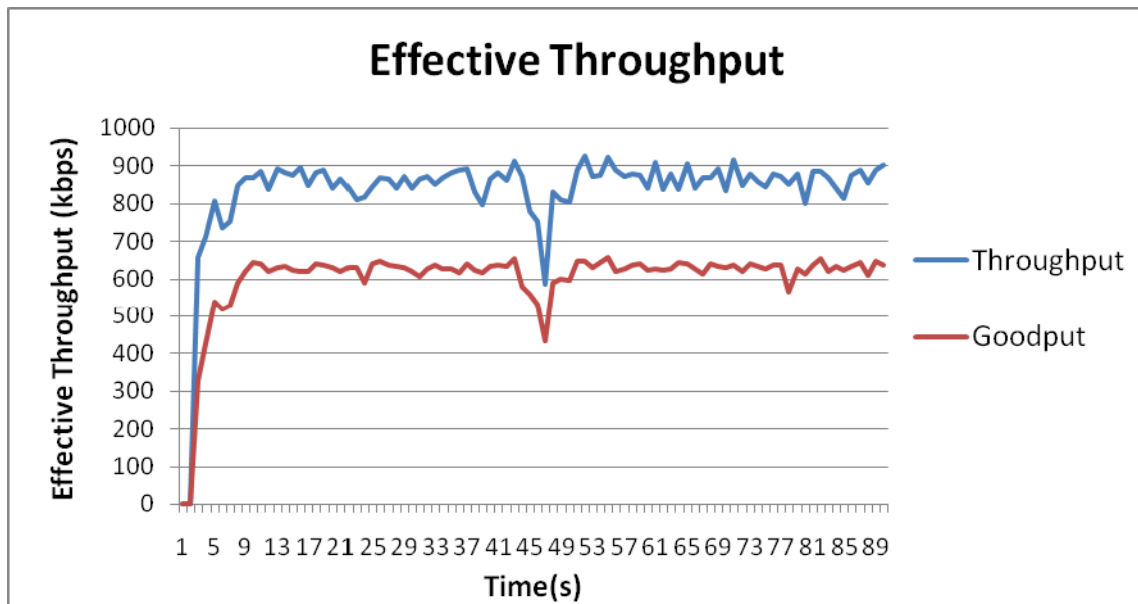


Figure 4-11 The average throughput

[Figure 4.12] shows the average source delay (section) of all peers in the system. The blue line represents the decoding index and the red line represents the playback index. We can see the difference between the two lines keeps in two sections. That is because we set the initial buffer size to be two sections. On the other hand, it means each peer in the system has a stable frame rate and source delay. Similarly, we have a higher source delay at 430s and it becomes stable after a period of time.

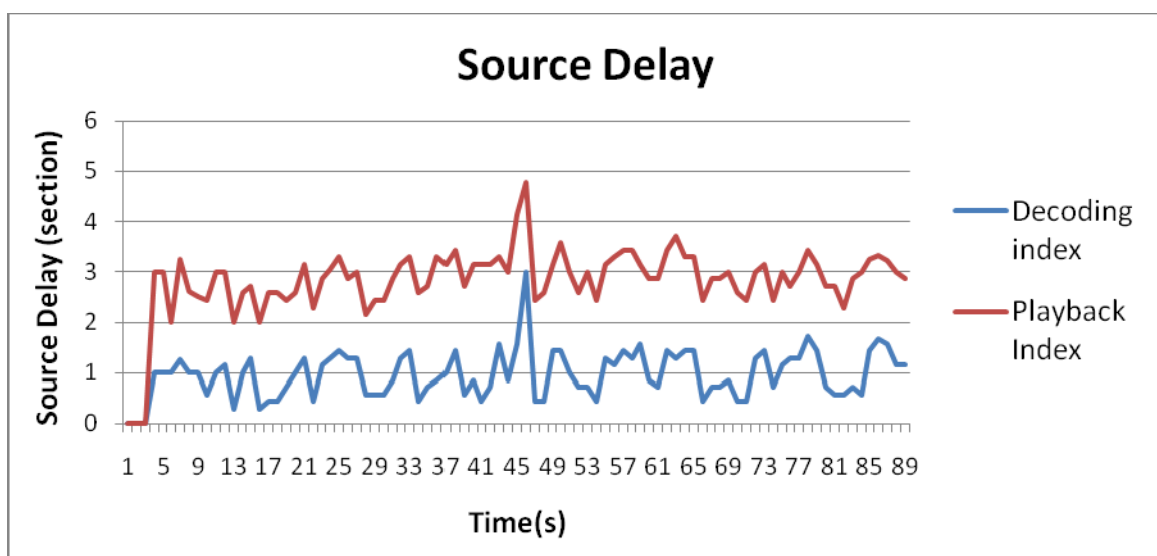


Figure 4-12 The average source delay

[Figure 4.13] shows the PSNR for each decoded frame. Due to the space limit, we list about 750 frames. For twenty continuous P-frame, we insert an I-frame. Therefore, the peak values represent the PSNR for the I-frame. And the average PSNR is 37.1309.

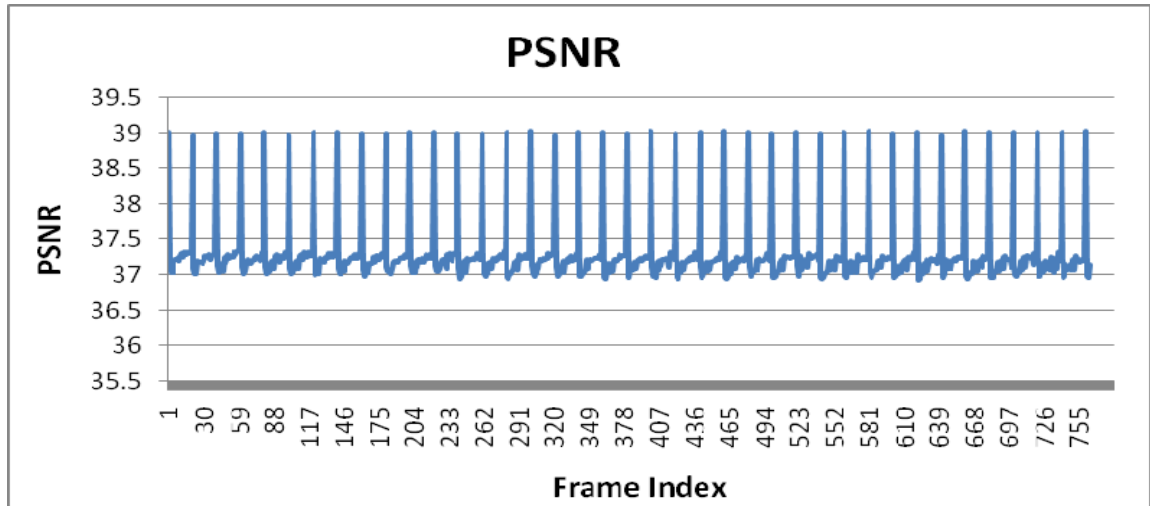
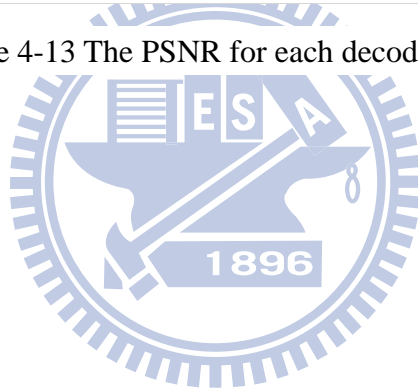


Figure 4-13 The PSNR for each decoded frame



Chapter 5. Conclusions

5.1 Conclusions

In this paper, we attempt to build a real time peer-to-peer streaming system. Referring to [9], we modify the method [9] and implement it into the peer-to-peer streaming system. At first, we choose the mesh-based architecture to construct the peer-to-peer topology. Next, we use VCM9 for our video compression and LT codes for reducing the data coordination packets. Finally, we have to establish the system protocol. The system protocol contains the connecting protocol, data protocol and peer management. The connecting protocol is for the new coming peers joining our streaming system. The data protocol teaches the peer how to allocate self downloading and uploading bandwidth and ask what section of data. The peer management protocol is to manage the peer pool and update the parents' score. We add the jitter parameter to parents' score evaluation. The simulated experiment under background traffic shows the score evaluation considering with jitter is good for the proposed system. We believe our peer-to-peer streaming system is suitable for the real world.

Reference

- [1] Jagadish HV, Ooi BC, Vu QH BATON: A Balanced tree structure for peer-to-peer networks. In: Proceedings of the international conference on very large data bases 2005, Trondheim, Norway. p. 661–72.
- [2] Hung-Chang Hsiao and Chih-Peng He, "A Tree based Peer-to-Peer Network with Quality Guarantees", IEEE Trans. on Parallel and Distributed Systems, Vol. 19, No. 8, pp. 1099-1110, Aug. 2008.
- [3] N.Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in Proc. IEEE INFOCOM, 2007, pp. 1415–1423
- [4] X. Zhang, J. Liu, B. Li, and T.P. Yum, "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming," in INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, 2005, pp. 2102-2111 vol. 3.
- [5] D. A. Tran, K. A. Hua, and T. Do, "ZIGZAG: an efficient peer-to-peer scheme for media streaming," in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE, 2003, pp. 1283-1292 vol.2.
- [6] Y.-H. G. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in Proc. ACM SIGMETRICS, 2000, pp. 1–12.
- [7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in Proc. ACM SIGCOMM, Oct. 2002, pp.205–217.
- [8] V. Pai et al., "Chainsaw: Eliminating trees from overlay multicast," in Proc. IPTPS, Feb. 2005.
- [9] Chih-Wei Fan-Chiang and Hsu-Feng Hsiao , "Dynamic and Resilient Peer-to-Peer Architecture for Live Streaming", NCTU, master's degree, 2009
- [10] Windows Media Codecs :
<http://www.microsoft.com/windows/windowsmedia/forpros/codecs/video.aspx>
- [11] M. Luby, "LT codes," in Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on, 2002, pp. 271-280.
- [12] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in Proceedings of the 4th ACM SIGCOMM conference on Internet measurement Taormina, Sicily, Italy: ACM, 2004.
- [13] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," SIGCOMM Comput. Commun. Rev., vol. 34, pp. 107-120, 2004.
- [14] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A transport protocol for real-time applications", RFC1889, January 1996.

- [15] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," ACM Computer Communication Review, Vol. 27, n. 2, Apr. 97, pp 24-36
- [16] A. Shokrollahi, "Raptor codes," Digital Fountain, Inc., Tech. Rep. DF2003-06-001, June 2003.
- [17] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng, "AnySee: Peer-To-Peer live streaming," in Proc. IEEE INFOCOM, Apr. 2006.
- [18] Sopcast : <http://www.sopcast.com/>
- [19] PPStream : <http://www.ppstream.com/>
- [20] TvuNetworks : <http://www.tvunetworks.com/>
- [21] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," Peer-to-Peer Networking and Applications, vol. 1, no. 1, pp.18–28, March 2008.
- [22] G. Huang, "PPLive – a practical P2P live systems with huge amount of users" (keynote), P2P streaming and IPTV workshop (P2P-TV), Kyoto, Japan , Aug. 2007
- [23] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," Multimedia, IEEE Transactions on, vol. 9, pp. 1672-1687, 2007.
- [24] A. Horvath, M. Telek, D. Rossi, P. Veglia, D. Ciullo, M. A. Garcia, E. Leonardi, M. Mellia, "Dissecting PPLive, SopCast, TVAnts", submitted to ACM Conext, 2008.
- [25] Derek Leonard, Vivek Rai, and Dmitri Loguinov, "On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks," SIGMETRICS Perform. Eval. Rev., vol. 33, pp. 26-37, 2005.
- [26] NF Huang, YJ Tzang, HY Chang, "Construction of an efficient ring-tree-based Peer-to-Peer streaming platform", Networked Computing and Advanced Information Management (NCM), 2010.