A NETWORK CODING EQUIVALENT CONTENT DISTRIBUTION SCHEME FOR P2P-VoD STREAMING

A PROJECT REPORT

Submitted by

L. KARTHIK CHANDRAN (30409104040)
M. KARTHIK (30409104041)
KARTHIKA SINDHU (30409104042)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

EASWARI ENGINEERING COLLEGE, CHENNAI

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2013

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**A NETWORK CODING EQUIVALENT CONTENT DISTRIBUTION SCHEME FOR P2P-VoD STREAMING**" is the bonafide work of "**L. KARTHIK CHANDRAN (30409104040), M. KARTHIK (30409104041), KARTHIKA SINDHU (30409104042)**" who carried out the project work under my supervision.

SIGNATURE

Prof.S.Kayalvizhi M.E., (Ph.D)
HEAD OF THE DEPARTMENT

Department of Computer Science
and Engineering

Easwari Engineering College
Bharathi Salai,
Ramapuram,
Chennai - 600 089

SIGNATURE

Mrs.N.Senthamarai M.E., (Ph.D)
SUPERVISOR

Department of Computer Science
and Engineering

Easwari Engineering College
Bharathi Salai,
Ramapuram,
Chennai - 600 089

CERTIFICATE OF EVALUATION

College Name : Easwari Engineering college

Branch & Semester : Computer Science and Engineering & VIII

S.No.	Name of the students who have done the project	Title of the project	Name of the supervisor with Designation
1. 2. 3.	L. KARTHIK CHANDRAN (30409104040) M. KARTHIK (30409104041) KARTHIKA SINDHU (30409104042)	A NETWORK CODING EQUIVALENT CONTENT DISTRIBUTION SCHEME FOR P2P-VoD STREAMING	Mrs. N. Senthamarai, M.E., (Ph.D), Assistant Professor (Sr. Gr.)

The report of the Project work submitted by the above students in partial fulfilment for the award of Bachelor of Engineering degree in Computer Science and Engineering of Anna University were evaluated and confirmed to be reports of the work done by the above students and then evaluated.

The viva-voce examination of the project work was held on _____.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

Streaming of large content, like videos are hampered by several constraints like delays, losses and more. Peer-to-peer (P2P) based technology for video streaming improves on the process, but still is plagued by delays during startup and seeking. In this paper, these delays are reduced using a network coding based algorithm where a random coding coefficient is used for each block of the video. It helps reduce the startup time by not requiring the peers to search for specific parents in the case of peer departure. It uses a batching and patching system to efficiently serve the video along with cache and relay technique. The peers do not use an index, instead uses information exchange and mining those info in order to reduce the delay caused by looking for new segments of data more efficiently. Interactivity performance in video is enhanced and lesser block loss rates are achieved. The process also uses interleaving distribution technique in order to effectively reduce the jump delay to zero. Hence, general streaming and seeking performance in P2P based Video-on-demand systems are vastly improved.

ACKNOWLEDGEMENT

We hereby place our deep sense of gratitude to our beloved Founder Chairman, **Dr. T. R. Pachamuthu B.Sc., M.I.E.,** for providing us with the requisite infrastructure throughout the course.

We would also like to express our gratefulness towards our Chairman, **Dr. R. Sivakumar M.D., Ph.D,** for giving support and facilities for our institution.

We convey our sincere thanks to our Principal, **Dr. Jothi Mohan Balasubramaniam M.E., Ph.D,** Easwari Engineering College for his interest and support.

We take the opportunity to extend our hearty thanks to our Vice Principal, **Dr. K. Kathiravan M.Tech., Ph.D,** Easwari Engineering College, for his constant encouragement.

We take the privilege to extend our hearty thanks to the Head of the Department, **Prof. S. Kayalvizhi M.E., (Ph.D),** for her suggestions, support and encouragement towards the completion of the project with perfection.

We would like to express our sincere gratitude to our project coordinator, **Dr. K. M. Anandkumar M.Tech., Ph.D,** Department of Computer Science and Engineering for his constant support and encouragement.

We also thank our internal guide, **Mrs. N. Senthamarai, M.E., (Ph.D),** Department of Computer Science and Engineering for her timely guidance throughout the overall progress of the project.

Finally, we whole-heartedly thank all the faculty members of the Department of Computer Science and Engineering for warm cooperation and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
DSL	-	Dynamic Skip List
IP	-	Internet Protocol
JMF	-	Java Media Framework
NCECD	-	Network Coding Equivalent Content Distribution
P2P	-	Peer-to-Peer
RTP	-	Real-time Transport Protocol
RTCP	-	RTP Control Protocol
VCR	-	Video Cassette Recorder
VoD	-	Video on Demand

CHAPTER 1

INTRODUCTION

1.1. OVERVIEW

Multimedia includes a combination of text, audio, still images, animation, video, and interactivity content forms. Multimedia is usually recorded and played, displayed or accessed by information content processing devices, such as computerized and electronic devices, but can also be part of a live performance.

Streaming media are multimedia that are constantly received by, and normally presented to, an end-user while being delivered by a streaming provider. The name refers to the delivery method of the medium rather than to the medium itself.

The distinction is usually applied to media that are distributed over telecommunications networks, as most other delivery systems are either inherently streaming (e.g., radio, television) or inherently non-streaming (e.g., books, video cassettes, audio CDs). Internet television is a commonly streamed media.

1.2. PROBLEM DESCRIPTION

Video-on-demand (VoD) is an interactive multimedia service, which delivers video content to the users on demand. Due to the frequent trick modes from the users such as play, pause, fast forward, fast search, reverse search and rewind, existing approaches either present long latencies on the user side or incur excessive stress on the server side.

1.3. OBJECTIVE

In this project, the focus is on the single-video VoD process, where a peer only transfers the video it is currently playing. For ease of expression, in the rest of the paper, the terms “client,” “peer,” and “node” are used interchangeably.

In order to provide streaming VoD services, stream re-use techniques such as batching, patching, and chaining are proposed. The stream reusability will be underutilized unless partnering peers keep persistent connections with each other. Consequently, user experiences are seriously degraded when they perform frequent video controls. To address this problem, prefetching is also employed. Hence, assuming network quality is ideal, the random access delays are reduced.

1.4. SCOPE OF THE PROJECT

Different strategies are adopted such as sequential, random, and global rarest strategies, but none of them addresses the content-based associations among different segments of videos. VoD users can be greatly improved with a better prefetching strategy. Bearing this in mind, an interaction oriented VoD scheme in large-scale Peer-to-peer (P2P) networks is proposed, which can be characterized as follows:

1. Video-oriented: Video inter-activities are efficiently predicted using the technique of association rule mining. Requests of Video controls are efficiently resolved locally.

2. Scalability: A combination of batching and patching is adopted here. This system is able to serve many more concurrent clients than the original capacity of the source server.

3. Interactivity: Based on the scheme of patching and the Cache and Relay strategy, This system provides abundant backup resource for asynchronous clients and frequent video control requests.

4. Short latency: This system adopts an efficient information exchange protocol and a collaborative prefetching strategy. Both the requests of joining in and the dynamic video controls can be responded with very short latencies. Also, Segmentation and encoding process help reduce startup latency.

CHAPTER 2

LITERATURE SURVEY

2.1. INTRODUCTION

There were many techniques that have been proposed previously in order to improve the efficiency of P2P based streaming. Some P2P systems using linear network coding have already been developed. But they do not include the impact of interactivity in the videos while streaming. In this project, this issue is also taken into account.

2.2. EXISTING SYSTEMS

Wang and Li [4] presented R^2, a new streaming algorithm designed to combine random network coding with a randomised push algorithm. R^2 focuses on improving the efficiency of live streaming in terms of startup buffering delays, resilience to the unpredictable behaviour of peers, and bandwidth saving of streaming servers. Wu et al. proposed some de-centralised strategies to eliminate conflicts among coexisting streaming overlays on contested bandwidth and combined those strategies with network-coding-based content distribution to realise efficient multi-overlay streaming.

Chi et al. [4] proposed a buffer-assisted search (BAS) scheme to increase partner search performance by decreasing the size of the index structure. They also designed a novel scheduling algorithm using the Deadline Aware Network Coding (DNC) to fully utilise network resources by adopting an appropriate coding window size.

Gkantsidis et al. [3] implemented a P2P content distribution system that uses network coding. They gave a detailed performance analysis of one such

P2P system to show that network coding is practical as it generates little overhead, both in terms of CPU processing and I/O activity.

Feng and Li [2] analysed and compared the performance of network-coding-based P2P streaming to that of traditional pull-based P2P streaming. Annapureddy et al. used network coding for P2P VoD services by efficiently distributing and scheduling video segments, thereby achieving high system performance.

Nguyen et al. [8] used a network coding technique to reduce duplicated storage and remove the requirement of tight synchronisation between senders. Gkantsidis and Rodriguez proposed a new scheme for the content distribution of large files using network coding. They designed and implemented a system, called Avalanche, and explored various practical issues in network coding. A major issue in any content distribution scheme is protection against malicious peers. Avalanche uses special sets of secure hash functions that support network coding operations; further, it requires very few computational resources.

Chang et al. [1] applied a linear programming approach with network coding to the evaluation of download finish times in a P2P network. They disproved the hypothesis that Min-Min scheduling yields the minimum average finish time for routing, and showed that coding can provide a robust optimal solution and better performance than routing in a dynamic network environment.

Ken Yiu et al. [7] proposed a Distributed Storage system to support User Interactivity in Peer-to-Peer Video Streaming using Distributed Hash Table (DHT) and VMesh algorithms. It has pointers to peers storing video segments far away from the current playback position that can help a user seek a far away position more efficiently, thereby reducing the seeking latency.

Dan Wang et al. [4] used multi-supplier data scheduling to achieve on-demand overlay media streaming based on Dynamic Skip Lists (DSL). But, the

layers are not well-balanced, leading to inaccurate jumps and hence more segment losses. Moreover, under high speeds, the quality of the system is worse.

Do, T.T. et al. [5] utilised a Join algorithm in order to allow clients to join the system faster. The parent node is selected using Round Robin Selection, Smallest Delay Selection or Smallest Distance Selection. It also helps an affected client to recover from a failure in less time. But there are high chances of segment losses and seek latency is also increased.

Some works have considered the implementation of network coding and P2P cooperative computing for wireless network and mobile devices. Dimakis et al. have introduced the notion of regenerating codes, which make a new peer only need functions of the stored data from the surviving parent peers to significantly decrease the repair bandwidth. Liu et al. [6] have implemented a P2P storage cloud for on-demand streaming called Novasky. A coding-aware peer storage and replacement strategy as well as an adaptive server push strategy are proposed to achieve storage efficiency, durable video availability, load balancing, and the balance of the demand and supply of data and bandwidth in the P2P storage cloud. However, previous works have not considered the following three issues. First, previous studies have not yet found a way in which to divide and distribute the data from one entire video to one parent peer or sender to allow any parent peer to easily replace leaving parent peers. Second, previous studies do not consider that an appropriate number of parent peers for multi-source streaming by the network coding technique must be determined to reduce the block loss rate from parent peers' departure and the dependent encoded blocks. Third, the impact of interactive activities on a P2P VoD streaming system has also not been considered. These three issues are analysed and resolved in this paper.

2.3. ISSUES IN THE EXISTING SYSTEM

- A video is divided into multiple segments and periodically broadcast to the clients through dedicated server channels.
- In order to provide uninterrupted streaming for all the users, the server keeps track of all the channels and ensures no interruptions exist between playing segments, resulting in heavy overhead, thereby increasing the search cost.
- Jump performance is poor and the video startup time is long.

2.4. PROPOSED SYSTEM

- The new system serves asynchronous peers with the combination of segmentation and encoding. Hence, with sufficient encoded blocks, the original segment can be decoded.
- The server does not need to record any buffer information of peers.
- The peer requests are resolved in a distributed manner.
- The technique of association rule mining is leveraged to find the frequent patterns of video playback inter-activities, based on which neighboring peers conduct collaborative prefetching. As a result, the seek latencies are minimized.

2.5. SUMMARY

In this section, the ways in which the P2P video transmission issues were previously dealt with have been discussed. Also, certain techniques have been proposed, by which the overall efficiency of the system can be improved.

CHAPTER 3

SYSTEM DESIGN

3.1. INTRODUCTION

The design of the system is split over the server and the peers. Each module has a specific function. The server-side has the functionality of segmenting and encoding the videos whereas each peer has the ability to decode, cache and play the video.

The functions can be depicted as follows :

- Segmentation and Encoding
- Decoding
- Cache and Relay
- Information exchange
- Playback

3.2. SYSTEM ARCHITECTURE

The system consists of a main server where the initial video is stored. Several peers can connect to the server at the same time in order to receive the video. The nodes are connected in a LAN. Each peer can transmit data with one another.

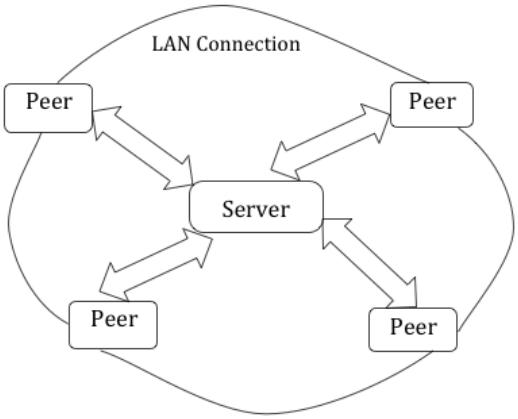

Figure : 3.1. System Architecture

3.3. FUNCTIONAL ARCHITECTURE

The peer initially requests for the video hosted by the server. The server, on incoming request from the peer, segments the video, encodes it using a random co-efficient vector and distributes it to the peers. The peer, after decoding the video, places it on a buffer. After sufficient blocks are available, the video starts playing to the user. The other blocks that are cached are relayed to other peers when required.

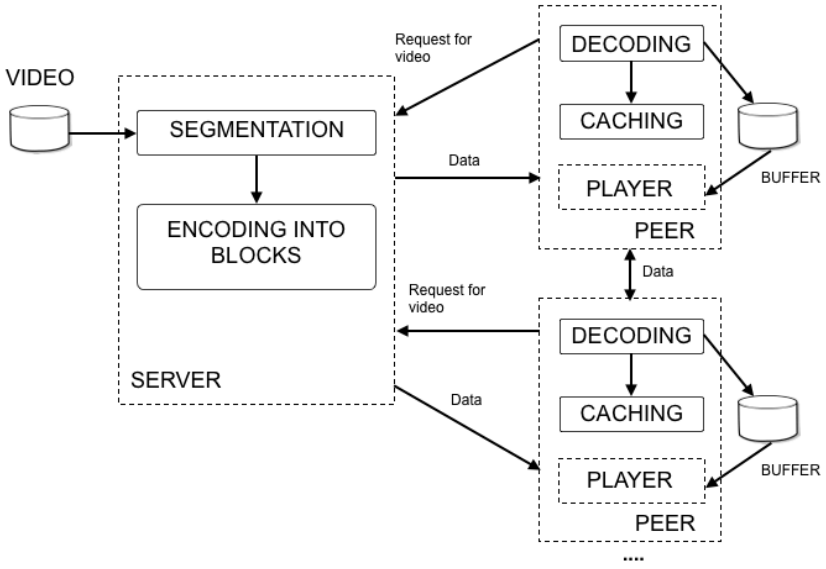

Figure : 3.2. Functional Architecture

3.4. MODULAR DESIGN

Modular design defines structure of the overall module. Modularity is a general system concept typically defined as a degree to which a system's components may be separated and recombined.

3.4.1. SEGMENTATION AND ENCODING

The server stores the video to be distributed in it's secondary storage. The server, on request from a peer, starts distributing the video over the data connection. The encoded segments are then sent over to the peers for playback.

The server uses batching to serve asynchronous peers. The server allocates a certain amount of dedicated outgoing bandwidth for each batching session. In each session, early joining peers directly become the children of the server.

After the allocated bandwidth is fully occupied, late peers are redirected by the server and become the descendants of the early ones. Since the peers in a session transfer same video content currently broadcast by the server, the streaming mechanism inside a batching session is similar with P2P live streaming.

Moreover, it reinforces the batching scheme with patching. The server sends a list of randomly selected peers to each joining peer.

When a peer joins in a session late and misses the initial part of the video, it picks up a few peers from the random list as patching sources and immediately starts to download the missing part from them.

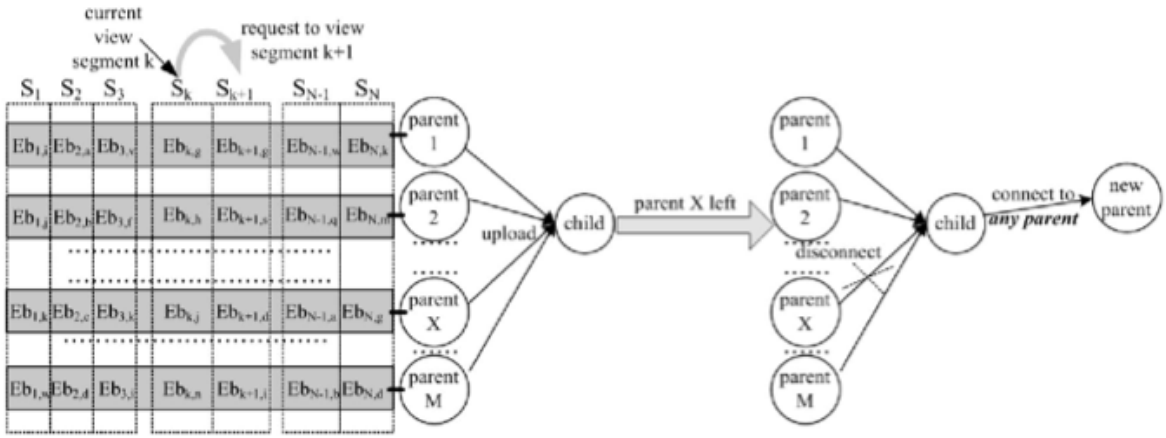

Figure : 3.3 NCECD Scheme

Algorithm for Encoding

Input : Video

1. Divide video into X segments (X1,X2…Xn)

2. Further, divide each segment into blocks b using Interleaving scheme (bX,1 ; bX,2… bX,M)

3. Encode the blocks using a random co-efficient vector f(i) = (ci,1 ; ci,2… ci,M) using NCECD scheme

Output : Encoded blocks (EbX,1 ; EbX,2… EbX,M), where EbX,i = $\sum_{j=1}^{M} c_{i,j} \cdot b_{X,j}$.

Original blocks of segment X = (bX,1 ; bX,2… bX,M) is recovered using

$X = \mathrm{F}^{-1}\mathrm{E}^{\mathrm{T}}$.

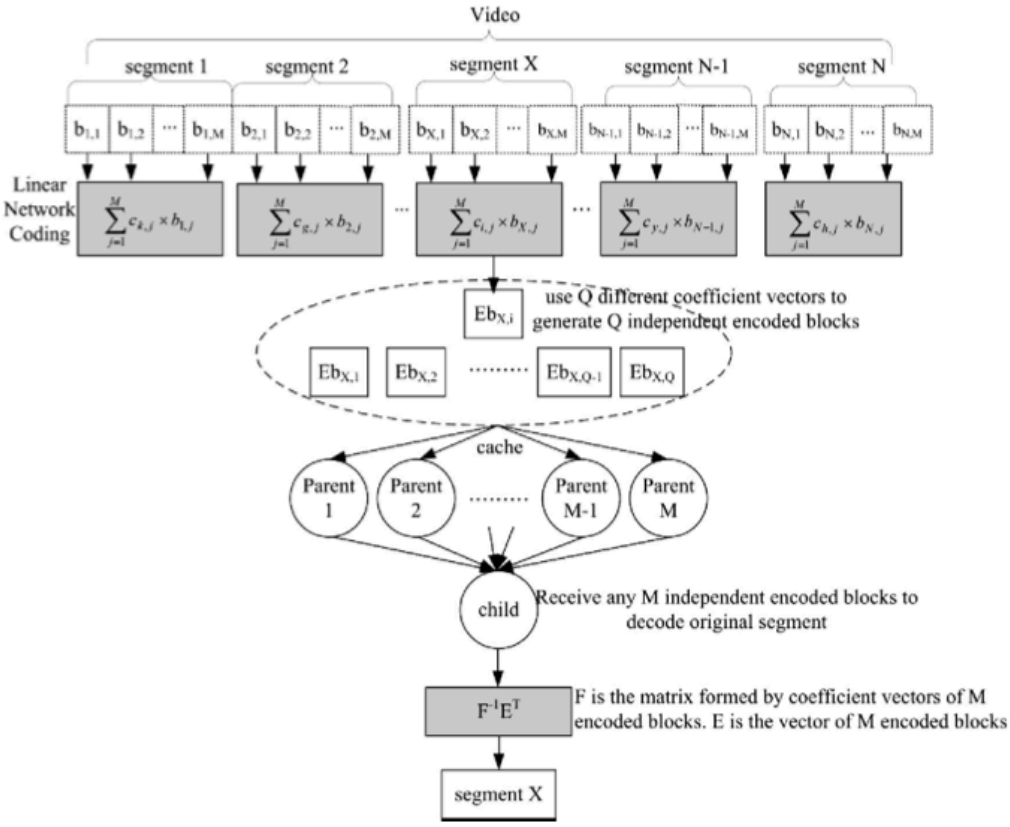

Figure : 3.4. Encoding Process

3.4.2. DECODING

Peers are clustered according to their arrival times and form sessions. Each session, together with the server, constructs an application multicast tree. Later, peers can retrieve the missing parts from the server or other peers.

VoD divides the peers into generations according to their requests. Peers in VoD always cache the most recent content of a video. Only one stream from an early peer is needed to serve a late peer in VoD, while two streams, a patching, and a base stream are necessary for serving a late peer.

Instead of deploying a patch server as failures and dynamics are handled locally in VoD, so that the server stress is further reduced.

3.4.3. CACHE AND RELAY

To support asynchronous accesses to the video content. When the server starts a new batching session, it need not make any late peers wait.

The early peers in a batching session obtain the video content and become the substitute video sources. Late peers can make up the missing content using patching from the early ones. Meanwhile, patching improves the system flexibility. This system provides abundant backup stream sources for patching by adopting the Cache and Relay strategy, where all the peers keep both the initial 5 minutes and the latest 5 minutes of the video played.

Any late peer can instantly find patching sources and make up the missing part immediately after join, no matter if it starts from the beginning or any other offset of the video. Because the patching sources are selected randomly, load balance is kept among the peers.

3.4.4. INFORMATION EXCHANGE

Peers conduct periodical Information exchanges to exchange their state information. During each period, a peer generates a state message, including its latest state information.

The format of the state message is {IP, Incremental playback record, Time stamp}, where IP is the peer's IP address, Incremental playback record refers to the string of segments the peer plays after it generates last state-message last time, Time stamp is the time since the peer joins in.

On the other hand, each peer maintains a list of records. Each entry in the list corresponds to a peer and records its latest state. On receiving a state message, a peer performs relevant operations before it forwards the message to its neighbors: If the time stamp of the state message is greater than that in the entry, the incremental playback record is inserted into the tail of the playback record in the entry, and the time stamp in the entry is updated.

Using this, a peer is able to accumulate the information of playback history of all the peers. Furthermore, through periodical information exchange every peer can keep aware of the global distribution of video data on all the peers.

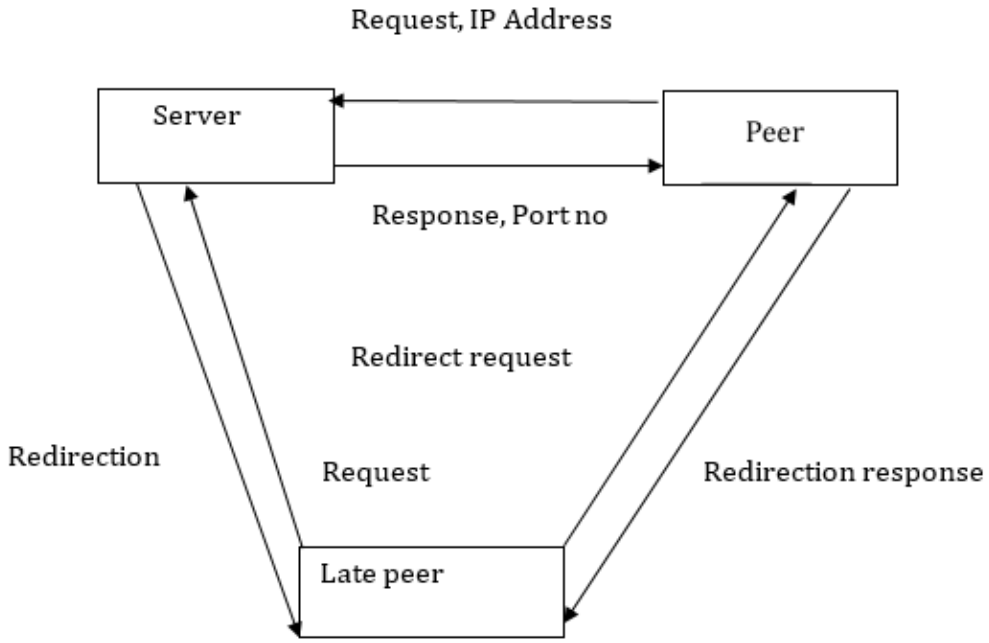

Figure : 3.5. Information Exchange Process

3.4.5. PLAYBACK

Here, we consider every minute of video as a segment. A peer maintains its playback record while streaming and playing the video. The playback record is a string of segment indices, which is initially empty.

When a segment is played, its index is inserted into the tail of the string.For example, suppose the current playback record of a peer is (1, 2, 3,4, 5, 6, 7, 8, 11, 12, 13, 14, 15, 7, 8), it depicts a playback history as follows:

The peer first plays the video from the first to the eighth minute, fast searches to the 11th minute, plays until the 15th minute.

The reverse searches to the seventh minute, and plays the eighth minute before the playback record is last updated.

3.4.5.1 Prefetching

Peers take the state information collected as the input of mining. The goal of mining is to find the segments most associated to the segment currently played. First, it is observed that the user behavior in the next few minutes is closely related with his/her experience during the last few minutes.

The preconfigured sizes of item sets in the association rules have apparent impact on the efficiency and accuracy of mining. Thus, these rules are mined

$$\mathbf{t1 \rightarrow t2, \text{ where } t1 \cap t2 = \Phi, \quad | t1 | = | t2 | = 3}$$

Second, the playback history of a VoD user actually forms a sequence of segments. Input and output of predictions based on the history information ought to be order sensitive.

For example, a user who plays the segments (4, 5, 1) will probably continue to watch the second and the third segments, while a user who plays the segments (1, 4, 5) will probably go on with the sixth segment.

Third, based on association rule mining, it is found that all the association rules that have a support and a confidence greater than the specified thresholds. Setting the thresholds of support and confidence helps to avoid the impact of random coincidence and improves the precision and accuracy of mining.

For a particular peer A, its playback history in the last 3 minutes is denoted as an ordinal string (a1, a2, a3). Let L be the local list of records kept by peer A.

Algorithm for prefetching

Peer A extracts all those segments which simultaneously satisfy the following requirements:

Input : Segment information

1. From each record in the list, three segments (if exist) at most are extracted.
2. They do not contain the immediate next two segments after the current progress of playback, because these segments are downloaded through urgent downloading.
3. They are the closest segments following segment a3 in the record.

Output : Pruned Segment information

3.5. SUMMARY

Thus, the system can be designed using the above techniques. The modules are implemented and multiple nodes can be connected over a LAN connection or can be run on a local system setup for demonstration purposes.

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1. INTRODUCTION

Implementation is the stage in the project where the theoretical design is turned into a working system. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the change over an evaluation of change over methods.

4.2. OVERVIEW OF THE PLATFORM

HARDWARE REQUIREMENTS

Processor	:	Intel Core i5 or higher
Hard Disk	:	40 GB or higher
Memory	:	1 GB or higher
Additional requirements	:	LAN connection

SOFTWARE REQUIREMENTS

Operating System	:	Windows 7/8
Language	:	Java (JDK 1.7)
Tools	:	NetBeans 7
Additional requirements	:	JRE, JMF package

4.2.1. INTRODUCTION TO JAVA

The Java programming language and environment is designed to solve a number of problems in modern programming practice.

Object-Oriented Programming

Object-Oriented Programming is at the core of Java. In fact, all Java programs are object-oriented—this isn't an option the way that it is in C++.

Abstraction

A powerful way to manage abstraction is through the use of hierarchical classifications.

Encapsulation

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

Inheritance

Inheritance is the process by which one object acquires the properties of another object.

Polymorphism

The concept of polymorphism is often expressed by the phrase "one interface, multiple methods."

4.2.2. JMF

The Java Media Framework (JMF) provides the object-oriented programming tools necessary to facilitate the construction of software, which delivers this content to the end-viewer.

It provides functionality for capturing, processing and viewing time-based media. By abstracting these elements through the JMF the programmer can construct a program that will not only have better functionality but will also be crossplatform compatible.

JMF Application Programmer's Interface (JMF API) utilizes four managers, which make it easier to create new interfaces with less modification to existing code. The four managers are as follows:

- **Manager :** coordinates construction of Players, Processors, DataSources and DataSinks.
- **PackageManager :** contains a listing of packages that utilize JMF classes. This includes custom Players, Processors, DataSources and DataSinks.
- **CaptureDeviceManager :** contains a listing of available capture devices.
- **PlugInManager :** contains a listing of available plug-ins such as Multiplexers, Demultiplexers, Codecs, Effects and Renderers.
- **Demultiplexer :** extracts multiple tracks from data streams, which were multiplexed when created.
- **Effect :** handles processing of special effects on a data stream.
- **Codec :** these handle the encoding and decoding of content types.
- **Multiplexer :** combines multiple audio/video tracks into a single stream for delivery.
- **Renderer :** processes data in a track and outputs it to an output device such as a screen or speaker.

When designing a media player, the first step in utilizing the JMF is to acquire a data source.

The difference between using a Player or Processor is in the format of the captured data. If the DataSource is to be played than a Player is used.The filename is generally a URL object.

Player mediaPlayer = Manager.createPlayer(filename);

By using a Processor the programmer can, for example, capture audio from a microphone, encode it in the MP3 file format, and write it to disk for later access.The JMF can be extended through the use of plug-ins by doing additional processing on a track or through constructing new DataSources and MediaHandlers.

Multiplexers handle combined one or more tracks into a single data source, which can then in turn be turned into multiple tracks by a Demultiplexer when being prepared for playback.Once the library is downloaded and installed the plug-in is referenced by the following:

new com.sun.media.codec.audio.mp3.JavaDecoder();

This provides the functionality of the MP3 decoder to the program and will allow the Player object to decode and playback music encoded with the MP3 encoding. The Player object goes through a series of steps, before starting playback of the media stream: ‘Unrealized’, ‘Realizing’, ‘Realized’, ‘Prefetching’, ‘Prefetched’, ‘Started’.

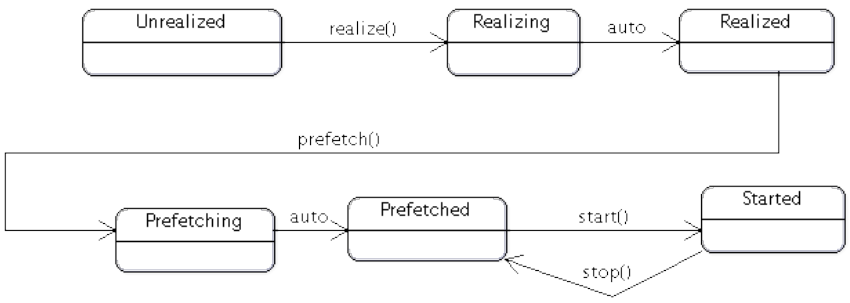

Figure : 4.1. JMF Player State Diagram

In addition to the two stages used by a Player, the Processor introduces a third stage called ‘Configuring’ during which the Processor determines if and how it will process the data being provided.

The method getControlPanelComponent can be called to get the Component, which can then be inserted, into the program’s interface.

playerControls = mediaPlayer.getControlPanelComponent();

mediaPanel.add(playerControls);

Streaming media means that the program must be able to keep up with the incoming data, prepare it, process it and deliver it to the end-viewer without dropping any of the information.

RTPControl provides support for dynamic media data and a media Format. An RTP media locator uses the following format:

rtp://address:port[:ssrc]/content-type/[ttl]

When sending streaming media content the data is received from a Processor. To open a file, the menu Player – Open … is accessed. The player filters the file list for only the three supported file types. As “mov” files can also be video.

When the file is opened, the JMF Manager creates a new Player object with the filename as the DataSource. The default controls provide the following functions:

- Play
- Pause

- Mute and volume adjustment
- Speed of playback adjustment
- Slider control over position of playback

When playback is finished the player stops. The JMF provided a very easy to learn and use functionality which made creating this media player.

4.2.3. JMF RTP

JMF enables the playback and transmission of RTP streams through the APIs defined in the javax.media.rtp, javax.media.rtp.event, and javax.media.rtp.rtcp packages.

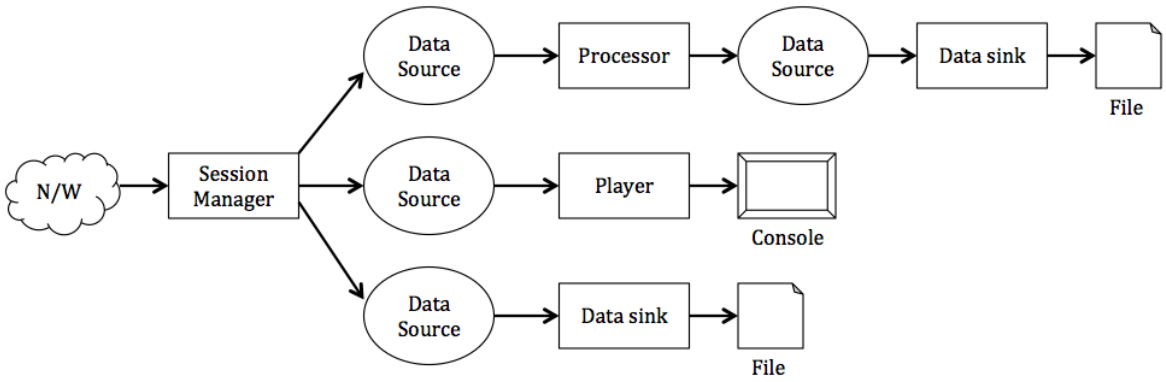

Figure : 4.2 RTP Reception

For example, the RTP APIs could be used to implement a telephony application that answers calls and records messages like an answering machine.Outgoing RTP streams can originate from a file or a capture device. The outgoing streams can also be played locally, saved to a file, or both.

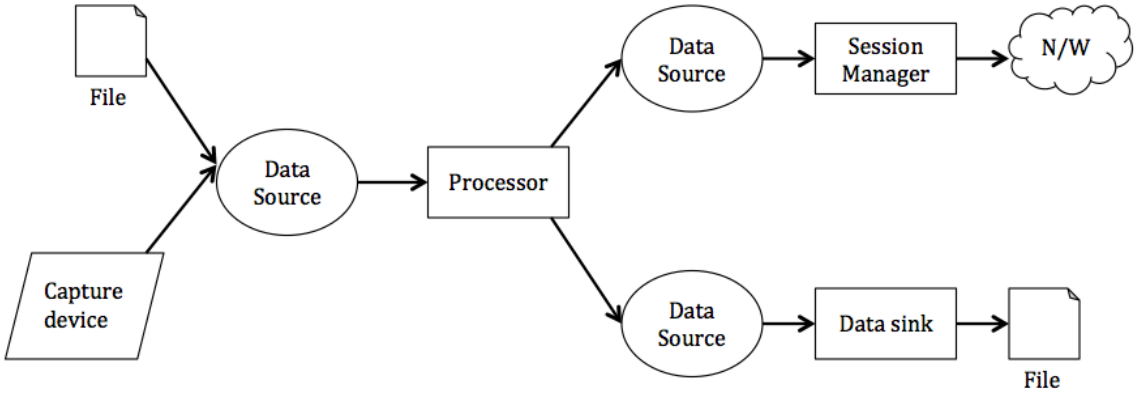

Figure : 4.3 RTP Transmission

For example, you could implement a video conferencing application that captures live audio and video and transmits it across the network using a separate RTP session for each media type.

4.2.4. RTP Architecture

The JMF RTP APIs are designed to work seamlessly with the capture, presentation, and processing capabilities of JMF.

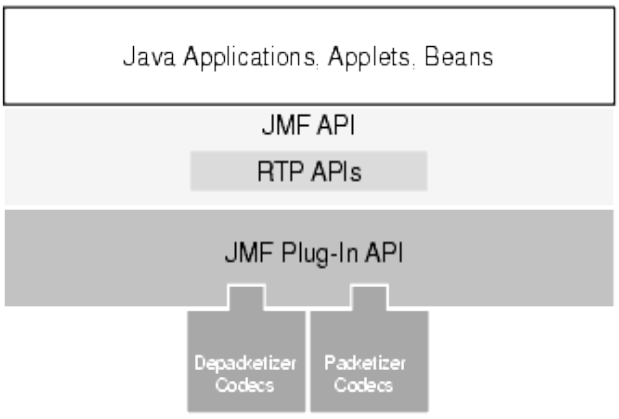

Figure : 4.4. High-level JMF RTP Architecture

4.2.4.1. Session Manager

In JMF, a Session Manager is used to coordinate an RTP session.The session manager also handles the RTCP control channel, and supports RTCP for both senders and receivers.

4.2.4.2. Session Statistics

The session manager provides access to global reception and transmission statistics:

- **GlobalReceptionStats** : Maintains global reception statistics for the session.
- **GlobalTransmissionStat**s : Maintains cumulative transmission statistics for all local senders.

4.2.4.3. Session Participants

Session Managers create a Participant whenever an RTCP packet arrives that contains a source description (SDES) with a canonical name (CNAME) that has not been seen before in the session (or has timed-out since its last use).

4.2.4.4. Session Streams

The Session Manager maintains an RTPStream object for each stream of RTP data packets in the session.

4.2.4.5. RTP Events

To receive notification of RTP events, you implement the appropriate RTP listener and register it with the session manager:

- **SessionListener**: Receives notification of changes in the state of the session.
- **SendStreamListener**: Receives notification of changes in the state of an RTP stream that's being transmitted.
- **ReceiveStreamListener**: Receives notification of changes in the state of an RTP stream that's being received.
- **RemoteListener**: Receives notification of events or RTP control messages received from a remote participant.

4.2.4.6. Session Listener

To receive notification about events that pertains to the RTP session as a whole, such as the addition of new participants.

4.2.4.7. Send Stream Listener

You can implement SendStreamListener to receive notification whenever:

- New send streams are created by the local participant.
- The transfer of data from the DataSource used to create the send stream has started or stopped.
- The send stream's format or payload changes.

4.2.4.8. Receive Stream Listener

There are seven types of events associated with a Receive Stream:

- **NewReceiveStreamEvent :** Indicates that the session manager has created a new receive stream for a newly-detected source.
- **ActiveReceiveStreamEvent :** Indicates that the transfer of data has started.

- **InactiveReceiveStreamEvent :** Indicates that the transfer of data has stopped.
- **TimeoutEvent :** Indicates that the data transfer has timed out.
- **RemotePayloadChangeEvent :** Indicates that the format or payload of the receive stream has changed.
- **StreamMappedEvent :** Indicates that a previously orphaned receive stream has been associated with a participant.
- **ApplicationEvent :** Indicates that an RTCP app packet has been received.

4.2.4.9. Remote Listener

To receive notification of events or RTP control messages received from remote participants and to monitor the session - it enables you to receive RTCP reports and monitor the quality of the session reception without having to receive data or information on each stream.

4.3. IMPLEMENTATION DETAILS

4.3.1. SERVER SIDE MODULE

The initial step is to setup the server (Fig 4.5). The server module is run first by performing a right-click on the Server file and clicking on 'Run File'. Here the video to be hosted is chosen first (Fig 4.6). The video format supported here is those with extension .mov. Then, the Port number through which transmission must take place is also specified. The Port number must be chosen in such a way that it doesn't interfere with other applications.

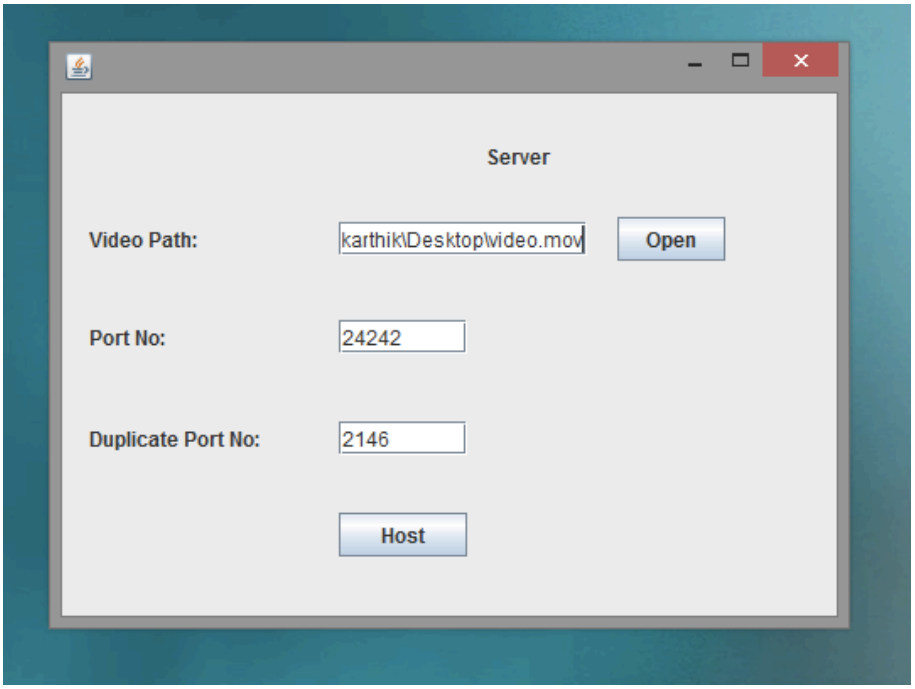

Figure : 4.5. Video being hosted on the server

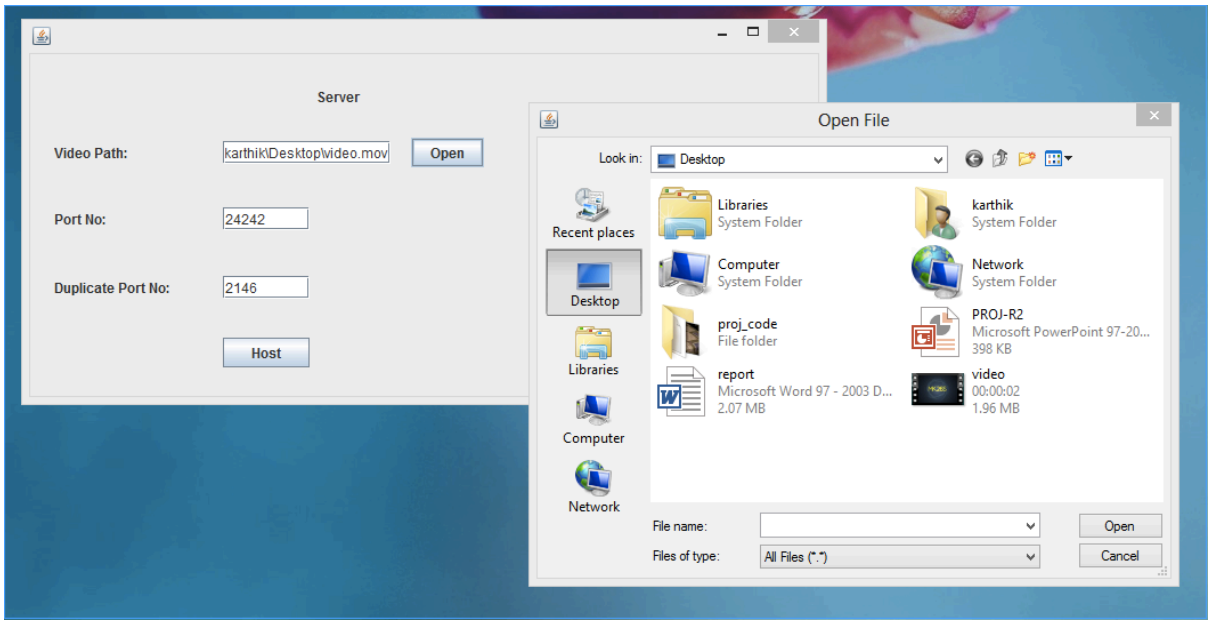

Figure : 4.6. Choosing a video to host

After the 'Host' button is clicked, the server is ready to accept incoming peers through the specified port number (Fig 4.7).

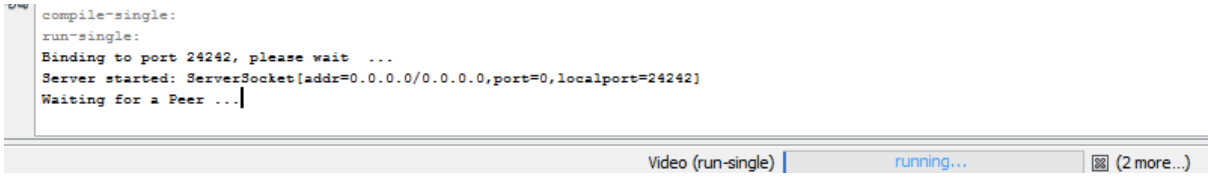

Figure : 4.7. Server ready to accept incoming connections

4.3.2. CLIENT SIDE MODULE

The client can be made to retrieve video from the server using the specified IP address. The location of where the incoming media must be saved also needs to be specified. Here, we use a localhost connection with IP 127.0.0.1 to demonstrate the working of the module. (Fig 4.8)

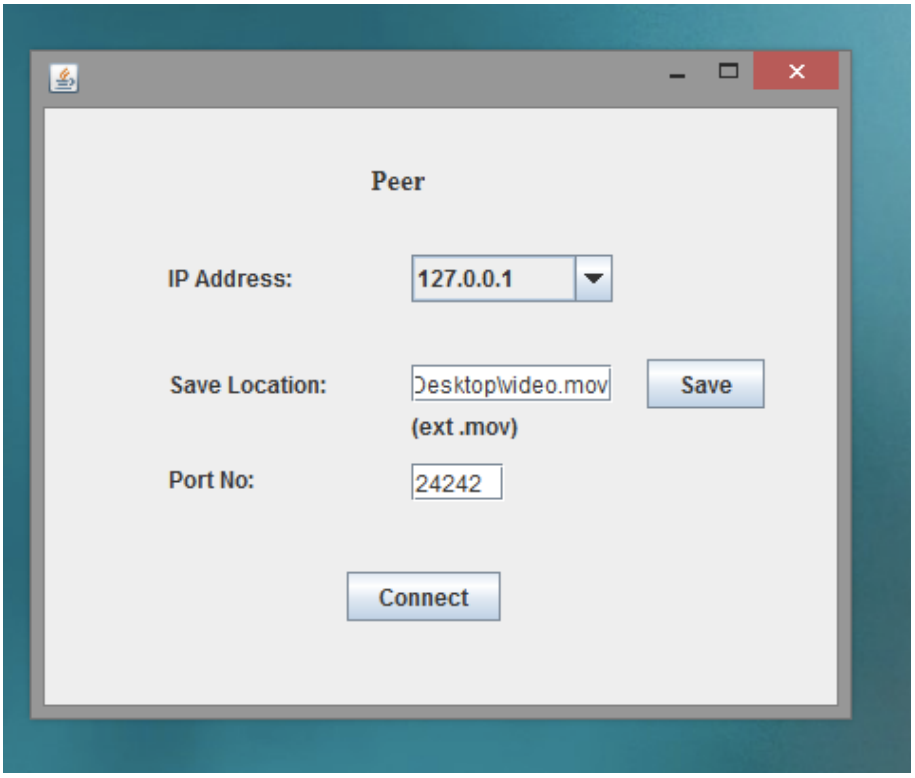

Figure : 4.8. Choosing an IP to connect to

Then, we need to choose where the media file would be saved so that it can play once the required segements are available (Fig 4.9).

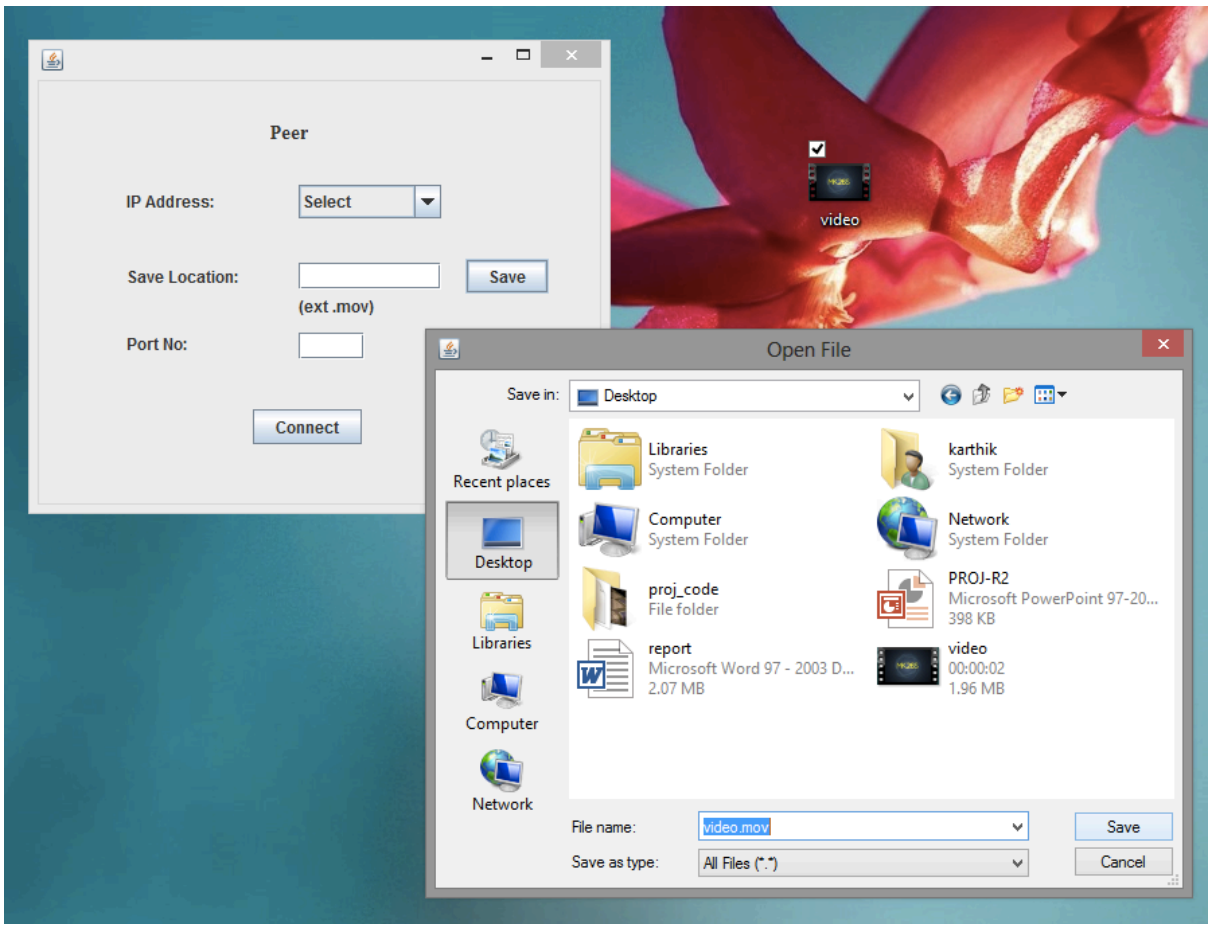

Figure : 4.9. Choosing a location for saving the media

At this point, the client starts receiving the video file from the specified server. The process is depicted in the console as shown below (Fig 4.10).

Figure : 4.10. Client receiving the Video

Finally, the video playback begins in a new window (Fig 4.11). The playback can be paused or stopped at any time and also trick modes are supported so that the user can perform seeking in the video.

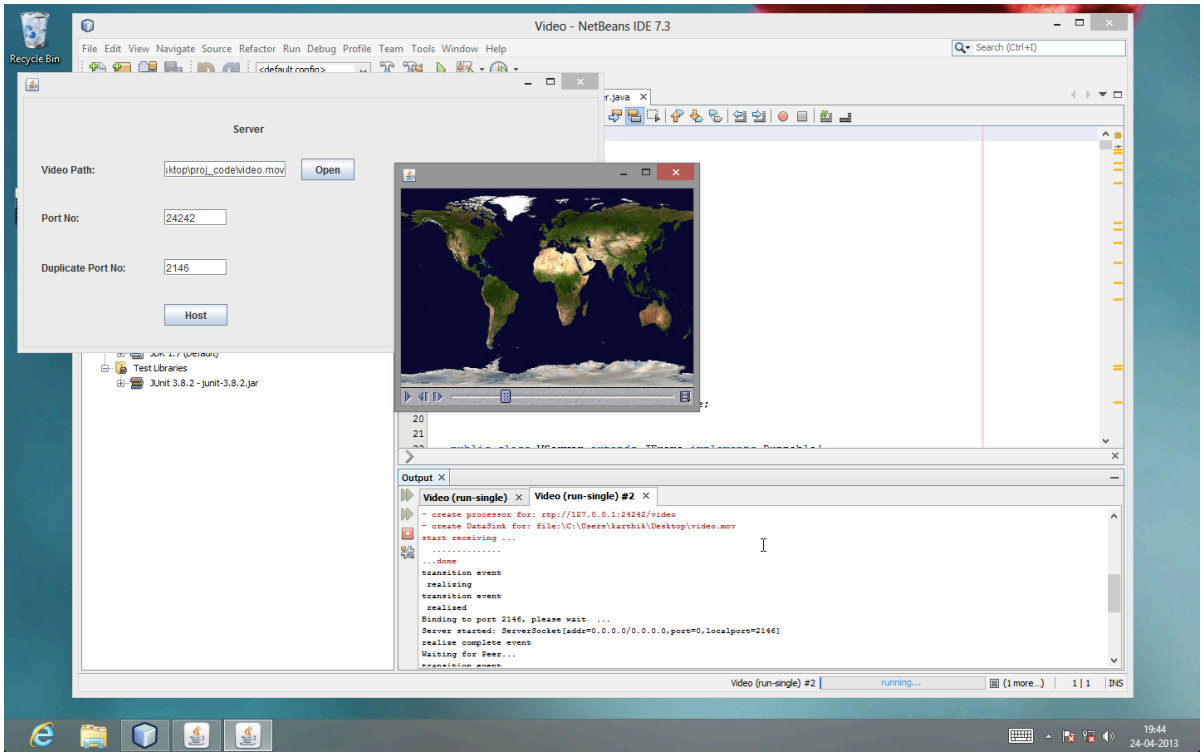

Figure : 4.11. Video playback begins

If any late peer arrives, it will receive the content from an existing peer instead of the server. This process is accompanied by information exchange where the peers transmit their state information in order to provide for the other peer.

4.4. SUMMARY

This chapter gives an overview of the platform used for building the project. It provides the implementation details with initial setup and working of each module with snapshots.

CHAPTER 5

RESULT AND PERFORMANCE ANALYSIS

5.1. INTRODUCTION

Sometimes referred to as profiling, performance analysis as it relates to software engineering is simply the process of evaluating how a particular software program is functioning. This process normally begins with how the program loads and what happens when each step in using the program is executed. The objective of performance analysis is to ensure the software program is working at optimum efficiency and to identify and correct any issues that may negatively impact that efficiency.

5.2. TEST CASES

Here, for testing purposes, a 11-second sample video is used which is in MOV format with a resolution of 352x240 with a bit-rate of 1.23 Mbps and a frame rate of 15fps. The bit rate, resolution and frame rate varies according to the video file used. We obtain an encoded video file as an output which is then decoded for playback on the peer.

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application.

- The first module has been tested to find whether the video was properly segmented and encoded.
- The second module has been tested to ensure proper decoding and playback of the video. Also caching and relaying process has been verified.

Integration tests are designed to test integrated software components to determine if they actually run as one program. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Hence, here the proper connection between the server and the peers were verified using LAN connectivity and also via a localhost setup.

5.3. PERFORMANCE MEASURES

1. Startup Delay - The duration between the time that a client arrives to that when the client connects to the parent peers required to begin viewing the video. This metric focuses on the delay in finding the required parent peers at the startup stage, and the delay of buffering several seconds (e.g., one data segment) of media data.

2. Jump delay - The duration between the time that a client requests a jump operation to the time that the client connects to the required parent peers (and begins viewing the video). This metric focuses on the delay in finding the required parent peers at the jump stage, and thus, the delay of buffering several seconds (e.g., one data segment) of media data. A lower jump delay indicates that the system can more efficiently support jump operations.

3. Media size - There is also reduction in the size of the media file due to the encoding that takes place. This measure indicates decreased transmission times while maintaining similar video quality.

5.4. PERFORMANCE ANALYSIS

Using NCECD scheme, according to the paper [1], the following complexity is obtained compared to other schemes such as VMesh and DSL. The average peer population was kept constant between these tests.

1. Startup time - O(1)
2. Jump time - 0

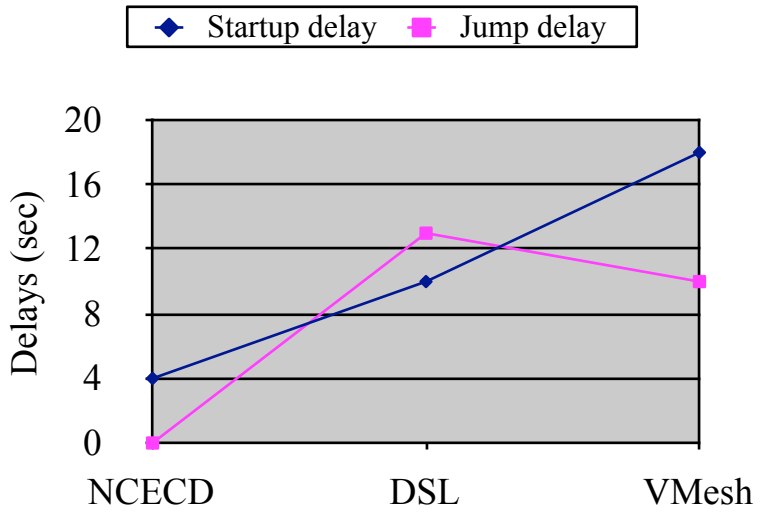

Figure : 5.1. Startup and Jump delay comparison

Since an encoding technique is used, the file size also is reduced anywhere from 4%-30% during the tests, depending on the initial size of the video file. Here the two files that were considered are 413 KB and 1740 KB. After the client received the video, the resultant file sizes were observed to have reduced to about 315 KB and 1610 KB respectively.

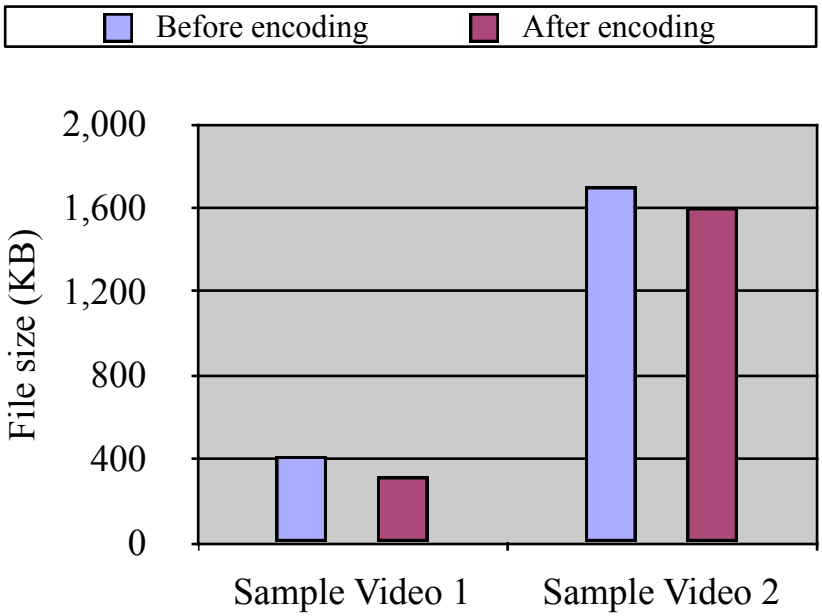

Figure : 5.2. File size reduction

5.5. SUMMARY

This chapter deals with performance measures, performance analysis and testing of the system under study. It describes the parameters that affect the functioning of the system and how each of the parameters are related to one another.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. CONCLUSION

Enabling large-scale VoD service in wide area Internet is crucial for many commercial applications. In order to provide an efficient VoD service in large scale P2P networks, this NCECD based scheme is proposed.

This scheme combines segmentation and encoding along with batching and patching as the basic service architecture and reinforces it with a Cache and Relay strategy. Based on the observations on user behavior and video inter-activities, the technique of association rule mining is exploited to find the associations within videos.

The segments requested in video inter-activities are thus accurately predicted according to the information collected through information exchange among peers. Moreover, a collaborative prefetching strategy is designed to optimize the resource distribution on the neighboring peers.

6.2. FUTURE WORK

This scheme has been demonstrated using a smaller network size. This scheme can be scaled to wide area networks in order to provide uninterrupted video service with efficient bandwidth usage and low delays. It ultimately reduces strain on the main server and utilises the network traffic more efficiently.

APPENDICES

APPENDIX - 1

SAMPLE SOURCE CODE

SERVER

Transmission and encoding of the video

```
private String createTransmitter() {

        String rtpURL = "rtp://" + ipAddress + ":" + port + "/video";
        MediaLocator outputLocator = new MediaLocator(rtpURL);

        try {
           rtptransmitter = Manager.createDataSink(dataOutput, outputLocator);
           rtptransmitter.open();
           try {
              Thread.currentThread().sleep(4000);
              }
              catch (InterruptedException ie) {
                 }
           rtptransmitter.start();
           try {
              Thread.currentThread().sleep(4000);
              }
              catch (InterruptedException ie) {
                 }
           dataOutput.start();
        } catch (MediaException me) {
           return "Couldn't create RTP data sink";
        } catch (IOException ioe) {
```

```
            return "Couldn't create RTP data sink";
        }
        return null;
}

void setJPEGQuality(Player p, float val) {
        Control cs[] = p.getControls();
        QualityControl qc = null;
        VideoFormat jpegFmt = new VideoFormat(VideoFormat.JPEG);
        for (int i = 0; i < cs.length; i++) {
            if (cs[i] instanceof QualityControl &&
                cs[i] instanceof Owned) {
                Object owner = ((Owned)cs[i]).getOwner();
                if (owner instanceof Codec) {
                    Format fmts[] =
((Codec)owner).getSupportedOutputFormats(null);
                    for (int j = 0; j < fmts.length; j++) {
                        if (fmts[j].matches(jpegFmt)) {
                            qc = (QualityControl)cs[i];
                            qc.setQuality(val);

                            break;
                        }
                    }
                }
                if (qc != null)
                    break;
            }
        }
}
```

PEER

Processing the video for playback

```
private String createProcessor() {
        if (locator == null)
           return "Locator is null";
        DataSource ds;
        DataSource clone;
        try {
           ds = Manager.createDataSource(locator);
        } catch (Exception e) {
           return "Couldn't create DataSource";
        }
        try {
           processor = Manager.createProcessor(ds);
        } catch (NoProcessorException npe) {
           return "Couldn't create processor";
        } catch (IOException ioe) {
           return "IOException creating processor";
        }
        boolean result = waitForState(processor, Processor.Configured);
        if (result == false)
           return "Couldn't configure processor";
        TrackControl [] tracks = processor.getTrackControls();
        if (tracks == null || tracks.length < 1)
           return "Couldn't find tracks in processor";
        boolean programmed = false;
        for (int i = 0; i < tracks.length; i++) {
           Format format = tracks[i].getFormat();
           if (  tracks[i].isEnabled() &&
                    format instanceof VideoFormat &&
```

```
                !programmed) {
                Dimension size = ((VideoFormat)format).getSize();
                float frameRate = ((VideoFormat)format).getFrameRate();
                int w = (size.width % 8 == 0 ? size.width :
                                (int)(size.width / 8) * 8);
                int h = (size.height % 8 == 0 ? size.height :
                                (int)(size.height / 8) * 8);
                VideoFormat jpegFormat = new
VideoFormat(VideoFormat.JPEG_RTP,
          new Dimension(w, h),

Format.NOT_SPECIFIED,
                                                Format.byteArray,
                                                frameRate);
                tracks[i].setFormat(jpegFormat);

                programmed = true;
            } else
                tracks[i].setEnabled(false);
        }
        if (!programmed)
            return "Couldn't find video track";
        ContentDescriptor cd = new
ContentDescriptor(ContentDescriptor.RAW_RTP);
        processor.setContentDescriptor(cd);

        result = waitForState(processor, Controller.Realized);
        if (result == false)
            return "Couldn't realize processor";
        setJPEGQuality(processor, 0.5f);
        dataOutput = processor.getDataOutput();
        return null;
}
```

Media player controller

```
public void controllerUpdate(ControllerEvent evt) {
if (evt instanceof ConfigureCompleteEvent ||
   evt instanceof RealizeCompleteEvent ||
   evt instanceof PrefetchCompleteEvent) {
   synchronized (waitSync) {
         stateTransitionOK = true;
         waitSync.notifyAll();
   }
} else if (evt instanceof ResourceUnavailableEvent) {
   synchronized (waitSync) {
         stateTransitionOK = false;
         waitSync.notifyAll();
   }
} else if (evt instanceof MediaTimeSetEvent) {
   System.err.println("- mediaTime set: " +
         ((MediaTimeSetEvent)evt).getMediaTime().getSeconds());
} else if (evt instanceof StopAtTimeEvent) {
   System.err.println("- stop at time: " +
         ((StopAtTimeEvent)evt).getMediaTime().getSeconds());
   evt.getSourceController().close();
}
}
```

REFERENCES

1. C.S. Chang, T. Ho, M. Effros, M. Medard, and B. Leong, “Issues inPeer-to-Peer Networking: A Coding Optimization Approach,”Proc. IEEE Int’l Symp. Network Coding, June 2010.

2. C. Feng and B. Li, “On Large-Scale Peer-to-Peer Streaming Systems with Network Coding,” Proc. ACM Int’l Conf. Multimedia, pp. 269-278, Oct. 2008.

3. C. Gkantsidis, J. Miller, and P. Rodriguez, “Comprehensive View of a Live Network Coding P2P System,” Proc. ACM SIGCOMM Conf. Internet Measurement (IMC ’06), pp. 177-188, Oct. 2006.

4. Dan Wang, Jiangchuan Liu, (2008), “A Dynamic Skip List-based Overlay for On-Demand Media Streaming with VCR Interactions”, IEEE Transactions on Parallel and Distributed Systems, IEEE, Vol. 19, Issue: 4, pp: 503 - 514.

5. Do, T.T., Hua, K.A., Tantaoui, M.A., (2004), “P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment”, In the proceeding of “IEEE International Conference on Communications, 2004”, pp: 1467 - 1472 Vol.3.

6. F. Liu, S. Shen, B. Li, B. Li, H. Yin, and S. Li, “Novasky: Cinematic-Quality VoD in a P2P Storage Cloud,” Proc. IEEE INFOCOM, pp. 936-944, Apr. 2011.

7. Ken Yiu, W.-P., Xing Jin, (2006), “Distributed Storage to Support User Interactivity in Peer-to-Peer Video Streaming”, In the proceeding of “IEEE International Conference on Communications, 2006 (ICC '06)”, pp: 55- 60.

8. K. Nguyen, T. Nguyen, and S.-C. Cheung, “Peer-to-Peer Stream- ing with Hierarchical Network Coding,” Proc. IEEE Int’l Conf. Multimedia and Expo, pp. 396-399, July 2007.

9. M. Wang and B. Li, “R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming,” IEEE J. Selected Areas in Comm., vol. 25, no. 9, pp. 1655-1666, Dec. 2007.

10. Yung-Cheng Kao, Chung-Nan Lee, Peng-Jung Wu, and Hui-Hsiang Kao, (2012), “A Network Coding Equivalent Content Distribution Scheme for Efficient Peer-to-Peer Interactive VoD Streaming”, IEEE Transactions On Parallel And Distributed Systems, IEEE, Vol. 23, Issue: 6, pp: 985-994.