A

DISSERTATION

ON

ANALYSIS OF HETEROGENEOUS PEER TO PEER VIDEO STREAMING

Submitted under the partial fulfillment of the requirement
for the award of the degree of

MASTER OF TECHNOLOGY
(SOFTWARE ENGINEERING)

Submitted by:
RAJ KUMAR
Roll no.: 12/SE/09
Reg. No.: 14/MT/SE/FT

Under the guidance of:
DIVYASHIKHA SETHIA
Assistant Professor,
Department of Computer Engineering

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
DELHI, 2011

SESSION 2009-2011

CERTIFICATE

DELHI TECHNOLOGICAL UNIVERSITY
BAWANA ROAD, DELHI – 110042

This is to certify that the work contained in this dissertation entitled **“ANALYSIS OF HETEROGENOUS PEER TO PEER VIDEO STREAMING”** submitted by RAJ KUMAR, Roll no. 12/SE/09 in partial fulfillment of the requirement for the award of the degree of Master of Technology in software Engineering at Delhi Technological, Delhi is a record of the candidate’s own work carried out by him under my guidance and supervision in the academic year 2010-2011.

(PROJECT GUIDE)
DIVYASHIKHA SETHIA
Assistant Professor,
Department of Computer Engineering
Delhi Technological University
Delhi

ACKNOWLEDGEMENT

It is a great pleasure to have the opportunity to extend my heartiest felt gratitude to everybody who helped me throughout the course of this project.

I take this opportunity to express my deep sense of gratitude and indebtedness to my learned supervisor **DIVYASHIKHA SETHIA** Asst. Professor, Dept. of COE, DTU, for her invaluable guidance, encouragement and patient reviews. With her continuous inspiration only, it has become possible to complete this dissertation. She helped in pointing out places in several drafts of the thesis where clarity could be improved and claims made more precise.

I also extend my gratitude towards **Dr. DAYA GUPTA,** Head, Dept. of COE, D.T.U., Delhi who has always been cooperative throughout the whole coursework and gave us valuable inputs. I am also thankful to all other faculty members and staff of the Department of Computer Engineering at Delhi Technological University for sharing their knowledge and experiences with me as well as for their kind support.

I also like to thank my batch mates at Delhi Technological University for sharing their ideas and opinions on several topics that were important for my work. I also owe gratitude towards my parents for their patience and support. They have been always around to cheer me up in the odd times of this work.

RAJ KUMAR

ABSTRACT

This thesis deals with the survey of peer to peer video streaming protocols and simulation of peer-to-peer video streaming architecture for heterogeneous node. The nodes are having different bandwidths and node may join and leave the system dynamically. The nodes of the ALM tree are deployed over the Internet widely and heterogeneously. So the extra transfer delay of packet will be added due to the bandwidth bottleneck between the heterogeneous nodes. Each node is having different packet loss, startup delay and playback jitters caused by the network congestion between the heterogeneous peers. In this thesis, a framework SmartPeerCast combination of the ALM tree and mesh framework of the P2P overlay is described. It maintains simplicity of ALM architecture and also utilizes all peers uploading bandwidth like the mesh framework to improve the startup time delay and playback jitters performance of the RTVB application .The nodes are added to overlay network based multicast tree at application level. It utilizes the leaf nodes bandwidth of high quality tree to forward the lower quality streaming to the nodes of the lower quality stream ALM trees. It uses QOS messages to change forwarding bandwidths and minimizing the packet jitters.

CONTENT

ACRONYMS

P2P	Peer to Peer
ALM	Application Layer Multicast
RTVB	Real Time Video Broadcasting
VOD	Video-On-Demand
QoS	Quality of Service
MDC	Multi-Description Coding
BSI	Bandwidth Sharing Index
RTT	Round Trip Time
ESM	End System Multicast

LIST OF FIGURES

LIST OF TABLES

MOTIVATION

Media Streaming over Internet is getting more popular day by day. Websites, such as YouTube [38], provides media content to millions of viewers. The earlier conventional solution for such applications is the client-server model, which allocates servers and network resources to each client request. So the client-server model is not appropriate when number of client increased. It fails in situation where resources are not increasing according to client. There are very less companies, like Google, who can afford providing such type of solution by increasing the number of resources. So there is a need of finding other solutions. It is an active field of research. IP multicast is an efficient way to multicast a video stream over the network, but it less support by the Internet Service Providers so it is rarely used.

An alternative solution is Application Level Multicast (ALM) [34], which creates overlay networks to disseminate large-scale video streams to many numbers of clients. Peer-to-peer overlay is an overlay network .In Peer-to-peer overlay each peer works as a sender and receiver both so it can download the data as well as upload the data. In this scheme the peers who have requested data either data is subset or complete can forward it to requesting peers. If number of peers increases the capacity of system increases. Peer-to-peer streaming is challenging because to have a smooth media playback, User is having time constraint while receiving data. Otherwise, either the quality of the playback is reduced or the continuity of the playback is disrupted. In live streaming, user must get the most recent part of the media delivered by the provider. Satisfying these timing requirements is more challenging in a dynamic network, where nodes join, leave, fail continuously, it is called churn. And network capacity changes due to network congestion.

Many different solutions have been already provided for peer-to-peer media streaming, but few of them have been able to satisfy all the above mentioned requirements. Many video streaming requirements are also conflicting, if one wants to get high quality stream, then he will have to store the data in buffer which will result high playback latency and start up delay

CHAPTER 1

INTRODUCTION

1.1 P2P VIDEO STREAMING

Peer-to-peer (P2P) video streaming have become popular. CoolStreaming [1] is first P2P live video application. This may be expected P2P video streaming data will highest demanded on the internet in the near future. P2P video streaming systems can be broadly classified into two categories: live streaming and video-on-demand (VoD). In a live streaming system, the live video content is distributed in real time and video-on-demand permits users watch video clips whenever they want. In P2P system, peers download data from the servers and other peers, and also upload what to other peers who required data. This also reduces the bandwidth burdens of the servers and also provides peers with good quality of service. The major issue in P2P live streaming is to provide real time data which is current and most recently updated. This means the difference of playback point between peers should be minimized.

Traditionally, video stream on the Internet follow the client-server model, in which a lot of servers are having much larger capacity and there are many limited capacity clients. This model has many problems. First, the capacity and bandwidth of servers is finite, so it suffers from scalability problem so number of client will be limited. Centralization of the system makes it single point of failure. In recent years application-level multicasting (ALM) [34] or end system multicasting (ESM) [29] has emerged as a solution to IP level multicasting for providing information to large sets of clients. But ESM in a dynamic Internet-scale environment have number of problems. First, an ESM system usually duplicates data on end-hosts and distributes them through multi-hop IP unicast links. A important challenge for ESM systems is to get high efficiency and minimize multicast latency. Second, various hosts have different computing capacities, their uploading network bandwidths, and their willingness and ability to share their resources. There is a need for an efficient ESM protocol that can make different clusters according to their properties and distribute work according to that. Third, nodes leave and join the network dynamically.

Two main methods for building overlays for P2P multicast media streaming are tree-based [2] and mesh-based [3, 4, 5, 45]. The tree based approach explicitly places peers in a single tree or multiple multicast trees, where they receive the stream from their parents and forward it to their children. In the mesh-based approach, the P2P overlay is unstructured. Peers are connected to each other, which is randomly selected. The video stream is break down into small data blocks sub stream are exchanged between neighbouring peers. The main advantage of mesh overlays compared to tree-based overlays is their much higher robustness to peer churn. In tree-based approaches, a peer can receive data only from its fixed parent and when that parent fails or leaves the network its whole sub-tree loses that data until the tree is again constructed. In mesh-based streaming systems, data chunks can be obtained from any neighbour that holds it and thus when one neighbour fails other neighbours may still provide the data.

1.2 OUTLINE OF REPORT

The report is organized as follows: Chapter 2 is an overview of Literature Survey. Chapter 3 discusses the Analysis of existing systems. Chapter 4 gives SmartPeerCast description. The simulation of SmartPeerCast is discussed in Chapter 5. The conclusion and future work is discussed in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

2.1. DELIVERY OVERLAY TOPOLOGY

There are mainly two overlay architectures for video transmission one is tree based and second is mesh based. In tree based approach peers are linked structurally. Nodes are having a fixed structure. In mesh overlay architecture nodes are not having a fixed architecture. Peers randomly connect to each other. Each peer works as a parent and as a child. Each one is having some merits and some demerits.

2.1.1. TREE BASED OVERLAY

Router makes the IP multicast tree on network layer. In the similar fashion user maintain the tree at the application layer in video streaming. Node receives the video from the parent node and distribute to the child node. There may be different architecture of given peers. Some parameter for creating architecture is depth of the tree and fan out of the non leaf nodes. Peer at the leaf node receive the data after the upper level nodes. The various nodes have different uploading bandwidths. Some buffering mechanism is used at the leaf nodes for reducing the transmission delay. Fan out of any nodes are limited to its uploading bandwidth capability.

Overlay architecture may be centralized or distributed. In centralized scheme only a single server is responsible for tree construction and maintenance of tree. If any peer wants to join in the tree it contacts to the central server. Server takes decision for the joining of node in the tree on the basis of node characteristics and topology. If any node crashes then central server takes the decision on the maintenance of the tree. Only a single node is responsible for whole architecture maintenance. Other nodes do not have any work for the maintenance of the tree.

This scheme is also having some limitation because it is a single point of failure. If server is crashed overlay is not going to be maintained.

In distributed approach every node is responsible for the maintenance of topology. The system is fault tolerance, because it is not a single point of failure. It is a scalable topology. Peers also may be connected to various trees .this is called multi tree architecture. In this topology there are many trees and peer which is leaf node in any tree will be on upper level in other tree. These trees are unique. For minimizing the delay the height of tree is minimized. MDC is used in tree topology. In MDC a stream is encoded in multiple sub streams. These sub streams are called descriptions. Number of description is directly proportional to the quality of video. If a peer receives various unique descriptions then the quality of receiving video will be high.

i. Nodes keeping on leave and join the topology the architecture suffers from the stability.
ii. The received video quality is heavily depends on non leaf node’s uploading bandwidth.
iii. As the depth of tree increases the delay increases.
iv. Multiple tree algorithms suffer from the problem of redundant path among the peers.

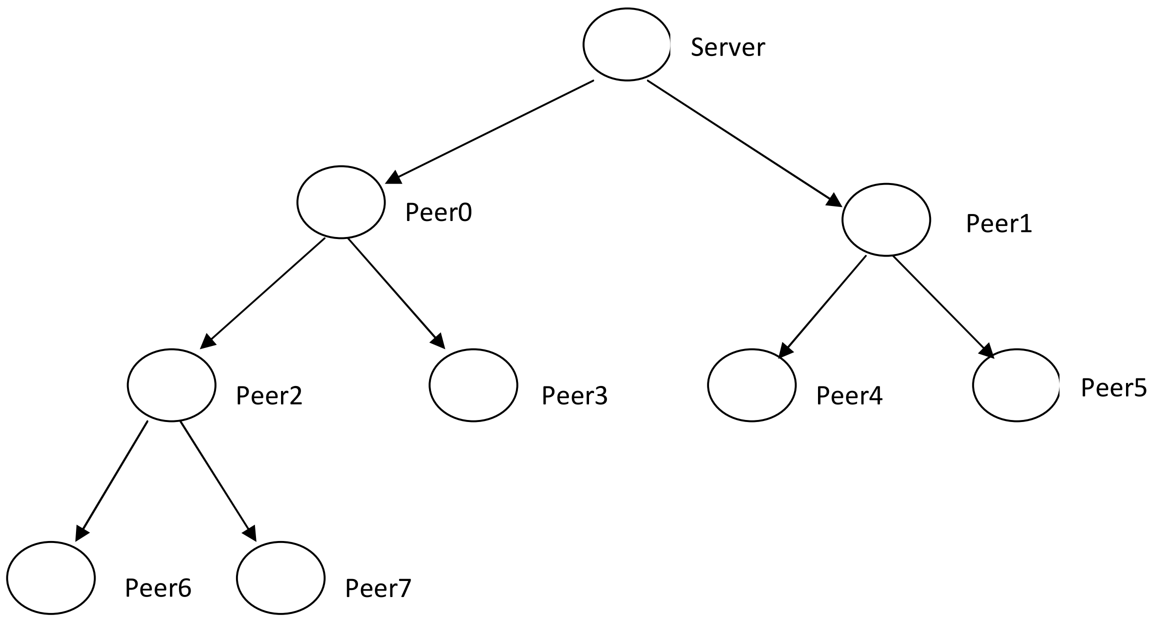

Fig 1: A tree-based overlay

2.1.2. MESH BASED OVERLAY

As in overlay tree architecture peer are connected in a fixed architecture. Every node is having a single parent and fixed number of child. So it is fixed from where the stream has to taken and where stream is to distribute. And if parent leaves or crashes in that situation this node and its entire child cannot take stream from anywhere. So if one node crashes its entire child fails to get stream. In mesh overlay architecture, there is no fixed architecture. Peer in this architecture changes the relationship to each other dynamically. A node can get stream any of its neighbors at any given time; a peer maintains peering relationship with multiple neighboring peers. A peer may download or upload video from or to multiple neighbors simultaneously. If a peer crashes then other nodes are not too much effect they are still taking the stream from other node. A node may be connected to many numbers of nodes. So it is fault tolerance architecture .so data availability is more in this architecture in respect to tree based architecture .one node may get the data from many number of nodes. Data is distributed to all nodes in the mesh.

i. This is not a single point of failure architecture.
ii. This is fault tolerance architecture.
iii. Stream availability is more than the tree architecture.

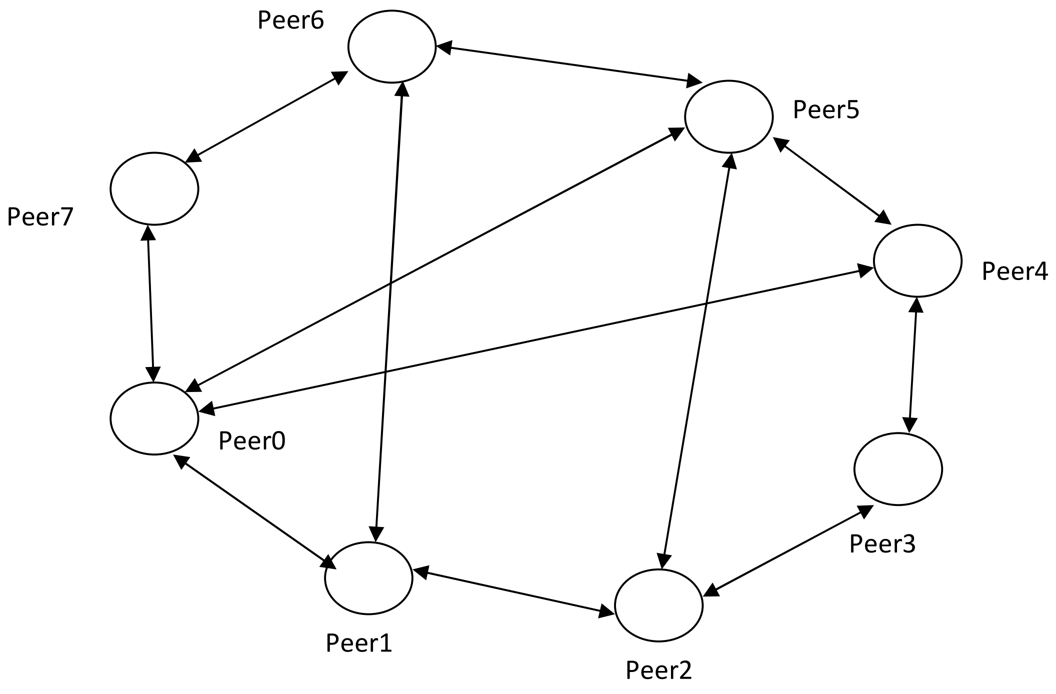

Fig 2: A mesh-based overlay

2.2. DATA EXCHANGE DESIGNS

There are two methods of data exchange:

i. push-based systems
ii. pull-based systems

2.2.1. PUSH BASED SYSTEM

In a push-based system [33], a peer distributes a received stream to all of its neighbours. In tree based architecture parent node distribute the data to the Childs. Same rules follow for all the nodes in the tree. In mesh based architecture there is no fixed child parent relationship and some node may get data from multiple nodes. So some data redundancy also occurs there. So every node distributes the data to each node to which it is linked.

2.2.2. PULL BASED SYSTEM

In a pull-based system, peers exchange data availability information to each other after a fixed interval of time. They share what data they have to each other. Now a peer may decide from where the data must be taken and what data must be taken. Then node request to other node about the data. In this scheme there is no data redundancy .but this scheme suffers from data availability information exchange after some interval and request process for getting any data.

2.3. VIDEO DATA CODING TECHNIQUE

The goals of video coding are represent the video data .The biggest goal of video coding is compress the video size so that it will take less memory for storage and less time in processing. And it also adds some meta-information and digital rights. The video stream is encoded in one or more sub streams. For the reconstruction of original stream these sub streams can be decoded. In P2P streaming systems as in multiple trees, a node can receive the video data e from multiple peers. A node can get various sub streams from multiple nodes after collecting the stream the stream must be decoded before playing. There are various data coding technique as single layer coding, layered coding, and Multiple Description Coding (MDC) [32], Redundancy-Free Multiple Description (RFMD) [32].

2.3.1. SINGLE LAYER CODING

In a single layer method, the video is encoded into a single layer, and every Group of Pictures (GOP) is divided sequentially into many blocks, so that formers blocks contain data from formers frames in the group. Each peer holds one block from each group. In case one block is missed, the frames contained in the next blocks of the group are not decodable and only the frames contained in the former blocks can be decoded. If the block number second is not present, only block number first can be decoded [9].

2.3.2. LAYERED CODING

There are various video quality layers**.** Video layers are sent in different multicast groups. In this scheme important layers of video are distributed with high quality of service (QoS) and the less important layers of video are distributed with fewer QoS. The base layer gives a basic level of quality and it can be decoded without enhancement layers. Enhancement layer does the base-layer quality and it is not useful.

MPEG-4 Fine Grain Scalable (FGS) is a demanding method for creating layered coding. A FGS encoder encodes the video into two layers, a base layer and a scalable enhancement layer. The enhancement layer is further break down into M −1 sub streams. So now there are M sub streams. One another scheme Scalable Video Coding (SVC), also has a base-layer and an enhancement layer. In both of the technique, the rate of the base layer must be high so that quality can be get from the base layer only. In comparison to single-layer coder, the distortion in scalable coder at the same rate is more [9, 43].

2.3.3. MULTIPLE DESCRIPTION CODING

“MDC [32, 40] creates many independent descriptions of the same signal. Video quality is directly depending to the number of descriptions that are received. MDC provides a solution for multi-path streaming situation, where independent descriptions may be sent to different connection in tree. It is less efficient than scalable encoding however; it is good in case of packet loss [9]. A popular scheme for multiple descriptions encoding with many descriptions is Multiple Description source coding through Forward Error Correction codes (MD-FEC) [9].

Firstly MD-FEC encodes each GOP into M layers. This is performed with any video coder such as MPEG4 FGS or SVC. It is shown in Figure 3 (a) for the case of layers. It is denoted by $L1$, $L2$, $L3$, and $L4$ for the bits in these 4 different layers. The m th layer is then again divided into m equal-length groups. It is shown in Figure 3 (b), layer two is divided into two equal-size groups $L21$ and $L22$; layer three is divided into three equal-size groups $L31$, $L32$ and $L33$; and layer four is divided into four equal-size groups $L41$, $L42$, $L43$ and $L44$. Then a (M, m) Reed- Solomon code is applied to the m groups from layer m to build M groups. After it $M2$

groups are then rearranged as in Figure 3(c). In layer 1, the RS method generates three redundant groups *R*11, *R*12, and *R*13. For layer 2 it generates two redundant groups *R*21 and *R*22. And for layer 3 it generates 1 redundant group *R*31. Thus, due to the RS code, if any *m* of the *M* groups is received for layer *m*, then layer *k* could be decoded”[9].

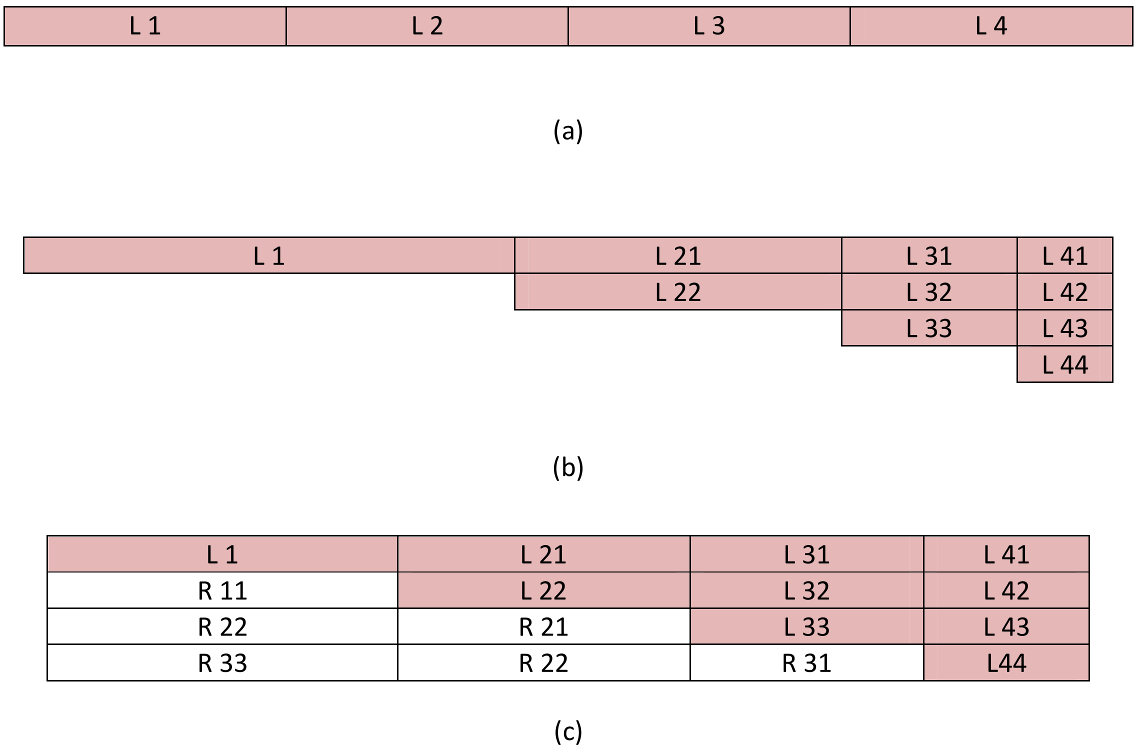

Fig 3: MD-FEC encoding method

When M2 group is generated the groups in rows are collected and sub stream are created in Figure 3 (c). The first sub stream is generated by combining *L*1, *L*21, *L*31 and *L*41. The fourth sub stream is generated by combining *R*13, *R*22, *R*31 and *L*44. The sub streams have the following important properties:

- Each sub stream has the same bit-rate.
- If receiver wants to recover *m* layers from the original layer encoded video, then receiver is required to receive any *k* of the *M* sub streams.

Thus each stream is having equal importance [9]. One similarity between MD-FEC and multiple description technique is that both can create any number of sub streams from a scalable stream generated which is generated by scalable coder. This is more desirable for P2P VoD. When a supplying peer crashes, the video quality should not be minimizing if it is waiting for another peer which can continue to supply the sub stream. When a receiver receives *m* of the *M* sub streams, only a limited portion of bits are used (for all values of m). Unused bit are having drawback and consume uploading bandwidth [9].

2.3.4. REDUNDANCY-FREE MULTIPLE DESCRIPTION CODING

It is a new multi-stream coding method proposed in [9]. This method tells that P2P video streaming is time critical application. Video transmission delay is very critical while measuring the quality of video. So in this method it is described all the encoded bits should not be send .redundant and unused bit must not be sent. This new coding scheme is taken from traditional MD-FEC coding.

Algorithm is described as below:

- Data is first encoded by MD-FEC and create *M* descriptions;
- If there are p supplying peers are , then each supplying peer only sends *k*/*m* portion of the data for layer *k*, where *k* = 1, . . . , *m*; each supplying peer sends different portion of layer *k* data;
- The work of receiving peer is to collect received *m* sub streams and determines layer *k* (*k* = 1 to *m*) by (*M*, *k*) FEC decoding and the lowest *m* layers are gotten.

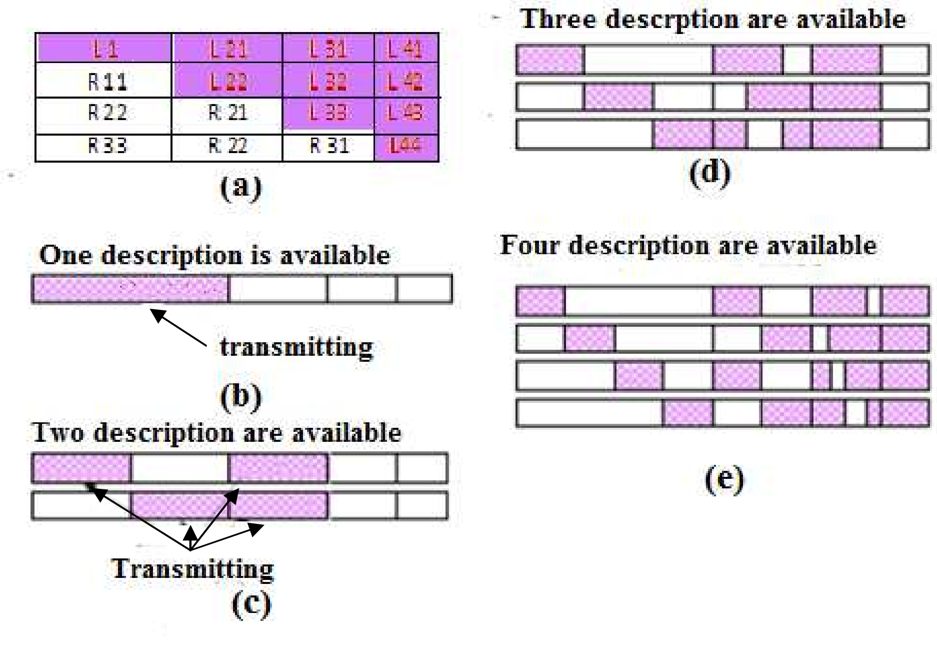
L 1
L 21
L 31
L 41
R 11
L 22
L 32
L 42
R 22
R 21
L 33
L 43
R 33
R 22
R 31
L44
(a)
One description is available
transmitting
(b)
Two description are available
Transmitting
(c)
Three descrption are available
(d)
Four description are available
(e)

Fig 4: Redundancy-Free Multiple Description Coding (RFMD)

“Figure 4 shows the RFMD transmission of MD-FEC data (M=4):

Figure 4(a) shows purple color represent the stored data in each description as source node , and gray color represent redundant bits; figure 4(b-e) represents the portion of the data (purple) delivered by each supplying node” [9]. Transmission rate on each peer is not constant. Data transmission rate heavily depends upon number of peers. There are methods for getting constant bit rate for different layers before MD-FEC encoding [9].

2.4. VIDEO QUALITY METRICS

Some common measuring metrics in P2P video streaming are described below. These Metrics are useful in evaluating the system performance.

1. Startup delay

This metric is very important in P2P video streaming systems [48]. For continuity of video watching end systems must buffer various data chunk before the starting of playback. Some systems have a fixed threshold value as number of chunk. This work is mainly filling the buffer with data chunks. The time to fill up this buffer with these chunks is called startup delay.

2. Transmission delay

It measures time for sending a data chunk from source node to sender. It may be seen as difference of playback point between peers while watching same live video. It must be minimized in live video or video on demand.

3. Streaming rate

This metric means the data streaming rate in the system. The system efficiency is determined by Maximum average streaming rate and average streaming rate. It is its download rate for any client. This metric is not demanded in P2P streaming because in P2P streaming there is no requirement to maximize download rate.

4. Continuity rate

This metric is demanded in P2P streaming system. It describes the continuity. This measures continuity of video how likely the video can be played without skips or pauses. It is also a good measuring metric to evaluate goodness of a scheduling algorithm.

5. System scale

It basically defines the numbers of peers. If the number of peers are more then there is no need maximize streaming rate. It is expected when average streaming rate is larger than the inheret playback rate of a video; all the peers are able to playback fluently. Thus extra bandwidth of the system can be used to add new peers to make the system scalable. The total number of all peers within the system is system scale.

2.5. PEERS HETEROGENEITY

Peers are heterogeneous in respect of access bandwidth, storage capacity and processing power and it changes dynamically. Systems must handle heterogeneous networks with different uploading bandwidths nodes. In heterogeneous bandwidth environments where one node is on DSL and another one is on Ethernet, P2P video streaming systems must manage the heterogeneity of network. The high bandwidth node must get data from high bandwidth node with good downloading speed and low bandwidth node must get data from low bandwidth node with less downloading speed .if high bandwidth node send data to low bandwidth node then uploading bandwidth is wasted. So there is a need of grouping the node according to their uploading bandwidths for optimal bandwidth utilization [11, 41].the performance of any topology depends upon number of high bandwidth peers. As the number increases performance increases and it depends upon the bandwidth of parent. In tree architecture performance heavily depends upon the depth of the tree if number of peer having high bandwidth increases the depth of tree reduces and performance and quality increases. The delivered quality and buffer requirements of high bandwidth peers are affected by the degree of bandwidth heterogeneity and the percentage of low bandwidth peers. If low bandwidth node is in large number the child node also suffers from parent speed it will raise content deadlock [13] and reducing the quality among peers.

Peers heterogeneity is also very important for video segments distribution among peers. If a segment exists at a large number of nodes, then it is considered that segment is being well represented, but if mostly nodes sharing the segment are having low upload bandwidth, and then aggregate delivery capacity for this segment will be very low. Current seeding capacity of the peers must take in account from avoiding such segment distribution problems [14].

CHAPTER 3

ANALYSIS OF EXISTING SYSTEM

3.1 COOLSTREAMING

CoolStreaming [1] is a data-driven overlay network which used for P2P live media streaming. It was developed in Universities of Hong-Kong and Vancouver. It is developed in Python language and it is a mesh base architecture. There are three layers in architecture: network layer, streaming layer and display layer. It is having good scheduling algorithm and good buffering procedure. This makes CoolStreaming get a smooth video playback and a very good scalability and get good performance. The system is tested over the Planet Lab test-bed [15]. It does not have any fixed architecture but based on data availability. Media streams are split into fixed size segments called stripes. Each peer has a buffer map consisting of 120 bits. Each peer shares data availability information to other peers. Each peer then downloads the data if it requires at that time.

Analysis: It is data availability based system. It have feature just like BitTorrent [18] .Cool Streaming is nice video streaming application. Data distribution algorithm is good and provides good performance. It less suffers from the peer crashes as in tree based approach.

3.2 END SYSTEM MULTICAST:

End System Multicast (ESM) [19] is developed by Carnegie Mellon University. It broadcasts the information to many users. It is a single tree approach that means any node may get data from its parent. Many algorithms are developed which group the nodes according to their uploading bandwidth. In addition, ESM tested the system on Planet Lab test-bed [15]. They also provide a contributor-aware policy. They do not support first-come-first-served approach, which is the major advantage. In a contributor-aware policy, each node knows about all the other nodes. They know which node is contributing and what data they are distributing. The first-come-first-served approach does not make any differences between peers so it does not have good performance. Thus contributor-aware policy utilizes resources more properly in comparison to the first-come-first-served basis. The first peer, who connects first, gets the service earlier. An important point is that it automatically detects the uploading capacity of a peer and help in judging when downloading or uploading data from the network [8]. ESM also have challenge with a network such as NATs and firewalls. ESM proposes to use NATs and firewalls [20] as peers within the overlay network.

Analysis: ESM is one of the most important of P2P media streaming and the goal is mainly to satisfy that application-level overlay multicast is a possible solution. It is a single tree architecture which is highly sensitive to peer's failures or disconnection. It also deals with the heterogeneity.

3.3 GNUSTREAM

Gnustream [22] is a receiver-driven peer to peer algorithm. It is implemented by Purdue University. There are three layers in this architecture

1. Control Streaming Layer (CSL)
2. Media Player Layer (MPL).
3. Network Abstraction Layer (NAL)

The task of NAL is to provide features, such as locating, routing and retrieving data the network. SCL is the main layer and it balances the download of data from various sources. It synchronizes the supply to the peers that are present in dynamic P2P overlays and potential network congestions. MPL is a build player with a limited buffer control [22]. This C++ application runs on top of the Gnutella P2P network [22]. It supports streaming of data from multiple sources and achieving load balancing between sending peers. The peer may be even or proportional in any overlay architecture. Whereas in the former case all peers send the same amount of data, in the later approach peers send as much data as they can send. Gnustream also assigns one or more sender peers as backup peer that helps in recovery after failure take place within a streaming session. The receiver performs sorting of the video streams and makes the re-computation of the media files. The player displays solely the video streams and does not process the audio signals. Additionally, Gnustream also provides a different variety of buffers for different purposes, which ensure responsiveness and robustness: double buffering for the display, control buffer, decoder buffer, playback buffer and data availability and data collection buffer. MPEG has been chosen as the only format because it is most demanded over the Internet.

Analysis: Similar to Cool Streaming, Gnustream has a layered architecture to build the solution. There are three layers in the system one to communicate with the network, one for streaming and one for the display, it is a good approach. The use of a single media format for video and for audio is a good approach in heterogeneous P2P streaming system. Mobile phones do support basic MPEG files. Finally, GnuStream has been implemented on the top of an existing P2P overlay: Gnutella. Even if the system is good for responsive and resilient to the failure of peers, it is good for small amount of users. Gnutella communicates using query flooding. More users mean more control messages over the network, so it can create congestion and can directly impact the streaming session. Uncontrolled broadcast can also problem in the case when modem links are present, which then slows down the propagation of queries. On the other hand, there are two major advantages of building a streaming solution on the top of an existing P2P network such as Gnutella. It is also lot easier to attract a large amount of participants using already deployed P2P overlays.

3.4 HYPERCAST

Hypercast [24] is a research project developed by the University of Toronto. The aim is to construct a robust P2P overlay multicast network. It is based on graphs. At the moment, the overlay is based on Hypercube and Delaunay triangulation topologies. It uses well defined graphs; it becomes possible to forward messages through the overlay network and do not need of routing algorithm. Hypercube and Delaunay triangulation graphs are used .The logical addresses of nodes is used to determine the next hop for routing information. The data transmission uses tree-based distribution architecture over either UDP or TCP. Each peer has an interface to the network called "overlay socket", which gives the basic P2P features such as leaving and joining of nodes the network and sending and receiving data. Hypercast is a Java framework to build P2P multicast applications. Hence, a media streaming application could be built on the top of it.

Analysis: Hypercast is a framework that helps in the development of multicast based applications. This Java framework is a good basis for implementing a P2P streaming system. It provides some useful idea for developing P2P streaming systems. A peer receives data from the Hypercast overlay network and forwards these data to an external and more powerful player via

UDP. This way, the receiver does not need to handle complex media management methods such as decoding.

3.5 PEERSTREAMING

PeerStreaming [25, 37] is a Microsoft P2P media streaming system. It is receiver driven approach. The general architecture follows a client/server scheme and the P2P network helps the server in disseminating the video content. In addition, when a peer views the media, it achieves at the same time a copy on its local hard-drive. It can now provide the media to other requesting peers, and help server by reducing load from the server. Any peer in the network could then provide the whole or part of the media to a client. It is important to design lightweight peers, which are less dependent on each other. A peer helping the server delivering the data should perform simple operation. The client has more responsibility and should perform more complex tasks: coordinating the peers to each other, receiving the media from multiple peers, doing load balancing, handling peers online and offline status and displaying the media in real-time. It is important to understand that both, servers, serving peers and clients are all nodes in the overlay network. A server is a peer, which has the data and sends it to the client, a serving peer is equivalent to machine in the P2P overlay, which has also the data or part of it and sends it to the client. A client is a peer, which requests data from the network.

Analysis: It supports multiple clients Load-balancing between sender peers is a good solution to enable streaming from multiple sources. Similar to GnuStream, a client can take data from multiple peers, a provider sends as much data as it can send at any given time. So it results in a proportional allocation of the streaming load.

3.6 P2PCAST

It is developed in C++ with the use of libasync library. P2PCast [26] is developed in the University of New York. It is forest overlay architecture. It is scalable and it can provide information to many numbers of users. It leverages the bandwidth of all users and then delivers the video stream. It firstly divide the stream into sub stream these sub stream is called stripes. Each stream is distributed through various multicast tree. Theses tree are repaired and maintain locally. Each peer contributes equally.

Analysis: It is an enhanced version of single-tree approach which suffers from many. If one node is source node in one tree then it will be leaf node in other tree. It has some memory management problems. Every peer contributes as much as consume. It does not rely on a single node while all the peers contribute approximately equally.

3.7 SWISTRY

Swistry [27, 42] is a P2P live video streaming application. It is developed in java. It is mesh based overlay architecture. It is a layered architecture. Peers are grouped into cluster according to their bandwidths. It was developed by E Zurich. It has similar characteristics to CoolStreaming. Transmission depend on the data presence when any node have the data it inform to other about data availability .After getting the information from the neighbors peers can download the data if they the need of data. All nodes in any layer have sufficient speed to download and upload the data .Different layer have different bandwidth nodes.

Analysis; It is java based System. It groups the peers into various layers. It is similar to the CoolStreaming. It increases the performance by grouping the node according to their bandwidth it reduces the transmission delay and also maintain the quality of video.

3.8 ZEBRA

Zebra [23] is a streaming system. It is developed by the MIT .This is used for medium type network. It is a tree based overlay system. It builds two trees. In this architecture if one node is source in one tree then it will be leaf node in other tree. Every node forward stream to its child but leaf node does not distribute data because they do not have any child. If any node leaves the system then only one node is affected. The server proxy is the manager of whole system it has the responsibility to manage whole architecture. If any node leaves then it changes tree. It has good performance for 100 users near about. The server proxy divides the stream into two sub stream and then forward to both of the trees. It increases the performance by grouping the node according to their location. If any peer join to the system then it contact to proxy server. Proxy server chooses the parent node for that node on the basis of round trip time.

Analysis: It is multiple tree overlay architecture. It is developed in C++.It is not vulnerable because it depends only on proxy server. If it crashes the system fails. For increasing the performance and reducing the delay nearby nodes are grouped in a cluster because they have less round trip time. If any node fails then its immediate is only affected

TABLE 1: A comparison of various P2P streaming Application

application / criteria	**CoolStreaming**	**ESM**	**Gnustream**	**Hypercast**
Streaming type	Live	Live	On demand	Live, On demand
P2p network	Own	Own	Gnutella	Own
Transport protocol	RTP	TCP	HTTP	UDP
Overlay type	Mesh	Single tree	Mesh architecture	Single tree architecture
Overlay size	Large	large	Medium	Medium, and small
streaming rate	Approximately 500kbps	Video: from 300 kbps to 100 kbps Audio:20 kbps	Approximately 150kbps	Not specified
mobile user support	No	No	Very little bit	Support to pda
Number of node	50000 nodes using PlanetLab	4000 real time user	5 node xenon	2 server node one client node
Programming language	Python	c ++	Visual c ++	Java language
Supporting format	Real video, window content	MPEG 4	MPEG1 and MPEG1	MPEG
Goodness	Distribute architecture	Application level multicast, automatic bandwidth checking	Layer architecture, load balancing	Routing protocol not required , java framework
Weakness	Message passing and scheduling algorithm are difficult	Peer leave continuously in single tree	Audio is not supported	It do not give focus on streaming

application / criteria	**PeerStreaming**	**P2PCast**	**Swistry**	**Zebra**
Streaming type	On demand	Live	Live	Live
P2p network	Microsoft SDK	Own	Own	Own
Transport protocol	TCP	TCP	UDP and HTTP	TCP
overlay type	Server p2p client	Multiple tree architecture	Mesh	Multiple tree
Overlay size	Large and medium	large	Medium	Approximately 100 node
streaming rate	Approximately 100 kbps	Not specified	220 kbps, 320 kbps, 530 kbps	40 kbps
mobile user support	No	No	No	No
Number of node	8 nodes	10 nodes	50 nodes	10 nodes
Programming language	C ++	C ++	Java	C ++
Supporting format	MPEG1, MPEG2, MPEG4, WMV,WMA	Not specified	MPEG1 and MPEG 2,MP3	Real video
Goodness	It defines media structure and good load balancing	Good tree structure	Clusters according to bandwidths.	Group peer which is near to each other. Only immediate child will change its parent.
Weakness	All processing is done only on client side,	Complex to manage	Support only old format	Single point of failure on server

CHAPTER 4

SMART PEER CAST

4.1 INTRODUCTION:

SmartPeerCast: it is a broadcast P2P framework. The SmartPeerCast [39] overlay network is composed by three type of nodes tracker, Source and peers.

The tracker node: It does the registration of the peers and the source node. It also changes the tree architecture dynamically. When peer leave or register to the network. The peer who is having higher uploading bandwidth is assigned to the root of ALM tree of the higher quality output channel. The peer registers to the tracker with the given information;
<Peer id, upload bandwidth, the maximum acceptable input connections, stream id>
Any peer may leave the network. If the peer leaves the ALM tree [34] tracker unregister this peer with the following details <peer id, stream id>. If peer crashes then child of this peer will contact to tracker for searching new parent node.

The source node: This is the root node of ALM tree and provides the real video for whole tree architecture. Source node may be TV capture card, an IP camera. These provide MPEG-4 or H.264 encoding stream [9] as the output. There are three ALM trees of various Quality .These three output channels output three different levels of quality with real time streams in the source node. The data is replicated to three source node of tree of different quality. High quality stream is 2 Mbps bits rate, the medium quality stream is 1 Mbps bits rate, and the low quality stream is 500 Kbps bits rate respectively. For every different quality a different ALM tree. The nodes are grouped in various clusters according to their uploading bandwidth

The Peers: These are in huge numbers in Network. The peers are heterogeneous because they are having different bandwidths in real video system. The transrating engine is used for translate the bandwidth. The sending peer uses the transrating engine for two purposes.

(1) Sending peer change its uploading stream quality dynamically if receiving peers bandwidth does not match with sending peer bandwidth. If receiver peer is getting data more than or less than the expected speed in that situation it sends QOS message. On the basis of QoS message sending peer changes its uploading bandwidth.

(2) If heterogeneous peers are communicating to each other. If the peers in the high quality stream ALM tree upload data to the peers in the medium quality tree or the peer in low quality tree then transrating engine is used to change the uploading bandwidth. It reduces the playback jitters and utilizes the leaf nodes bandwidth.

Thus the SmartPeerCast performance is increased by fully utilizing the ALM trees' leaf nodes. The receiver scheduler is the important module. If the sending peer quality is not meeting with receiving peer's QoS requirement, the receiving peer selects a new parent for data transmission.

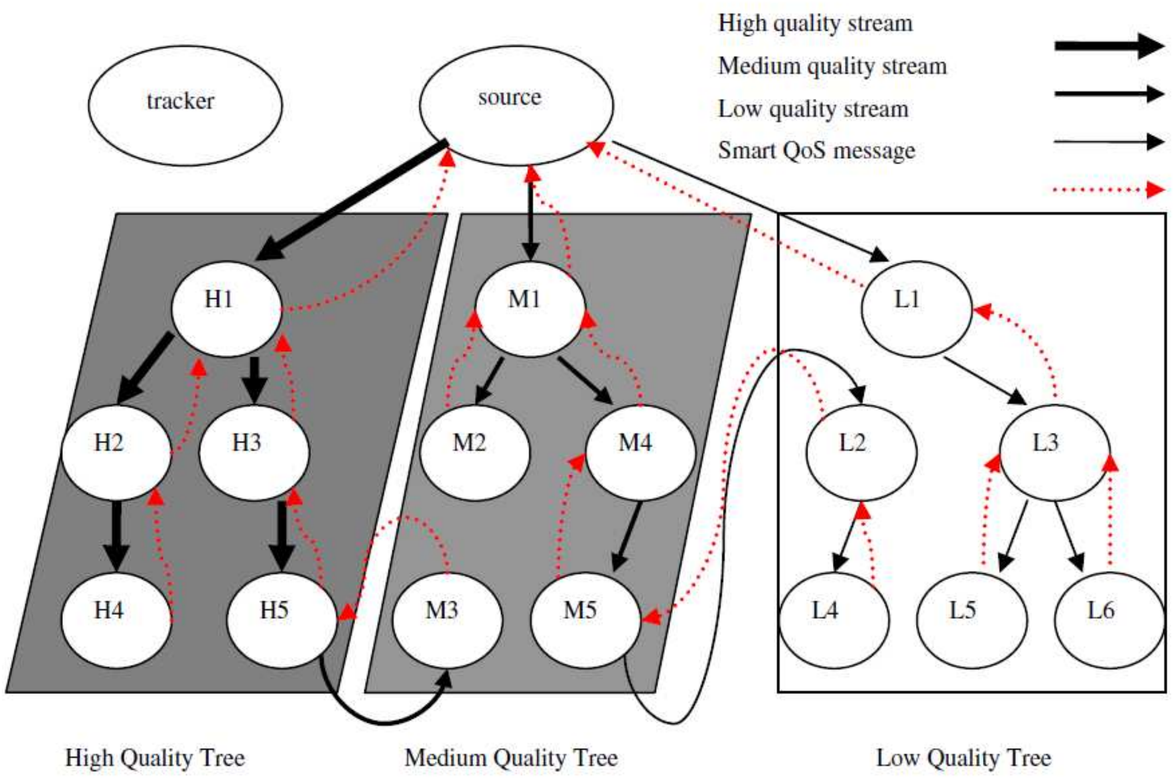

Fig 5: SmartPeerCast Overlay architecture

If any node joins then it joins with this information

(1) The new peer Pi connects to the tracker T with the message

< Peer id, uploading bandwidth, the maximum acceptable input connections, stream id>.

Every node uploading bandwidth is very important because on the basis of this bandwidth the Nodes the peer join various tree and the quality of tree is maintained according to this bandwidth.

(2) Every node can only receive the stream with the bits rate less than its uploading bandwidth. For example, if peer p uploading bandwidth is 400 Kbps, it can only receive the low quality stream with less than 400 Kbps bits rate.

(3) Peers are grouped in various clusters according to uploading bandwidth. When any node joins any tree then it searches the all peer who may be parent of this node. Parent may be from same quality tree or the high quality tree.

(4) For searching the parent node. Any node traverses the three different trees and then select parent node. First, it searches in the tree with the same quality level as itself. If all peers in this cluster are saturated and cannot be parent, it then search the peers in the trees with higher quality level after it the best suitable node is selected. First priority goes to same quality tree and then proceeds in the same way. If a peer in the same tree can be selected as the parent node, then Transrating engine does not waste time.

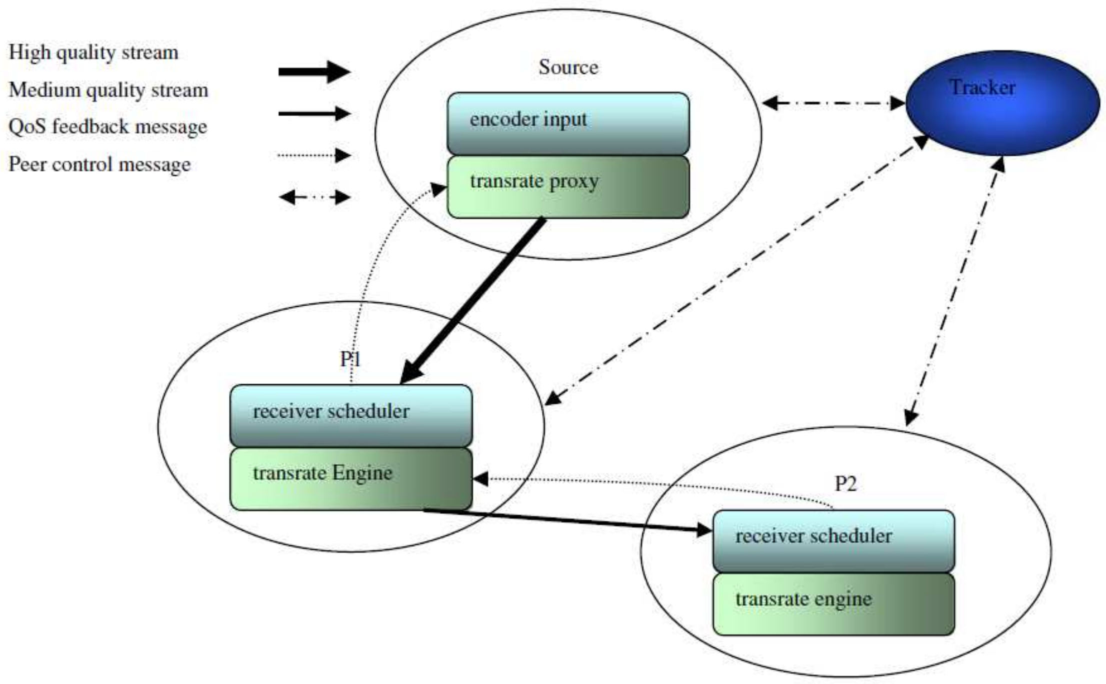

Fig 6: SmartPeerCast implementation architecture

The Smart QoS algorithm

Figure 7 describes the Smart QoS framework between the receiving and the sending peer. The receiving peer buffer is divided into three different segments. It is low water, medium, high watermark. The current playback position in the player buffer is used to make comparison with the low and the high water mark and then QoS message are sending. On the basis of these messages sending peer change the uploading bandwidth. When sending peer receives these QoS message, it adjusts its uploading bits rate dynamically by the transrating engine to reduce the playback jitters. Two types of QoS events are generated in the Smart QoS algorithm and they are send by the low and the high water mark boundary checking .There are two ways to reduce the playback jitter QoS message and transrating engine. When position changes from 2➔3 as in the Fig. 4 in that situation first QoS event is generated. It happens when the current playback position is changing from the high water buffer area to the normal buffer area. This QoS event tells that there is a bandwidth bottleneck when P_j is uploading the stream to P_i because the P_j's data forwarding speed is less slow than the Pi's playing speed and, the data present in buffer is reducing rapidly. When the sending peer Pj receives this QoS message, it uses transrating engine to reduce the forwarding stream's bits rate. And then the packet loss and the playback jitter in P_i decreases. The other QoS message is generated when the current playback position is changing from the normal buffer area to the high water buffer area and switches from 1➔2 shown in the Fig. 4. This second type QoS event indicates that P_j's data forwarding speed is more than the Pi's playing speed. P_j increases the stream quality by the transrating engine and it improves the P_i's playback quality. The transrating engine is used here to fully utilize the network bandwidth to and to provide better stream quality.

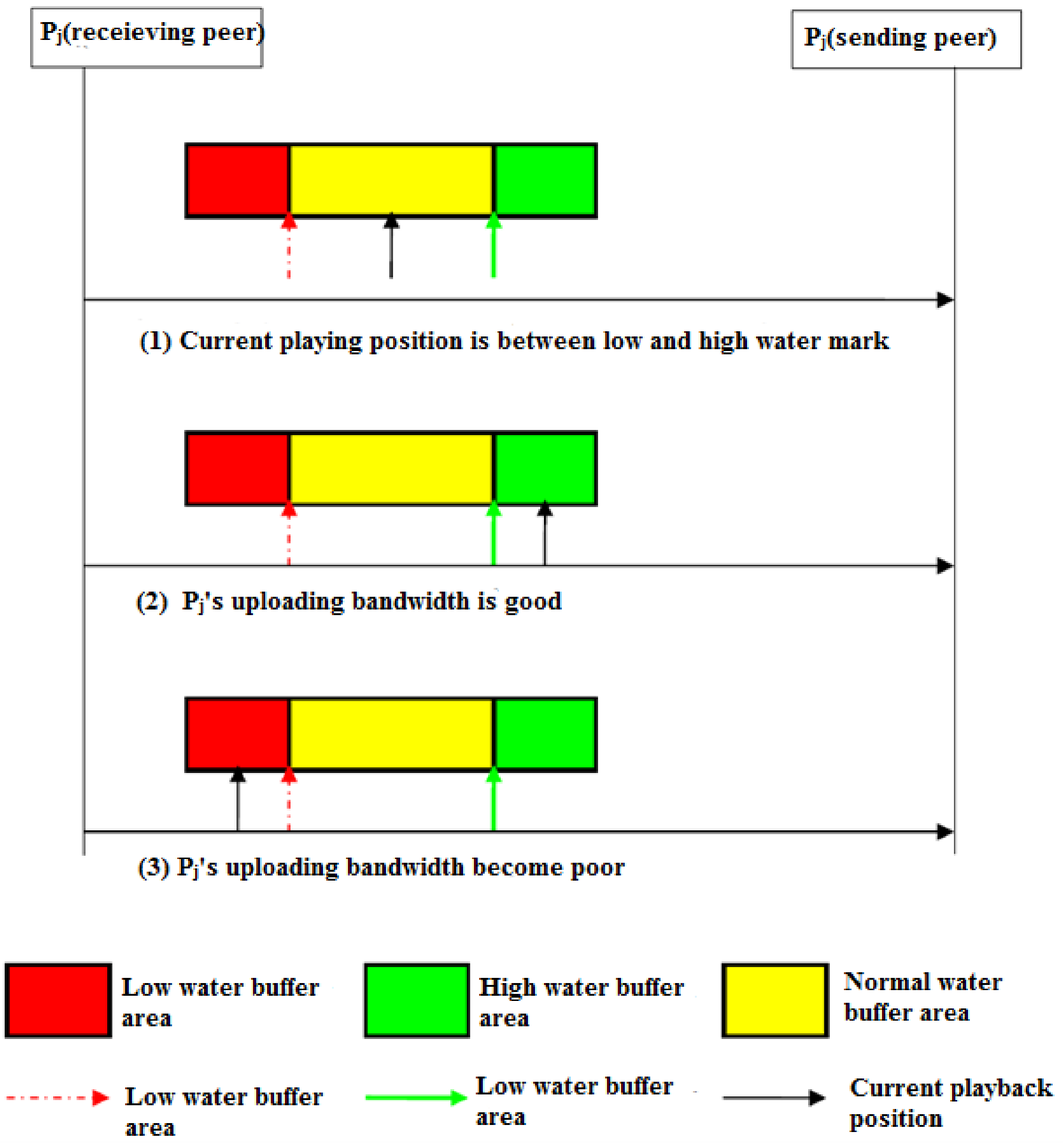
Pj(receieving peer)
Pj(sending peer)
(1) Current playing position is between low and high water mark
(2) Pj's uploading bandwidth is good
(3) Pj's uploading bandwidth become poor
Low water buffer area
High water buffer area
Normal water buffer area
Low water buffer area
Low water buffer area
Current playback position

Fig 7: SmartPeerCast QOS architecture

The receiver scheduler algorithm:

This algorithm runs at the receiving peer after a fixed interval. The receiving peer tells to tracker about the sending peer performance and then tracker change the parent node. When parent cannot continue to provide QoS then tracker change the parent node The Bandwidth Sharing Index (BSI) in SmartPeerCast is used to make comparison between forwarding quality difference between the sending and the receiving peer.

BSI is defined as:

$$BSI= T_{lWM} / T_{Total};$$

T_{lWM} tells the total time that the current playback position is below the low water mark during the playing in Pi. T_{Total} presents the total streaming time between Pj and Pi. BSI is defined as the time when Pi spends in playing the low quality stream in respect to its total playback time. According to this definition, as BSI value between P_i and increases, link transferring quality becomes worse. The quality is worse because of network congestion. A constant BSI threshold value is taken and the comparison is made with this threshold and decides parent must be change or not. If value is larger than node should contact to tracker to change its parent. If node is in high quality level tree then it may be thrown to the low quality tree or may be thrown out of system. This algorithm helps in achieving high quality video.

TABLE 2: Notations for algorithm

TERMS	**DEFINITION**
T	The tracker node
S	The streaming source node
P_i	Peer
Peers clusters[i]	It is the array of the peer clusters. And then luster the peer according to uploading bandwidth in three different trees. These tree are having different quality level
Q_i	It describes quality level of any peer. There are three quality levels as HIGH_QUALITY, MEDIUM_QUALITY and LOW_QUALITY
sb_{sid}	The bandwidth required to high quality for the streaming of sid.
Ubi	Uploading bandwidth of any given peer.
Sid	It represents real time live streaming id... All the three output streaming trees in the source node have the same sid.

4.2 ALGORITHMS:

SmartPeerCast algorithms are described as follows:

4.2.1 NEW PEER JOIN

```
Join Overlay Tree (T, Sid)
 {
        1. Peer registers itself to the tracker node by providing its uploading bandwidth,
           stream id.
        2. Search for the parent node on the basis of uploading bandwidth.
        3. It searches from current quality level to high quality level. First preference goes to
           same quality tree then increases upward according to the increasing quality tree.
        4. It finds the best suitable node to be parent.
        5. It handshake with that node and transmission starts.

}
```

4.2.2 QUALITY OF SERVICE

L: the peer playing buffer size in bytes

LWM: it indicates low watermark. It is on the 20 percentage of total buffer size.

HWM: it indicates high watermark. It is on the 80 percentage of total buffer size.

p: the current playback position in playing buffer L

T_{LWM}: The total time when peer current playback position is less than the low watermark position.

T_{HWM}: The total time when peer current playback position is more than the high watermark position.

1. First initialize the buffer till the high watermark position. Start up depends on initial buffering.

```
While (p <= HWM)
{
        Receive packet from peer $P_j$
        Play the packet
        If (p <= LWM)

        {
            1. Current buffer position is changing to low mark buffer.
            2. A QoS message is generate to reduce the stream quality and sending
               node recues    the stream quality.
            3. $T_{LWM}$ ++ ;
        }
        If (p >= HWM)
        {
            1. Current buffer position is changing to high mark buffer.
            2. A  QoS message is generate to increase the stream quality and
               sending node reduces the stream quality.
            3. $T_{LWM}$ ++ ;
        }
    }
```

4.2.3 PEER INCENTIVE

t: it is a constant and fixed to 10 sec.

T total : the parent node's node total streaming time

BSI: the bandwidth sharing index between sending and receiving node.

Jitter threshold: it indicates a limited number of playback jitters. If jitters are more than this value in any link then quality is decreasing

```
Algo(T, $P_j$)
{
    Calculate the BSI between sending and receiving node.
    BSI = $T_{LWM} / T_{TOTAL}$;
    If (BSI >= jitter threshold)
    {
        1. Tracker decides to leave the connection and change the parent node on the basis
           of the BSI value.
        2. Now node search for the new parent node.
        }

Else
        {
                1.  Tracker  upward the node in high quality tree on the basis of BSI value

        }
}
```

4.3 PROPOSED OVERLAY ARCHITECTURE:

In this thesis a new architecture is proposed with different ALM tree architecture in which we place the peer in the tree according to tree level. The peer with different forwarding bandwidth is placed on the different levels. Peers having higher forwarding bandwidth are placed on the top of tree or level 0. And in the same manner the node in the middle of tree are having medium forwarding bandwidth. And the leaf nodes have low forwarding bandwidths. Rather than replicating the video stream on multiple trees in this architecture we have only single tree so we do not need replication of video stream on multiple tree.

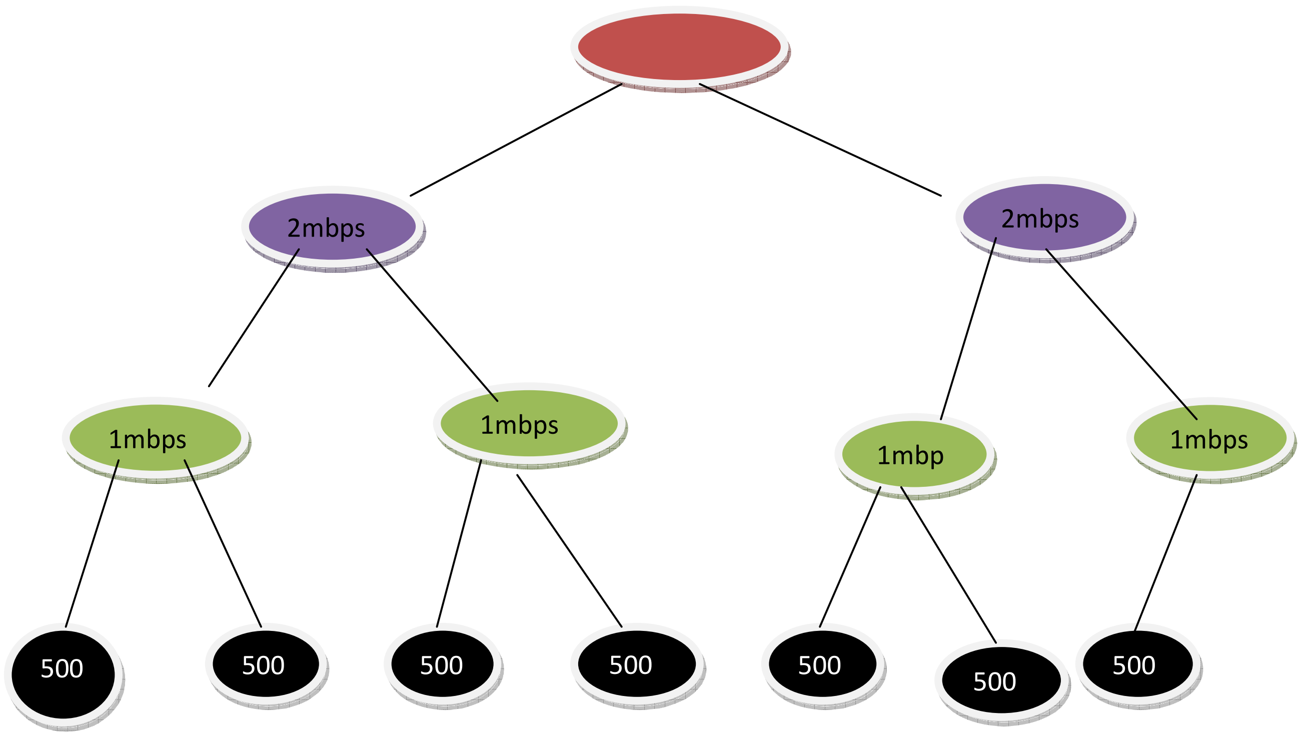

Fig 8: Proposed overlay architecture

And it also reducing the communication delay which is taken by translating engine while uploading the stream from one tree node to another tree node. In this way we optimally utilize the nodes bandwidths.

Blue node: these are high bandwidth nodes (2 Mbps).

Green node: these are medium bandwidth nodes (1 Mbps).

Black node: these are low bandwidth leaf nodes (500 Kbps).

CHAPTER 5

SIMULATION OF SMARTPEERCAST

5.1 SIMULATION INTRODUCTION

Simulation is defined as the experiment with a model to determine about the dynamic behavior of a system. Instead of experimenting with the system, the experiments are done with a model of the system. Simulation is done when real system is very critical or experiment on that system is much expensive. For simulation we create same system as the real system. It does not need so many resources.

There are three categories of simulation models, defined by the way the system state changes:

Continuous: the state changes continuously with time. Such systems are usually described by sets of differential equations.

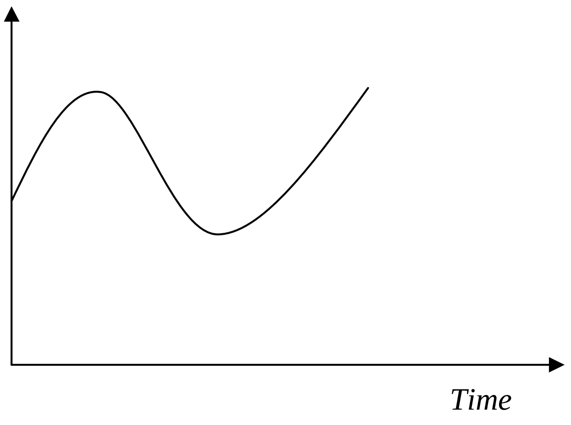

Fig 9 continuous simulation model

Discrete: the state changes only at discrete instances of time or after some events occurs.

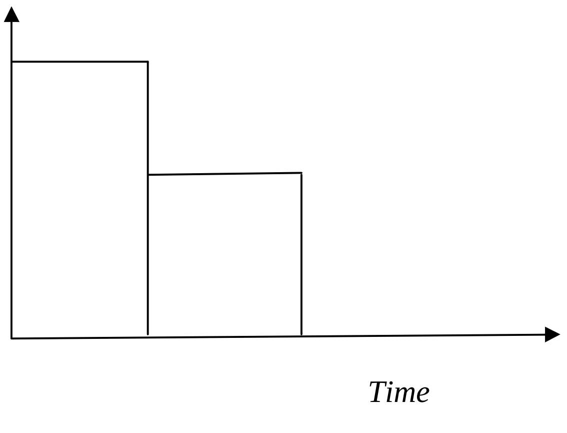

Fig 10 discrete simulation model

Combined continuous and discrete:

The state varies instantaneously at event times. But in between consecutive event times the system state may changes continuously.

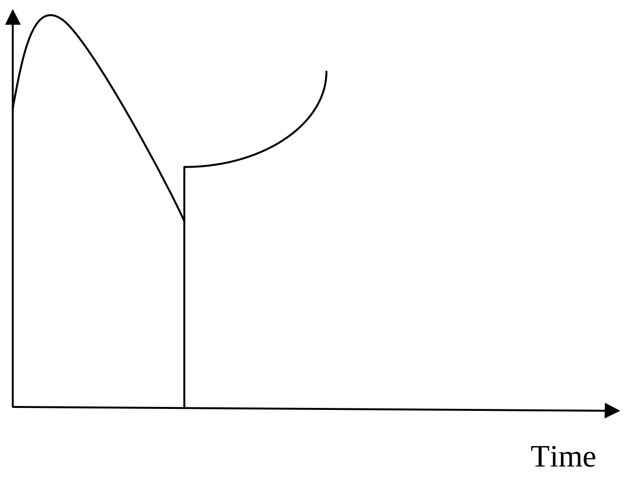

Fig 11 continuous and discrete simulation model

2.2 Three approaches for discrete event simulation

There are basically three methods for discrete event simulation:

1. The *event-based*
2. The *activity-based* and
3. The *process-based* approach.

(1) The event-based model

In the event-based model there are a collection of events. Each event is having some time to happen and some action to perform if event happens. It is simple simulation model and can be easily implemented in any language. In this thesis it is implemented in java.

(2) The activity-based approach

In the activity-based approach there are numbers of activities in the model. Each activity has some action to perform. Each activity is having some starting condition, action to be performed.

(3) The process-based approach

In the process-based approach the model consists of a collection of processes. Each process is timely linked to each other. Processes are just like the real world problems. So this approach is good for simulation but it is difficult to implement in any language. In this thesis work will only consider discrete event simulation. In discrete event simulation the model are discrete and state changes according to event happening.

All the peers in various tree wait for event occurring. When all peer finished their work then the simulation time is increased. In each time every peer do something either it is receiving the stream or it is forwarding the stream or for that time it is sitting idle. It sends such information to time controller unit. That increases the time after getting response from all peers.

SmartPeerCast Module:

Simulation of SmartPeerCast is done in java language.

There are various modules in the simulation

1. Node Insert Module
2. Node Remove Module
3. Packet Communicate Module
4. Packet Loss Module
5. Tree Display Module
6. Time controller Module

Node Insertion Module:

The network is heterogeneous in nature. So we create three trees according to the quality
And upload bandwidths. The node may insert in tree according to their bandwidths. Tree is dynamic and node always keeping on joining and removing from the tree. Every peer has its receiving and displaying buffer. Display rate is different for every peer. According to the rate every peer displays data.

Node Remove Module:

Peer may leave network at any time so the availability of stream is affected by this dynamically nature of tree. So if any node leaves the tree then its child’s parent must be changed and tree must be reconstructed.

Packet Communicate Module:

Packet are delivered from any node to its child nodes .and every node store those packets in its own buffer .when its sending time reach then it start to send packet to its child nodes. Every node has different sending time according to its uploading bandwidths. In this simulation 30 frames per second are generated by the source node. There are total of 1500 packet, which are continuously delivered by source node to its child’s node. There is some packet loss.

Packet Loss Module:

Peers are heterogeneous in nature and network is not too much reliable. So packet loss occurs while transmitting the packet. The packet loss is simulated with the help of random number generation. Every peer has different packet loss according to their uploading bandwidth. So leaf node has more packet loss. In each link between any two peers the packet loss is different. It is very critical to measure the packet loss analysis.

Tree Display Module:

This module basically shows the architecture of three different quality trees .It shows the position of node in tree. If any node joins or leaves the system then this module shows tree architecture.

Time controller Module:

It manages the time in the system .when the node communicates in the network then they have different time to deliver the packet and different show time to display the packet. When display buffer size reaches beyond the threshold then it stars up to display the packet. Time is simulated here on the basis of response of every peer. When all peer finishes their task then time is increased by one. It is main module of the architecture.

5.2 IMPLEMENTATION:

```
Package vtre2;

Import java.io.*;

Import java.util.Random;

Import java.util.Vector;

Class packet

{

    int id;

    long gentime;

    long sendtime;

    long recvime;

    int length;

    String s;

    packet(int id,long timestamp,int length)

     {

        this.id=id;

        this.gentime=timestamp;

        this.length=length;

     }

}

class Node

 {

  int fb;

  int maxConnection;

 public int iData;
 public Vector  send;
 public Vector  recieve;
 public Vector display;
 public  int c1=0;
public int c2=0;
public int disp=0;
public  int time=0;
 public int totaltime=0;
 public int lost=0;
 public  int ctr=1;
 public int[]x=new int[15];
 public int lst=0;
 public int snd=0;
 public Node(int key,Vector x,Vector  y,Vector  z)
           {
         iData = key;
         send=x;
         recieve=y;
         display=z;
         fb=key;
            }
 public int getKey()
      { return iData; }
 public void setKey(int id)
      { iData = id; }
```

```
   }
class Heap
{
public static Node[] heapArray;
public boolean[] flag;
public static int tim=0;
public   int r=0;
public int maxSize;
private int currentSize;
public Heap()
{}
public Heap(int mx)
{
      maxSize = mx;
      currentSize = 0;
      heapArray = new Node[maxSize];
      flag=new boolean[maxSize];
      }
 public boolean isEmpty()
      { return currentSize==0; }


  public boolean insert(int key)
      {
      if(currentSize==maxSize)
         return false;
```

```
    Vector x = new Vector(150);
     Vector y = new Vector(150);
     Vector z = new Vector(150);
     Node newNode = new Node(key,x,y,z);
     heapArray[currentSize] = newNode;
     trickleUp(currentSize++);
     return true;
     }

 public void trickleUp(int index)
 {
 int parent = (index-1) / 2;
 Node bottom = heapArray[index];
 while( index > 0 && heapArray[parent].getKey() < bottom.getKey())
        {
        heapArray[index] = heapArray[parent];
        index = parent;
        parent = (parent-1) / 2;
        }
  heapArray[index] = bottom;
     }
public void communicate(packet p[],int i,int f) throws NullPointerException
{
  if(f==1)
 {
```

```
 for(int h1=0;heapArray[h1]!=null;h1++)
 {
 int v1= heapArray[h1].iData;
 int sec1= (512*8*10)/v1;
 heapArray[h1].time=sec1;
 if(heapArray[2*h1+1]==null&&heapArray[2*h1+2]==null)
 {
  System.out.println("this is leaf node"+heapArray[h1].iData+" can'tpass data ");
  }
  else
  { System.out.println("this is low quality tree frame per second generated by source: "+10);
         System.out.println("time to take:"+heapArray[h1].iData+"is:"+ heapArray[h1].time);
   }
    flag[h1]=true;
     ++r;
        }
    }
 for(int k=0;k<30;k++)
 {
heapArray[i].recieve.add(p[k]);
System.out.println(heapArray[0].iData+"is generating the packet id:"+ p[k].id +"at time"+f);
   }
     if(f!=2)
    {
    int p3=  timr();
    for(int h3=0;heapArray[h3]!=null;h3++)
     {
         try
         {
         th t =new th(h3,p3);
         Thread  t1 = new Thread(t);
          t1.start();
          }catch(Exception e){}
        }
    }
    else
    {
 int m =r-1;
 while(heapArray[m].lst<2)
    {
    int p3=  timr();
    for(int h2=0;heapArray[h2]!=null;h2++)
        try
         {
         th t =new th(h3,p3);
        Thread  T2 = new Thread(t);
         T2.start();
         }catch(Exception e){}
     }
      }
    }
heapArray[0].ctr++;
   }
public synchronized int timr()
{
int p2=1;
 for(int p1=1;p1!=r;p1++)
 {
if(flag[p1]==false)
 p2=0;
  }
 if(p2==1)
  return (++tim);
  else
  return (tim);
}
 public void lost()
{
for(int h=1;heapArray[h]!=null;h++)
 {
  for(int i=0;i<=1;i++)
    {
 System.out.println("total lost between node"+heapArray[h].iData+"and"+heapArray[(h-1)/2].iData+"is"+heapArray[h].x[i]+"at times "+i);
    }
```

```
}
}
 public void displayHeap()
  {
for(int m=0; m<currentSize; m++)
 if(heapArray[m] != null)
  {
   System.out.print( heapArray[m].getKey() + " ");
   }
    else
    System.out.print( "-- ");
    System.out.println();


    int nBlanks = 32;
   int itemsPerRow = 1;
   int column = 0;
   int j = 0;
   String dots = "...............................";
   System.out.println(dots+dots);
    while(currentSize > 0)
     {
      if(column == 0)
      for(int k=0; k<nBlanks; k++)
              System.out.print(' ');
 System.out.print(heapArray[j].getKey());
```

```
   if(++j == currentSize)
   break;
   if(++column==itemsPerRow)
   {
     nBlanks /= 2;
     itemsPerRow *= 2;
     column = 0;
      System.out.println();
     }
     else
    for(int k=0; k<nBlanks*2-2; k++)
    System.out.print(' ');
        }
      System.out.println("\n"+dots+dots);
      }
   }

class th extends Heap implement Runnable
{
int h;
int f;
    th(int index,int f)
{h=index;
 this.f=f;
}
```

```
public void  run()

{

if(heapArray[h].recieve.size()>heapArray[2*h+1].recieve.size()||heapArray[h].recieve.size()>hea
pArray[2*h+2].recieve.size())

{

  if(heapArray[2*h+1]==null&&heapArray[2*h+2]==null)

   {

  }

 else

{

  if(h>0)

 {

  if(heapArray[h].c2==0)

   {

    heapArray[h].totaltime=heapArray[(h-1)/2].time+heapArray[h].time;


   }

   }

 if(h==0)

{

  if(heapArray[h].c2==0)

    heapArray[h].totaltime=heapArray[h].time;

    }

int s=heapArray[h].recieve.size()-heapArray[h].send.size();

 if(heapArray[h].totaltime==f&&s>0)
 {
 try
{ int temp1=0;
 int temp2=0;
 if(heapArray[2*h+1]!=null)
 {
   temp1=heapArray[2*h+1].lst;
   }
   if(heapArray[2*h+2]!=null)
   {
    temp2=heapArray[2*h+2].lst;
    }
System.out.println("hi i "+heapArray[h].iData+"lost is"+heapArray[h].x[heapArray[h].snd]+"at
temp"+heapArray[h].snd);
for(int i=0;(i!=(30-heapArray[h].x[heapArray[h].snd]));i++)
{
  packet d=(packet)heapArray[h].recieve.get(heapArray[h].c1);
  heapArray[h].send.add(d);
   if(heapArray[2*h+1]!=null)
  {
System.out.println(heapArray[2*h+1].iData+"is recieving the packet at"+f);
  Random randomGenerator = new Random();
   int randomInt = randomGenerator.nextInt(100);
  if(randomInt >5)
{
 System.out.println("packettid:"+d.id+"send
by"+heapArray[h].iData+"to"+heapArray[2*h+1].iData);

 heapArray[2*h+1].recieve.add(d);

heapArray[2*h+1].display.add(d);

 System.out.println("packetid:"+d.id+"recieve
by"+heapArray[2*h+1].iData+"from"+heapArray[h].iData);

System.out.println("packet id:"+""+d.id);

System.out.println("packet length:"+""+d.length);

  }

else{

System.out.println("packet lost:"+d.id+"at node"+heapArray[2*h+1 ].iData);

 (heapArray[2*h+1].x[temp1])=(heapArray[2*h+1].x[temp1])+1;

 System.out.println("p lost"+"atindex"+temp1+"is"+(heapArray[2*h+1].x[temp1]));

        }

   if(heapArray[2*h+2]!=null)

   {

    System.out.println(heapArray[2*h+2].iData+"is recieving the packet at"+f);

     Random randomGenerator = new Random();

    int randomInt = randomGenerator.nextInt(100);

    if(randomInt >5)

    {

  System.out.println("packetid:"+d.id+"sendby"+heapArray[h].iData+"to"+

 heapArray[2*h+2].iData);

   heapArray[2*h+2].recieve.add(d);

  heapArray[2*h+2].display.add(d);

 System.out.println("packetid:"+d.id+"recieveby"+heapArray[2*h+2].iData+"from"+
heapArray[h].iData);
 System.out.println("packet id:"+""+d.id);
System.out.println("packet length:"+""+d.length);
 }
 else{
System.out.println("packet lost:"+d.id+"at node"+heapArray[2*h+2].iData);
 (heapArray[2*h+2].x[temp2])=(heapArray[2*h+2].x[temp2])+1;
  System.out.println("p lost"+"at index"+temp2+"is"+(heapArray[2*h+2].x[temp2]));
        }
  }
   heapArray[h].c2++;
   heapArray[h].c1++;
        }
 }catch(Exception e){ }
heapArray[h].totaltime=heapArray[h].totaltime+1;
heapArray[h].snd=heapArray[h].snd+1;
if(heapArray[2*h+1]!=null)
heapArray[2*h+1].lst=heapArray[2*h+1].lst+1;
if(heapArray[2*h+2]!=null)
heapArray[2*h+2].lst=heapArray[2*h+2].lst+1;
}
else
{
if(heapArray[h]!=heapArray[0])
 System.out.println(heapArray[h].iData+"is idle"+"at time"+f);
 else if(f>30)
  System.out.println(heapArray[h].iData+"is idle"+"at time"+f);
      }
 }
}
  else
  {
   if(heapArray[h]!=heapArray[0])
   System.out.println(heapArray[h].iData+"is idle"+"at time"+f);
   else if(f>30)
   System.out.println(heapArray[h].iData+"is idle"+"at time"+f);
      }
display();
}
 public void display()
    {
if(heapArray[h].display.size()%5==0&&heapArray[h].display.size()!=0&&heapArray[h].display
.size()>=heapArray[h].disp+5)
 if(heapArray[2*h+2]!=null)
  {
  int y=heapArray[2*h+2].recieve.size();
  if(heapArray[2*h+2].disp<=y)
   {
   while(heapArray[2*h+2].disp!=y)
   {
```

```
  packet j=(packet)heapArray[2*h+2].display.get(heapArray[2*h+2].disp);

 System.out.println(heapArray[2*h+2].iData+"is showing its display buffer packet id:"+j.id+"at
time"+f);

  heapArray[2*h+2].disp++;

           }

       }

   }

 if(heapArray[2*h+1]!=null){

 int z= heapArray[2*h+1].recieve.size();

 if(heapArray[2*h+1].disp<=z)

 {

 while(heapArray[2*h+1].disp<=z)

 {

 packet j=(packet)heapArray[2*h+1].display.get(heapArray[2*h+1].disp);

System.out.println(heapArray[2*h+1].iData+"is showing its display buffer packet id:"+j.id+"at
time"+f);

 heapArray[2*h+1].disp++;

   }

 }

 }

flag[h]=true;

}

}

class Main extends Thread

{

static int id=0,k=0;
static int counter=0;
public static void main(String[] args) throws IOException
{
 int value;
 Heap low = new Heap(150);
 Heap medium = new Heap1(150);
  Heap high = new Heap2(150);
  boolean s1,s2,s3,success;
   while(true)
   {
   System.out.print("Enter first letter of: ");
   System.out.print("show, intialize,communicate,lost enquary:: ");
   int choice = getChar();
    switch(choice)
    {
    case 's':
    System.out.print("\n\n ");
    System.out.print("heapArray1: ");
    low.displayHeap();
   System.out.print("\n\n ");
    System.out.print("heapArray2: ");
     medium.displayHeap();
     System.out.print("\n\n ");
     System.out.print("heapArray3: ");
     high.displayHeap();
     System.out.print("\n\n ");
      break;
     case 'c': k++;
     for(int g=0;g<2;g++)
     {
     packet[] d = new packet[30];
     for(int k=0;k<30;k++)
    {
     packet p= new packet(++id,System.currentTimeMillis(),512);
     d[k]=p;
     }
    try{
     ++counter;
    try
    {
    low.communicate(d,0,counter);
     }catch(Exception e){};
    try
    {
     medium.communicate(d,0,counter);
     }catch(Exception e){};
  try
  {
   high.communicate(d,0,counter);
 }catch(Exception e){};
```

```
 }
 catch(Exception e){ };
  }
break;
case 'i':
boolean y;
for(int i=0;i<4;i++)
   {
 y=low.insert(512);
 y=medium.insert(1024);
 y=high.insert(2048);
   }
break;
case 'l':
 low.lost();
 medium.lost();
 high.lost();
 break;
default:
  System.out.println("Invalid entry\n");
           }
         }
      }
public static String getString() throws IOException
```

```
{
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String s = br.readLine();
    return s;
 }

  public static char getChar() throws IOException
    {
    String s = getString();
    return s.charAt(0);
    }
  public static int getInt() throws IOException
    {
    String s = getString();
    return Integer.parseInt(s);
    }
}
```

5.2 RESULT ANALYSIS:

It shows that data lost at the leaf node is higher than the intermediate node. In the simulation user are asked to enter the peer uploading bandwidth. According to that system keeps that node in the tree. There are three trees according to the quality of stream. There are 7 nodes in the system. Three nodes are in high quality tree, two nodes are in middle quality and two nodes in low level tree. Packet loss is shown by random number generator. Average packet loss in low tree is one packet per five packet transmitted, in middle two packet per ten packet transmitted and in high level one packet per ten packet transmitted This is result is simulated for only 10 packets and 5 frames per second are generating. The node has different delivery time and different display time on the basis of bandwidth. The packet loss is shown on various links among peer.

```
run:
Enter first letter of: show, insert, communicate ,lost enquiry:: i
Enter value to insert: 5
Enter first letter of: show, insert, communicate ,lost enquiry:: i
Enter value to insert: 10
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i
Enter value to insert: 15
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i
Enter value to insert: 20
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i
Enter value to insert: 35
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i
Enter value to insert: 45
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i
Enter value to insert: 55
Enter first letter of: show, show, insert, communicate ,lost enquiry:: i

Enter value to insert: 65

Enter first letter of: show, show, insert, communicate ,lost enquiry:: s


 heapArray1: 20 15 10 5

..............................................................

                                20

                15                                  10

       5

..............................................................


 heapArray2: 45 35

..............................................................

                                45

                35

..............................................................


 heapArray3: 65 55

..............................................................

                                65

                55

..............................................................


 Enter first letter of: show, show, insert, communicate ,lost enquiry:: c

time to take:20is:6
time to take:15is:8
this is leaf node10 can’t pass data
this is leaf node5 can’t pass data
20is generating the packet id:1at time1
20is generating the packet id:2at time1
20is generating the packet id:3at time1
20is generating the packet id:4at time1
20is generating the packet id:5at time1
time to take:45is:2
this is leaf node35 can’t pass data
45is generating the packet id:1at time1
45is generating the packet id:2at time1
45is generating the packet id:3at time1
45is generating the packet id:4at time1
45is generating the packet id:5at time1
time to take:65is:1
this is leaf node55 can’t pass data
65is generating the packet id:1at time1
65is generating the packet id:2at time1
65is generating the packet id:3at time1
65is generating the packet id:4at time1
65is generating the packet id:5at time1
hi i 65
hi i 65lost is0at temp0
55is recieving the packet at1
```

```
packet id:1send by65to55

packet id:1recieve by55from65

packet id:1

packet length:512

55is receiving the packet at1

packet id:2send by65to55

packet id:2recieve by55from65

packet id:2

packet length:512

55is receiving the packet at1

packet id:3send by65to55

packet id:3recieve by55from65

packet id:3

packet length:512

55is receiving the packet at1

packet id:4send by65to55

packet id:4recieve by55from65

packet id:4

packet length:512

55is receiving the packet at1

packet id:5send by65to55

packet id:5recieve by55from65

packet id:5

packet length:512

55is showing its display buffer packet id:1at time1
```

```
55is showing its display buffer packet id:2at time1

55is showing its display buffer packet id:3at time1

55is showing its display buffer packet id:4at time1

55is showing its display buffer packet id:5at time1

20is generating the packet id:6at time2

20is generating the packet id:7at time2

20is generating the packet id:8at time2

20is generating the packet id:9at time2

20is generating the packet id:10at time2

hi i 20lost is0at temp0

15is receiving the packet at6

packet id:1send by20to15

packet id:1recieve by15from20

packet id:1

packet length:512

10is receiving the packet at6

packet id:1send by20to10

packet id:1recieve by10from20

packet id:1

packet length:512

15is receiving the packet at6

packet id:2send by20to15

packet id:2recieve by15from20

packet id:2

packet length:512
```

```
10is recieving the packet at6

packet id:2send by20to10

packet id:2recieve by10from20

packet id:2

packet length:512

15is receiving the packet at6

packet id:3send by20to15

packet id:3recieve by15from20

packet id:3

packet length:512

10is receiving the packet at6

packet id:3send by20to10

packet id:3recieve by10from20

packet id:3

packet length:512

15is receiving the packet at6

packet id:4send by20to15

packet id:4recieve by15from20

packet id:4

packet length:512

10is receiving the packet at6

packet id:4send by20to10

packet id:4recieve by10from20

packet id:4

packet length:512
```

```
15is receiving the packet at6
packet lost:5at node15
p lostatindex0is1
10is receiving the packet at6
packet id:5send by20to10
packet id:5recieve by10from20
packet id:5
packet length:512
10is showing its display buffer packet id:1at time6
10is showing its display buffer packet id:2at time6
10is showing its display buffer packet id:3at time6
10is showing its display buffer packet id:4at time6
10is showing its display buffer packet id:5at time6
15is showing its display buffer packet id:1at time6
15is showing its display buffer packet id:2at time6
15is showing its display buffer packet id:3at time6
15is showing its display buffer packet id:4at time6
15is idle at time6
hi i 20lost is0at temp1
15is receiving the packet at7
packet id:6send by20to15
packet id:6recieve by15from20
packet id:6
packet length:512
10is receiving the packet at7
```

```
packet id:6send by20to10

packet id:6recieve by10from20

packet id:6

packet length:512

15is receiving the packet at7

packet id:7send by20to15

packet id:7recieve by15from20

packet id:7

packet length:512

10is receiving the packet at7

packet id:7send by20to10

packet id:7recieve by10from20

packet id:7

packet length:512

15is receiving the packet at7

packet id:8send by20to15

packet id:8recieve by15from20

packet id:8

packet length:512

10is receiving the packet at7

packet id:8send by20to10

packet id:8recieve by10from20

packet id:8

packet length:512

15is recieving the packet at7
```

```
packet id:9send by20to15

packet id:9recieve by15from20

packet id:9

packet length:512

10is receiving the packet at7

packet id:9send by20to10

packet id:9recieve by10from20

packet id:9

packet length:512

15is receiving the packet at7

packet id:10send by20to15

packet id:10recieve by15from20

packet id:10

packet length:512

10is receiving the packet at7

packet id:10send by20to10

packet id:10recieve by10from20

packet id:10

packet length:512

10is showing its display buffer packet id:6at time7

10is showing its display buffer packet id:7at time7

10is showing its display buffer packet id:8at time7

10is showing its display buffer packet id:9at time7

10is showing its display buffer packet id:10at time7

15is showing its display buffer packet id:6at time7
```

```
15is showing its display buffer packet id:7at time7

15is showing its display buffer packet id:8at time7

15is showing its display buffer packet id:9at time7

15is showing its display buffer packet id:10at time7

15is idle at time7

20is idle at time8

15is idle at time8

20is idle at time9

15is idle at time9

20is idle at time10

15is idle at time10

20is idle at time11

15is idle at time11

20is idle at time12

15is idle at time12

20is idle at time13

15is idle at time13

20is idle at time14

hi i 15lost is1at temp0

5is receiving the packet at14

packet id:1send by15to5

packet id:1recieve by5from15

packet id:1

packet length:512

5is receiving the packet at14
```

```
packet id:2send by15to5

packet id:2recieve by5from15

packet id:2

packet length:512

5is receiving the packet at14

packet id:3send by15to5

packet id:3recieve by5from15

packet id:3

packet length:512

5is receiving the packet at14

packet id:4send by15to5

packet id:4recieve by5from15

packet id:4

packet length:512

5is showing its display buffer packet id:1at time14

5is showing its display buffer packet id:2at time14

5is showing its display buffer packet id:3at time14

5is showing its display buffer packet id:4at time14

20is idle at time15

hi i 15lost is0at temp1

5is receiving the packet at15

packet id:6send by15to5

packet id:6recieve by5from15

packet id:6

packet length:512
```

```
5is recieving the packet at15

packet id:7send by15to5

packet id:7recieve by5from15

packet id:7

packet length:512

5is receiving the packet at15

packet id:8send by15to5

packet id:8recieve by5from15

packet id:8

packet length:512

5is receiving the packet at15

packet id:9send by15to5

packet id:9recieve by5from15

packet id:9

packet length:512

5is receiving the packet at15

packet id:10send by15to5

packet id:10recieve by5from15

packet id:10

packet length:512

5is showing its display buffer packet id:6at time15

5is showing its display buffer packet id:7at time15

5is showing its display buffer packet id:8at time15

5is showing its display buffer packet id:9at time15

5is showing its display buffer packet id:10at time15
```

```
45is generating the packet id:6at time2

45is generating the packet id:7at time2

45is generating the packet id:8at time2

45is generating the packet id:9at time2

45is generating the packet id:10at time2

hi i 45lost is0at temp0

35is receiving the packet at2

packet id:1send by45to35

packet id:1recieve by35from45

packet id:1

packet length:512

35is receiving the packet at2

packet id:2send by45to35

packet id:2recieve by35from45

packet id:2

packet length:512

35is receiving the packet at2

packet id:3send by45to35

packet id:3recieve by35from45

packet id:3

packet length:512

35is receiving the packet at2

packet id:4send by45to35

packet id:4recieve by35from45

packet id:4
packet length:512
35is receiving the packet at2
packet id:5send by45to35
packet id:5recieve by35from45
packet id:5
packet length:512
35is showing its display buffer packet id:1at time2
35is showing its display buffer packet id:2at time2
35is showing its display buffer packet id:3at time2
35is showing its display buffer packet id:4at time2
35is showing its display buffer packet id:5at time2
hi i 45lost is0at temp1
35is receiving the packet at3
packet id:6send by45to35
packet id:6recieve by35from45
packet id:6
packet length:512
35is receiving the packet at3
packet id:7send by45to35
packet id:7recieve by35from45
packet id:7
packet length:512
35is receiving the packet at3
packet id:8send by45to35
packet id:8recieve by35from45
packet id:8

packet length:512

35is recieving the packet at3

packet id:9send by45to35

packet id:9recieve by35from45

packet id:9

packet length:512

35is receiving the packet at3

packet id:10send by45to35

packet id:10recieve by35from45

packet id:10

packet length:512

35is showing its display buffer packet id:6at time3

35is showing its display buffer packet id:7at time3

35is showing its display buffer packet id:8at time3

35is showing its display buffer packet id:9at time3

35is showing its display buffer packet id:10at time3

65is generating the packet id:6at time2

65is generating the packet id:7at time2

65is generating the packet id:8at time2

65is generating the packet id:9at time2

65is generating the packet id:10at time2

hi i 65

hi i 65lost is0at temp1

55is receiving the packet at2
```

```
packet id:6send by65to55

packet id:6recieve by55from65

packet id:6

packet length:512

55is receiving the packet at2

packet id:7send by65to55

packet id:7recieve by55from65

packet id:7

packet length:512

55is receiving the packet at2

packet id:8send by65to55

packet id:8recieve by55from65

packet id:8

packet length:512

55is receiving the packet at2

packet id:9send by65to55

packet id:9recieve by55from65

packet id:9

packet length:512

55is receiving the packet at2

packet id:10send by65to55

packet id:10recieve by55from65

packet id:10

packet length:512

55is showing its display buffer packet id:6at time2
```

```
55is showing its display buffer packet id:7at time2

55is showing its display buffer packet id:8at time2

55is showing its display buffer packet id:9at time2

55is showing its display buffer packet id:10at time2

Enter first letter of: show, insert ,communicate ,lost enquiry:: l

total lost between node15and20is1at times 0

total lost between node15and20is0at times 1

total lost between node10and20is0at times 0

total lost between node10and20is0at times 1

total lost between node5and15is2at times 0

total lost between node5and15is0at times 1

total lost between node35and45is0at times 0

total lost between node35and45is0at times 1

total lost between node55and65is0at times 0

total lost between node55and65is0at times 1
```

CHAPTER 6

CONCLUSION AND FUTURE WORK

Peer to Peer video streaming is very challenging in heterogeneous network where the peer joins and remove from the tree dynamically. It is very difficult to get quality stream .Video is very much time critical and always need updated data. Transmission delay must be low. There are various overlay architectures for data transmission in video streaming but all have some merit and some demerits. SmartPeerCast is combination of tree and mesh based approach. It uses some QoS message to maintain the quality of video data. SmartPeerCast achieves a less startup time delay and a less packet loss ratio. The peers are grouped together according to their uploading bandwidths. The peers at the same ALM tree use the transrating engine to change the stream quality automatically. It sends QoS messages to change the uploading bandwidths. It reduces the packet loss ratio. It also utilizes the leaf node bandwidths. The leaf node of high quality tree may deliver the data to the medium or low quality tree nodes.

The simulation of SmartPeerCast limits to the same tree communication. The node in the same tree is able to communicate data to each other and they are able to download and upload the data. Future work is that communication among various trees must also take place.

REFERENCES AND BIBLIOGRAPHY

[1] X. Zhang, J. Liu, Bo Li, and P. Yum. CoolStreaming/ DONet: A data-driven overlay network for peer-to-peer live media streaming. Technical report, University of Hong-Kong and Vancouver, 2004.

[2]. Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, "A.: SplitStream: High-bandwidth multicast in cooperative environments". In: SOSP'03: Proceedings of the nineteenth ACM Symposium on Operating Systems Principles, New York, NY, USA (2003) 298–313.

[3]. Pai, V.S., Kumar, K., Tamilmani, K., Sambamurthy, V., Mohr, A.E.: "Chainsaw: Eliminating trees from overlay multicast". In: IPTPS. (2005) 127–140.

[4]. Magharei, N., Rejaie, R.: "PRIME: Peer-to-peer receiver-driven mesh-based streaming"
In: 26th Annual IEEE Conference on Computer Communications IEEE INFOCOM 2007. (2007).

[5]. Pianese, F., Perino, D., Keller, J., Biersack, E.: "PULSE: an adaptive, incentive based, unstructured p2p live streaming system". IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks 9(6) (2007).

[6].Jurca D., Chakareski J., Wagner J.-P., Frossard P., "Enabling adaptive video streaming in P2P systems", in *IEEE Communications Magazine*, Volume 45, Issue 6, pages 108-114, June 2007.

[7].Carra D., Lo Cigno R. and Biersack E.W., "Graph Based Analysis of Mesh Overlay Streaming Systems" , in *IEEE Journal on Selected Areas in Communications*, Volume 25, Issue 9, Pages 1667-1677 , December 2007.

[8].Liu Y., Guo Y. and Liang Ch., "A survey on peer-to-peer video streaming systems", in *Journal of Peer-to-Peer Networking and Applications*, Springer, New York, February 2008.

[9].Liu Z., Shen Y., Panwar S., Ross K. W., and Wang Y., "Efficient Substream Encoding for P2P Video on Demand", in *Packet Video 2007*, pages 143-152, 12-13 November 2007.

[10].Xu D., Kulkarni S. S., Rosenberg C. and Chai H. K., "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution", in *Multimedia System*, volume 11, number 4, pages 383-399, 2006.

[11].Pianese F., Perino D., Keller J., and Biersack E.W., “PULSE: An Adaptive, Incentive-Based,Unstructured P2P Live Streaming System ”, in *IEEE Transactions on Multimedia*, Volume 9, Number 8, pages 1645-1660, December 2007.

[12].Magharei N. and Rejaie R., "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches", in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007)*, Anchorage, Alaska, pages 1424-1432, 6-12 May 2007.

[13].Magharei N. and Rejaie R., “PRIME: Peer-to-peer Receiver driven mesh-based streaming”, in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM 2007)*, Anchorage, Alaska, USA, pages 1415-1423, 6-12 May 2007.

[14].Annapureddy S., Guha S., Gkantsidis Ch., Gunawardena D. and Rodriguez P., “Is High-Quality VoD Feasible using P2P Swarming?” in Proceedings of the 16th international conference on World Wide Web (WWW '07), Banff, Alberta, Canada, pages 903-912, 8-12 May 2007.

[15] http://www.planet-lab.org.

[16] M. Handley, S. Floyd, and J. Widmer. RFC 3448: TCP Friendly Rate Control. Technical report, IETF, 2003.

[17] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. Data-driven overlay streaming: Design, implementation and experience. Technical report, University of Hong-Kong and Vancouver, 2004.

[18] http://www.bittorrent.org.

[19] Carnegie Mellon University. End System Multicast. http://esm.cs.cmu.edu, 2005.

[20] Yank hua Chu, Aditya Ganjam, T.S. Eugene Ng, and Sanjay G. Rao. Early experience with an internet broadcast system based on overlay multicast. Technical report, Carnegie Mellon University, 2004.

[21] http://www.sparknet.fi.

[22] Xuxian Jiang , Yu Dong , Dongyan Xu , Bhargava, B. , “ GnuStream: a P2P media streaming system prototype”IEEE multimedia conference ICME 2003.

[23] Xuxian Jiang, Yu Dong, Dongyan Xu, and Bharat Bhargava. GnuStream: A P2P media streaming system prototype. Technical report, Purdue University, 2003.

[24] http://www.hypercast.org.

[25] Jin Li. PeerStreaming: A practical receiver-driven peer-to-peer media streaming system. Technical report, Microsoft, 2004.

[26] A. Nicolosi and S. Annapureddy. P2PCast: A peer-to-peer multicast scheme for streaming data. Technical report, University of New York, 2003.

[27] ETH Zurich. Swistry. http://dcg.ethz.ch/projects/swistry, 2006.

[28].Jiang X, Dong Y, Xu D, Bhargava B (2003) GnuStream: a P2P media streaming system prototype. ICME apos:03. Proceedings 2003 International Conference on Multimedia and Expo. IEEE, Piscataway, USA, vol. 2, Issue 6–9, pp II - 325–8.

[29]Chu YH, Rao SG, Seshan S, Zhang H (2002) A case for end system multicast. IEEE J Sel Areas Commun 20(8):1456–1471.

[30]. Jurca D, Chakareski J, Wagner J-P, Frossard P (2007) Enabling adaptive video streaming in P2P systems. IEEE Commun Mag 45(6):108–114.

[31]Magharei N, Rejaie R, Guo Y (2007) Mesh or multiple-tree: a comparative study of live P2P streaming approaches. IN-FOCOM 2007, 26th IEEE International Conference on Computer Communications.

[32] Z.Liu, Y.Shen, S.Panwar "Efficient substream encoding and transmission for P2P video on demand" p-p 143 Packet video 2007.

[33]. Liu JC, Rao SG, Li B, Zhang H (2008) "Opportunities and challenges of peer-to-peer Internet video broadcast". Proceedings of the IEEE. IEEE, Piscataway, USA, vol. 96, no. 1, pp 11–24 IEEE, Piscataway, USA, pp 1424–1432.

[34] Y.Zhu,M.You,W.Shu "comparison study and evaluation of overlay multicast networks" p.p 493-6 vol.3 multimedia and expo ICME 2003.

[35] S.Tae Kim, R. Konda, P.Mah "Adaptive mode decision algorithm for inter layer coding in scalable video coding" 17 international IEEE conference on Image processing (ICIP) p-p 1297-1300 2010 .

[36]. http://www.peercast.org

[37]. http://www.ppstream.com

[38]. http://www.youtube.com

[39].W. Wang,Y. Chen "SmartPeerCast: a Smart QoS driven P2P live streaming framework" MultimediaTools Appl Springer Science+Business Media, LLC 2010.

[40].V.K. Goyal, “Multiple Description Coding: Compression meets the network”, IEEE Signal Processing Magazine, pp.74 – 93, September.2001.

[41]. X.Zhang,H.Hassenein “Video On-Demand Streaming on the internet a survey ” 25th Biennial Symposium on Communications 978-1-4244-5711-3 2010 IEEE.

[42] N. Magharei, Y.Guo “Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches” pp 1424-1432 IEEE INFOCOM 2007.

[43] Z.Chen,C.Lin,X.Wei “Enabling on-demand Internet Video Streaming Services to Multi-terminal Users in Large Scale” IEEE Transactions on Consumer Electronics, Vol. 55, No. 4, NOVEMBER 2009.

[44] P. Giacomazzi, A. Poli “Performance analysis of peer-to-peer video streaming systems with tree and forest topology” 14th IEEE International Conference on Parallel and Distributed Systems 2008.

[45] Z.Lu, Y.Li, J.Wu “MultiPeerCast: A Tree-mesh-hybrid P2P Live Streaming Scheme Design and Implementation based on PeerCast” The 10th IEEE International Conference on High Performance Computing and Communications 2008.

[46]. Pianese F, Perino D, Keller J, Biersack EW (2007) PULSE: an adaptive, incentive-based, unstructured P2P live streaming system. IEEE Trans Multimedia 9(8):1645–1660.

[47].Liu Y, Guo Y, Liang C (2008) “A survey on peer-to-peer video streaming systems. Peer-to-Peer Networking and Applications”. Springer, New York, vol. 1, no. 1, pp 18–28.

[48]. Tran DA, Hua KA, Do T (2003) “ZIGZAG: an efficient peer-to-peer scheme for media streaming”. INFOCOM 2003, Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE, Piscataway, USA, Vol. 2, pp 1283–1292