Hindawi Publishing Corporation
Advances in Multimedia
Volume 2012, Article ID 109619, 18 pages
doi:10.1155/2012/109619

*Research Article*

# Enhancing Scalability in On-Demand Video Streaming Services for P2P Systems

**R. Arockia Xavier Annie,[1] P. Yogesh,[2] and A. Kannan[2]**

[1] *Department of Computer Science and Engineering, College of Engineering, Anna University, Chennai 600025, India*
[2] *Department of Information Science and Technology, College of Engineering, Anna University, Chennai 600025, India*

Correspondence should be addressed to R. Arockia Xavier Annie, annie@annauniv.edu

Received 27 March 2012; Revised 7 June 2012; Accepted 28 June 2012

Academic Editor: Martin Reisslein



Recently, many video applications like video telephony, video conferencing, Video-on-Demand (VoD), and so forth have produced heterogeneous consumers in the Internet. In such a scenario, media servers play vital role when a large number of concurrent requests are sent by heterogeneous users. Moreover, the server and distributed client systems participating in the Internet communication have to provide suitable resources to heterogeneous users to meet their requirements satisfactorily. The challenges in providing suitable resources are to analyze the user service pattern, bandwidth and buffer availability, nature of applications used, and Quality of Service (QoS) requirements for the heterogeneous users. Therefore, it is necessary to provide suitable techniques to handle these challenges. In this paper, we propose a framework for peer-to-peer- (P2P-) based VoD service in order to provide effective video streaming. It consists of four functional modules, namely, Quality Preserving Multivariate Video Model (QPMVM) for efficient server management, tracker for efficient peer management, heuristic-based content distribution, and light weight incentivized sharing mechanism. The first two of these modules are confined to a single entity of the framework while the other two are distributed across entities. Experimental results show that the proposed framework avoids overloading the server, increases the number of clients served, and does not compromise on QoS, irrespective of the fact that the expected framework is slightly reduced.

## 1. Introduction

Today, Internet faces proliferation of social network groups that use advanced technology to transfer large commercial data such as image, audio, and video. This trend has led to the popular websites such as YouTube, Flickr, and Joost. As a result, the number of user requests for various video contents through the Internet has grown exponentially every year [1].

Even with reduction in cost on storage and connectivity, the Internet still faces problem in providing quality video to all its customers. Video server faces scalability problem to a large extent with millions of users added to the community every year. Therefore, serving heterogeneous clients efficiently is still an unsolved problem at the servers [2].

Video-on-Demand (VoD) is one such application that has large viewership. It is different from video live streaming. In live video streaming systems, nodes request for data around a particular playback time [3], with ultimately no interactive request such as Fast-Forward (FF) or Back-Ward (BW), and hence become more or less like the broadcast service which is trivial. As number of users scale up, it is necessary to increase the buffer size. This helps to provide a shared frame so that the needed frame could be retrieved from among those peers. There are dedicated servers for different live streaming subjects that handle heterogeneous client groups by sending different streams satisfying the user's bit rate pattern [4]. Our work handles live streaming and goes a step further to handle applications such as e-learning and VoD with interactivity.

Emerging advancement in distributed systems such as peer-to-peer (P2P) systems with new challenges is becoming more complex. This is due to the challenges in (i) dealing with large scale systems, (ii) achieving real-time VCR (Video

Cassette Recorder) interactivity more effectively, and (iii) provision of video quality with less resources in distributed settings [5–7]. Solutions towards all the previously mentioned challenges lie in deploying appropriate systems in integration with (Quality of Service) QoS supportive system, resource management algorithms, protocols, and approaches in design. In this work, we provide a new solution by combining winning factors such as optimal multiversion and multilayer adaptive streaming [8] video server for the distributive sharing P2P clients and a proxy kind of tracker to reduce the load at the sever that manages peers. We compared this proposed work with existing P2P Gnutella and client server systems with relevant QoS and resource management techniques. Performance improvements for two important resources namely, quality and time towards the interactivity and scalability factors have been proved to be attractive with our proposed design. Moreover, latency is greatly reduced in our work compared with the existing works.

A typical video streaming system, as described in [1, 2], requires three major components: (1) a media-stream server that stores and retrieves video based on the available user bandwidth, (2) an intended proxy server called tracker which tracks the video content in the P2P systems, and (3) the designated heterogeneous P2P client group.

Currently, media servers use multicast streaming and serve the same video through multiple channels to satisfy heterogeneous end users. Here, the encoding rate differs based on the bit rates, and hence it is adaptive streaming [9]. Encoding is performed because the server needs different bit rates for the same video to be sent across for varied services like mobile phones, PCs, set-top boxes, ipads, and so forth; this creates additional workload at the server. This is overcome in [10] which uses same multicast group to serve variety of users by dropping few frames for lower bit rate users. This work has been later extended by yu et al. [10] to overcome the defect of reduction in bit rate with an average increase in frame rate of about 30% for better video quality.

However, close observation and analysis of yu et al's [10] work made us to think of combining other solutions for VoD provided through a proxy server called as tracker in this proposed work.

In order to improve the video quality even for a lesser advantageous end user (namely, users with lower bandwidth speed/bit rates), we propose a new solution that combines the features of all the above solutions. Hence, we achieved the user-perceived video, without quality deterrent from media server by proposing a new *Quality Preserving Multivariate Video Model (QPMVM)* for heterogeneous peer group. The term "multivariate" in this work means usage of either multilayered video streams or multiversioned video streamsor integrated video streams for optimal solution tosatisfy heterogeneous users.

Multiversion systems encode a video sequence into several independent streams at different rates [8, 11]. This requires dropping of frames of the higher bit rates to satisfy the lower bit rates. Moreover it helps to store multiple versions of the same video file with different bit rate as per the user requirement. It also streams the particular bit rate versioned file of the same video to requester. Though it increases the storage cost, the quality provided is improved, and streaming latency is reduced. During transmission, the server switches among different versions to achieve the desired sending rate. Here, we have used multiple versions for providing better quality by storing minimal number of versions, which is as low as three versions permanently for the uploaded video by a client.

Multilayer systems encode a video sequence into several nonoverlapped, dependent streams [12, 13]. Multilayer systems work in par with the Scalable Video Encoding (SVC) with the difference that here the server provides input to the dropping of layers of frames on the fly to satisfy the requestor's bit rate. This is necessary because the contributions of Fine Grained (FG) layers to the overall video quality are different [14] which are handled by the server using the transcoder module of QPMVM.

Combining the two streaming pattern is termed as hybrid stream. Hybrid streams are generated when the multiversion streams switch to multilayered one for the required bit rate by dropping frame from the current multiversion stream. Multiversion cannot be combined with Scalable Video Coding since it contains multiple base layers and is more suitable to scale in three-dimension space [14].

When the video quality is conserved to certain extent, we move one step further for reducing the delay and latency. This is performed by sharing of peers and henceforth provides VCR interactivity in VoD application that is more desirable with this proposed model. The tasks concentrated within the peer groups include (i) apportioning cached content, (ii) value-based content placement at the peer cache for sharing, (iii) peer energizer (a sharing initiator), and (iv) heuristic-based scheduler at the tracker system for late joiners of the multicast group and interactive requestors. All these multifaceted functionalities have been combined, to scoop up the VoD system as per the user's satisfactory notion on quality of the perceived video [15].

The media server proposed in this work manages the video requests from heterogeneous clients and is also responsible for handling the scalability factors affecting the server.

The tracker present in this framework is situated at the edge router of the peer groups connecting to the backbone network. The tracker acts as message transmitter across the peers and services media streams that the peers have requested to and from the media server. The management of the peers, like the peer request scheduling, managing entry and exit of each peers, Look-Up-Table (LUT) management using Ant-Based optimized routing for the peer groups are handled at the tracker rather than by the server [16, 17]. The combination of the QPMVM at server and the tracker enhances the system as a whole and hence reduces the latency during scalability.

This work provides an optimal solution for video streaming by finding multiple views of the problem in terms of server, tracker, and peer clients. There are many significant contributions that are made in this work. First, the provision of QPMVM at the server reduces the load and improves the video quality. Second, the tracker proposed in this framework increases the streaming ability within the peer groups. Third, the peers are motivated to share the

distributed video contents. Finally, the latency is reduced for normal playback as well as VCR operations.

The remainder of this paper is organized as follows: we discuss the related work in Section 2; our proposed video streaming system architecture is given in Section 3; the performance metrics used in the evaluation of the system are presented in Section 4; we discuss our results obtained and analyse them in Section 5. Finally in Section 6, we summarize our findings and suggest possible directions for future investigations.

## 2. Related Research

The Video-on-Demand (VoD) as portrayed in large number of systems involves the server component, the proxy or content distribution server along with huge number of clients who are end users of the VoD systems [16, 18, 19]. At the server side, the existing literature has extensively discussed numerous solutions to optimize the storage/retrieval of video files [8, 10, 20]. Any server, which maintains video files in its database with a single version pertaining to a particular bit rate, cannot serve multiple heterogeneous clients with ease and without degrading the quality. In order to serve these heterogeneous clients, the server adopts two methods according to the literature available in this area. Each method has its own advantages and disadvantages. For instance, the multiversioning avoids the cost of encoding/decoding during video streaming with the help of independent stored video file. But this produces frequent switches among stored versions that result in low reference locality and hence incur additional I/O cost. Furthermore, storing multiple versions requires a lot of storage when the number of requested video for different content is tremendously large during concurrent user access.

In multilayered system, each multilayer video can be stored as Coarse Grained (CG), Fine Grained (FG), or a combination of both. Here, the base layer is the most basic version of the video stored, and the subsequent layers are built on top of them. Hence, it is not necessary to waste storage for storing maximum number of different versions of the same video. Instead, only three different layers are used in our work. This, however, has indirect impact on time as the transcoding is performed in real time based on the current bit rate pattern of the peers [14]. Moreover, adaptation is implemented by introducing an extra layer. If there is any problem in transmitting the base layer itself, then the end video's quality is reduced drastically. The hierarchical encoding is more suitable for multicast to heterogeneous clients while the switching streams can be used in both unicast and multicast [20]. But when a video is played at client side, the user may wish to fast forward (FF), rewind (RW), and so forth. This requires the video to be played at an increased speed. In short, the client needs an interactive server, which responds immediately as per the client request. This becomes quite difficult, if the server has a single large file, because, to play the video at an increased speed, the frames must be identified and transcoded at run time [10]. It becomes easier when the server maintains a special copy of video, which gives the client a fast version of the same video. In this work, we achieve higher performance using the proposed Quality Preserving Multivariate Video Model (QPMVM).

Today, as the cost of storage has come down, we effectively constructed the server by combining both multiversion as well as multilayered approaches and tested the system for heterogeneous P2P group. P2P client pattern is the most accepted among the many VoD system solutions. P2P media streaming is becoming popular rapidly for the reason that the streaming service of client/server model takes up too much server resource and could not meet the increasing demand of scalability [7]. CoopNet, PALS, PROP, Toast, and Zebroid provide on-demand streaming using P2P networks. Each of these systems seeks to support an infrastructure-based system with P2P networks and thus achieves scalability to some extent [20, 21]. However, all these existing solutions are not sufficient to provide effective services in the current scenario. Therefore, it is necessary to provide new store and retrieval techniques to meet the current demands. In this work, we achieve scalability through multiversions. Moreover, this work has been implemented in P2P technology and shows the advantages of serving on a large scale that has already been introduced to the media streaming systems. VCR (Video Cassette Recorder) functionality is one of the most used features in real-world streaming applications [22] which operates for FF, BW, seek, pause, and so forth. Also, the problem of scalability to a large user population in media streaming systems is only mitigated to a certain degree in the past but not solved.

Optimal methods for video file storage are becoming increasingly important owing to the rising numbers of video server populations. Adaptations that optimize playable frame rate by storing minimal number of configurations require intensive computation through loading of processor [23, 24]. Storing all possible configurations for a single video file for all the bit rates a client can have during internet connectivity requires tremendous amount of storage. Only when the memory used is sparse and the processor load is reduced [25], video servers would find it appealing. To meet these apparently conflicting demands, this work proposes a system for optimizing video server storage.

## 3. Video Streaming System Architecture

The video streaming system proposed in the paper possesses the media server which services the client requests for the videos stored in its database. A proxy, called as tracker in this work, lies between the media server and the P2P clients. The proxy is implemented in one of the peer nodes or at the edge router of the network connecting the media server and the P2P client systems. Here, it is implemented at the edge router of the system so that it conveniently tracks the client requests that passes through it, as well as the network condition at the peer end. Figures 1(a) and 1(b) show the high-level setup of the video streaming system architecture proposed in this paper which comprises of all the three main components of

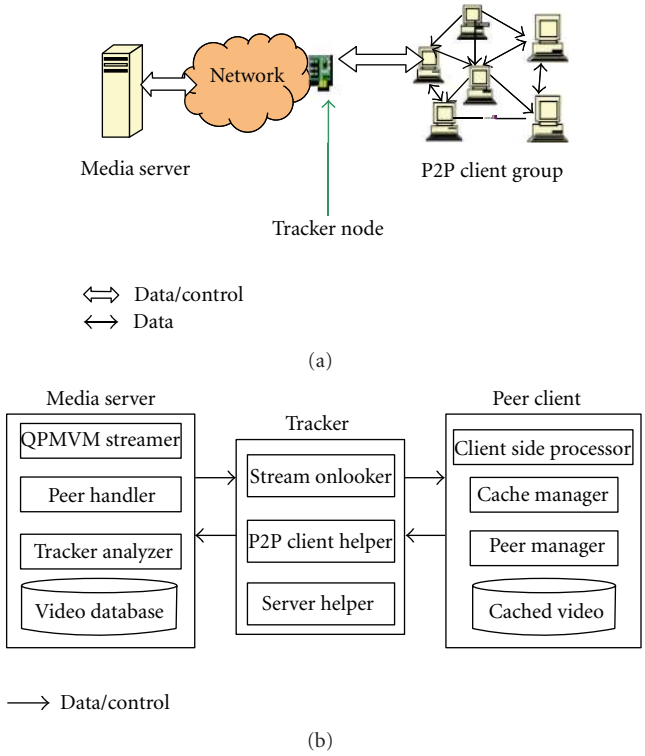


Figure 1: (a) System architecture; (b) High level system.

the VoD system, namely, the media server, tracker, and the peer clients.

*3.1. Media Server.* A video server, which is connected to the Internet, may have diverse client populations and hence is very difficult for the server to respond to multiple clients with varied user profiles. The situation of multiple video requests by heterogeneous clients is handled in Napster server and few other VoD servers by having multiple versioned streaming systems [26–28]. But the same can be achieved using multilayered approach as well. In this approach, though the same stream is sent to both the users, it requires dropping of some frames. In such a scenario, few very important frames such as "I" and "P" frames might also be dropped depending on the overflow of the buffer at client end and the transmission speed at the server/client network. In such a case, the quality at the user end would not be commendable.

By considering the advantages of both the systems proposed by Yu et al. [10] and Cheng-Hsin and Mohamed [8], we propose a hybrid approach in this paper for providing effective VoD using multiversioning with multilayering. In addition, we propose a new proxy called tracker deployed at the edge routers of each client networks in order to reduce the delay in searching and forwarding from the server. From the experiment conducted with this new model, it has been observed that this multiversioned system provides better quality than the multilayered system [12, 29]. On one hand, when a channel experiences network congestion, a multiversioned system which has a precomputed version file, sends the correct frame based on the bandwidth availability at that time. On the other hand, during such congestion, the only possible option for the multilayered system is to drop frame(s) irrespective of its priority. Therefore, in this research work, the features of both multiversioning and multilayering are integrated, and a tracker with local user profile database and decision-making capability has been proposed.

*3.1.1. Quality Preserving Multivariate Video Model (QPMVM).* As mentioned earlier both multiversioning and multilayering systems have both advantages as well as disadvantages. The choice depends on various factors that arise from the exact user profile details known to the server at the start of a streaming request. Moreover, the packet receiving details from the network side which are received at the server via probing the network to know about the congestion and slow delivery helps to analyze the reasons for congestion and delay. This challenge is addressed in this work by introducing the QPMVM at the server as one of the modules in order to manage the server by providing consistent throughput without overloading the server even during peak number of requests. Figure 2 shows the architecture of the proposed QPMVM model at the server. We do not make any assumption about the format of layered video. Any scalable or layered video coding scheme can be incorporated into our framework. This can work on lower-end codec such as MPEG-2, MPEG-4, MPEG-21 as well as higher end codec such as MPEG-4 Part 10/H.264 AVC (Advanced Video Coding). As H.264 SVC (Scalable Video Coding) is delay intense with respect to computational complexity than the other codec forms [30], this work is better off with H.264 AVC with single-base layer. Hence, in this work, we have considered all MPEG video formats with the intent that most of the videos used in the Internet fall into this category. QPMVM has the advantage of identifying through the seed estimator, a way of holding lesser number of multiple versions for a video which is encoded by the transcoder. From the stored multiversions, the multilayering is done on the fly through the Decider module.

Generally, whenever the server detects any network change, it drops a layer of frames, in order to achieve the desired bit rate for the given video. Dropping "B" frames allows protection of more important "I" and "P" frames for the same amount of bandwidth [29, 31]. The data size for the three frames is known to be I > P > B. Also, dropping "B" frames sparsely does not affect much of the user's perception of the video [32–34]. But dropping multiple layers of frames, without keeping in mind the frame priorities with respect to its Group of Picture's (GOP) can affect the received video quality at the client to a great extent. In such a scenario, due to bit error and packets loss it is not possible for the receiver to recover all the frames. For example, the first B frame can be recovered only if the first I frame and the first P frame are successfully recovered. The loss of P frame will cause subsequent P and B frames to be unrecoverable, affecting the quality of the video. To increase the expected number of frames successfully reconstructed, the server can adapt to the network condition by changing the amount of redundant

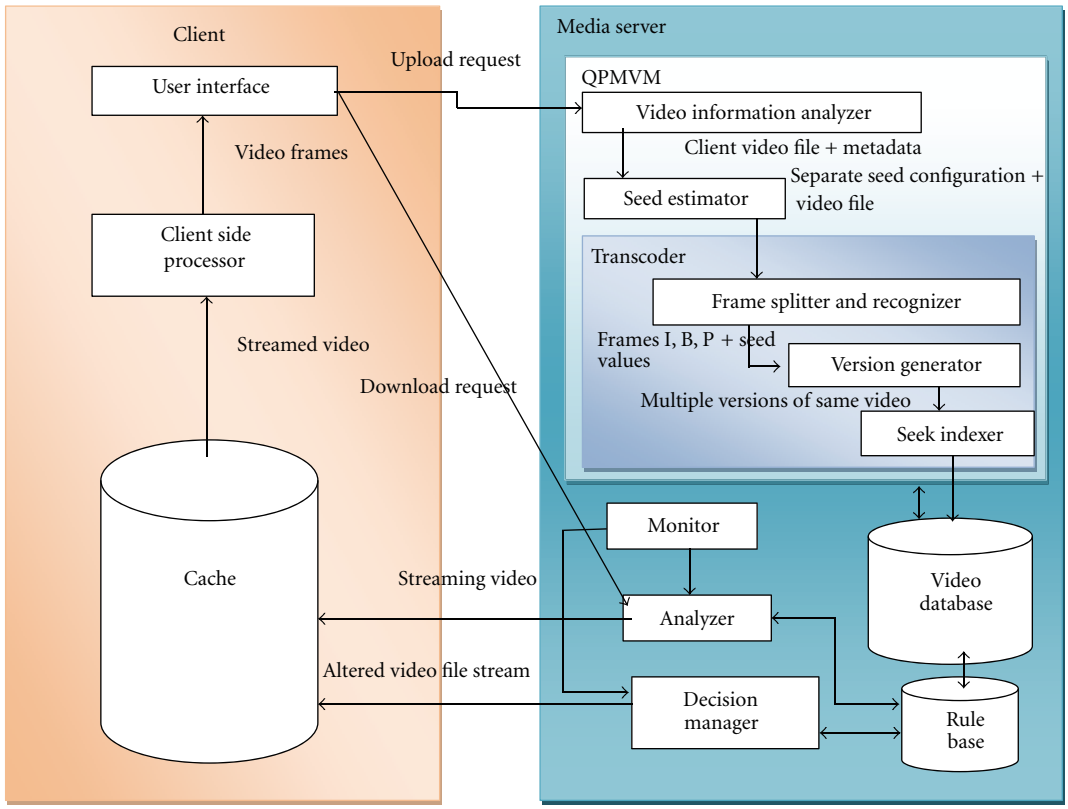


Figure 2: Media Server.

information sent to protect the video against different packet loss rates [4, 7, 9]. The system must alter the bit pattern in the frames' Group of Pictures (GOP) to improve the efficiency.

This is because of the fact that when a video is being played at an increased speed, "$n$" number of frames may be dropped per GOP (of "$m$" frames). When receiving the video file, the receiver will expect "$m$" frames when actually only "$m - n$" frames are received. This reduction in frames must be set in the GOP's bit pattern so that the receiver expects only "$m - n$" frames. This is done by supplying the end user with Forward Error Correction (FEC) code as done in Jiangchuan [13] within the MPEG frame header. Block diagram of QPMVM subsystem and the important activities of the subsystem are shown in Figure 4.

The QPMVM system in media server can be logically divided into two components, namely, upload and download video streaming. The uploading component uses Transmission Control Protocol (TCP), and the video streaming is performed using Real Time Protocol (RTP). When the system is executing in an upload scenario, it takes a video as input from the client side. As video sent from the server is implemented using RTP, it gives added advantage in determining video streaming metrics that are more appropriate to the real world scenario that combine Real Time Streaming Protocol (RTSP) as well as Real Time Control Protocols (RTCP) in a connection-oriented network. Next, based on the client side's profile which includes details such as IP address, bandwidth, and user preference category (i.e., high priority user, medium priority user, low and very low priority), the server decides its video streaming method to be either the layered approach or the versioned approach for a video request to a particular client based on the user profile and details from the network. Once these details are available at the server side, the video is analysed to decide on the seed configuration of the video that is to be streamed. Once the seed configurations are identified, based on the information gathered from the video file the versions are generated. The process of generating versions involves identifying the frames based on the information gathered. The versions are then stored in the database.

When the QPMVM system is executing in a video streaming scenario, multiple components work at the same time. First of all, the client request for a particular file is received at the server which triggers a check for the availability of the file in the database. Further, a check for a compatible version is done. Once both the checks get completed, the users' connection profiles are now taken into account, and if an exact version has not been found amongst the existing versions, then transcoding is done at run time and the video is streamed across and played on the client side.

Throughout the entire process of streaming the video file, the network conditions are monitored, and when a change in the network condition is detected, communication with the client is activated to check how many frames of the video have already been played at the client. This frame information coupled with the change in the network bandwidth information is used to identify which part of the

video must be played and in which version. Basically our pattern of evolving around user profile data does not mean that very often communication through message transfer is made to acquire knowledge of the current situation, rather only when the network causes reduction in data flow do we make the server change its stand from multiversion to multilayering. This is beneficial immensely as the time for deciding is reduced.

In other works, the communication with the client to know about the details of frames played (current situation) is not done periodically rather only when the situation demands. This obviously helps during client VCR request pattern where, for a medium or a lower prioritized client the layered versions are sent for that specific VCR requests. The following subsections explain the different modules in the QPMVM system with their implementation details.

*QPMVM Algorithm:*

**Request Type I**

**Input:** Request to upload a video into the server database from a client

**Output:** Server stores the uploaded video into the database

**Process:**

(1) Video information extracted from the uploaded video
(2) Seed configurations identifier to create versions
(3) Frames split and stored

**Request Type II**

**Input:** Request a video to the server by a client

**Output:** Client receives the video

**Process:**

(1) Based on client network priority

(2a) If same video version is present in database it is retrieved and sent to client

Else

(2b) Multilayered video is sent to the client

**Request Type III**

**Input:** Video VCR request to the server by a client

**Output:** client receives the video part

**Process:**

(1) Based on client network priority

(2) Appropriate video part with required version is layered to client.

The QPMVM is a major module in the server side which extracts the video metadata, splits frames, creates versions, and successfully stores it as indexed version in the video database. It uses the aspects like bit rate control, buffer control, and frame positioning control methods from the javax.media classes of Java Media Framework (JMF). It encompasses three modules:

(i) Video Information Analyser,
(ii) Seed Estimator,
(iii) Transcoder.

*(i) Video Information Analyser.* It receives the video file from the client and analyses the metadata to separate useful information like Group of Pictures (GOPs) and so forth. This is done by extracting the MPEG header from every video file. The upload operation from the client sends a video file. The video information analyzer splits the metadata apart from the video part of the received video file. The metadata is parsed, and the information about the client is gathered such as its IP address and its user profile as being high or medium or low. The video information is also gathered from the metadata like frame rate, GOP, video duration, bit rate, and so forth, both these information is then stored for future processing. The output from the video information analyzer is sent to the seed estimator and peer handler modules.

**Input:** Client uploaded video metadata

**Output:** Client information and video information

**Process:**

(1) By extracting data on the uploaded client's connectivity, the client is categorized as:

If the connecting bandwidth is

(a) up to 52 Kbps then, very low priority client
(b) between 52 Kbps and 112 Kbps then, low priority client
(c) between 112 Kbps and 256 Kbps then, medium priority client
(d) between 256 Kbps and 512 Kbps then, high priority client

(2) Video metadata is extracted from the video file.

*(ii) Seed Estimator.* As the name suggests the seed estimator receives the metadata and decides upon the seed values that we intend to make the uploaded video into several versions. The seed values here mean the relative number of versions that is required for a particular video file to be stored in the database. It is the seed values generated that decide upon the version numbers. This is intended for client nodes with Internet connections ranging from 16 Kbps through a telephone dial up networks to 512 Mbps through DSL networks.

In order to arrive at the seed values, we calculate them by analyzing the bit rate of the video; subsequently we reduce the given bit rate by just half the bit rates and figure seed versions that equal these halved rates. We stop factoring the bit rate by two until the bit rate reached is greater than or equal to 16 kbps because it is known that user's connection setup speed cannot be slower than this. The reason we go in for halving the bit rates without doubling it is we signify the uploaded video version as it is with its provided quality and

go in for supporting the lower end clients with almost the same quality by storing versions with lesser bit rates. Most importantly, these subversions (means lower bit rate versions of the given video file) are each arrived through the generated seeds, and hence each video version is stored expressing the seed values. This paves in for storing only very few versions for the overall vast difference in the number of client bit rate pattern, thereby reducing the storage space at the server enormously.

**Input:** Video bit rate of the uploaded video file by a client

**Output:** Number of seeds for the input video file

**Process:**

(1) Uvbr = video bit rate of the uploaded video file by a client

(2) seed_count = 0

(3) while (Uvbr ! = 16 Kbps)

Uvbr = Uvbr/2

If Uvbr generated is within the priority limit of the client group, then

seed_count = seed_count + 1.

Else

No change in seed_count.

*(iii) Transcoder.* The transcoder is another major submodule within the processor. This module uses the input from the other two submodules within the processor and performs transcoding operation. This ensures different layers exist upon these base versions. Transcoder includes (a) frame splitting and recognizing I, P, and B frames, (b) version generation, and (c) seeking the frames in the video.

*(a) Frame Splitting and Recognizing I, P, and B Frames.* Frame splitting module identifies the Intra (I), Bidirectional interpolated (B), and Prediction (P) frames and splits them. GOP gives us information that says how many frames are present between consecutive I frames. Each frame of the video has a separate header which is called the picture header. This is essential because it contains information which tells us about I, P, and B frames. The picture header has the frame information in byte 5 of the MPEG header. The bits set at 3rd, 4th, and 5th position are checked, and the following are inferred: "001" is I frame; "010" is a P frame; "011" is a B frame. The seed values provided by the Seed Estimator module and the video received from client are used here as references to generate multiple versions of the same video to accommodate heterogeneous client bit rate pattern. The realization of I, P, and B frames from all the GOPs is done. Here, information from the video file's hint track called the *video track,* is used to include streaming information. A video file usually contains multiple *tracks.* Individual tracks can have metadata, such as the aspect ratio of a video track, title of the video, episode numbers, and variable bit rate/constant bit rate [35, 36]. The track information set at the MPEG header of the uploaded video file provides more information which is hinted in the video file per data unit of the stream. This will help in extracting the GOP during sending the video for requests. The information of the current frame being processed from the uploaded video file lends it to being I, P. or B frame.

**Input:** Video frames

**Output:** Recognize and Split Frames as I or P or B

**Process:**

(1) For each instance of the Track-information or hint tracks from the metadata of the uploaded video file

(1a) Store images in Image-control.

(1b) Extract data units from the hint tracks

(2) while (!Reached end of the Frameset)

{ I = track_no $*$ GOP

P = track_no + $(k + 1) * j$

B = all other frames }

Where-"track_no" is the nth I frame encountered at that time, "$k$" is the number of B frames between the consecutive I and P frames "$j$" is P frame counter.

*(b) Version Generation.* The user video needs to be morphed to the lower versions. This is done with the help of the seed values that have been identified. Based on the property that dropping of B frames does not affect the quality of the video that much, the system proceeds to drop B frames at first to accommodate to the lower bit rate versions. We use the information obtained from frame splitting and recognizing for the seed value from the seed estimator to generate versions for the users with lower bit rate pattern. In this work when creating multiversions, the frame rate is not affected much, because the bit rate has no impact on the frame rate of a video. It is with respect to a codec [23, 35]. Also, the aim is to fit the multiversion file into any possible client bit rate without losing quality. Different versions of same video file are generated using the following algorithm.

**Input:** Client uploaded video file, seed_count from seed estimator

**Output:** Video versions generated

**Process:**

(1) Read video as bit stream

(2) Check for GOP header and picture header, if (match found)

(a) Read nth bit and obtain whether I, B or P frame

(b) For each seed obtained from seed estimator, drop every 5th B frames subjected to the required bit rate (information from Monitor).

If (no more B frames to drop) drop no frames.

Generate versions.

Every 5th B frame is dropped, since it is not very difficult to generate one, as it might lie between two B frames and/or

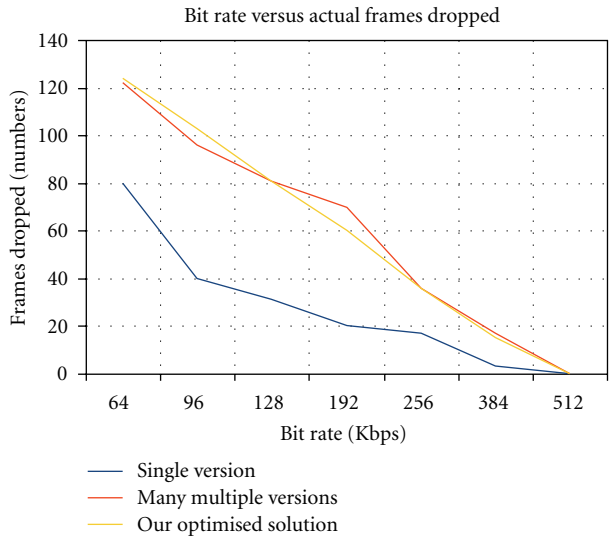


Figure 3: Bit rate (kbps) versus actual no. of packets dropped.

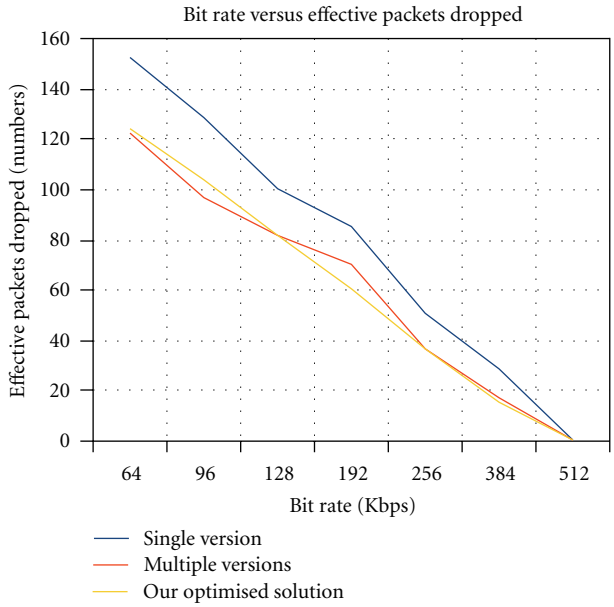


Figure 4: Bit rate (kbps) versus actual no. of packets dropped.

an I and a P frame as well. This is better than dropping consecutive B frames [34]. Any GOP under consideration would have the probability that based on this information it would speed up the encoding/decoding process to a greater extent. This is better than to scan the frames of importance before dropping, which is time consuming or not to scan at all which is unfavourable when quality is perceived [37]. This is supported by our simulation result as well from Figures 3 and 4.

*(c) Seeking the Frames in the Video.* When a client prefers to play a video file, he selects it and starts to view the video file. Sometimes, after the client has started to view the file, and when the video is still being played, the same client's network condition may change for the worse. Congestion may throw up, and video frames may get clogged at the bottleneck link which may not be able to transfer the high bit rate frames that the client is currently viewing. But the link might allow few frames of lower bit rates to pass through it. Our system is designed to adapt to such changes and provide the next best available video version for that condition. In order to achieve this, the seek indexer traverses the video file after every nth frame to avoid retrieving the entire video file in the new version to be played for the client. The inputs to this module are video frames played by the client, and the seed value corresponding to the bit rate change needed to be placed on the incoming video due to the change in the network condition.

**Input:** link strength, video version

**Output:** frame to be played is identified in stored video version

**Process:**

(1) Extract the video version file to be played for bit rate given by "i" at the client.

(2) while (not end of file )

```
{   Check frame rate;
     if (frame_no mod GOP = = 0)
        i = frame_no / GOP;
     if (frame_no is a multiple of GOP)
        Timestamp = frame_no_i / n;
}
```

Where "i" is the indication of the frame we try to determine as the one to be played at the client.

*3.1.2. Monitor and Analyser.* The input to the monitor is from the client side. Output is the identified network conditions from the RTSP protocol. The functionality of the analyzer is simple. It merely searches for the client compatible version, and if it is not found, it sends the nearest compatible lower version. This is understood from the following

The output from monitor module gives the network configuration details of the client's download request, and this input query is sent to the database, to search for the client compatible version.

**Input:** link strength, video viewed by the client

**Output:** video to be played is identified in stored video version as per the link strength

**Process:**

(1) If the queried file is found, search for version

(a) if version not found, extract immediate lower version file and send

(b) Else, send the version matched file.

*3.1.3. Decision Manager .* In a video streaming application, after a client has partly downloaded a file, if the video is still being played, the same client's network configuration may change. Our system is designed to adapt to such a change and provide the next best available video version. Here, it becomes important that we do not retransmit the frames that have already been played on the client side. The decision manger predicts and decides which frames can be skipped and which frames should be sent.

The process dealt at the decision manager can be explained with the following example: If a client-A is connected to the system for some "$t_x$" seconds, during which "Y" number of frames is received at that client. If the network configurations change at the some "$t_{X+1}$th" second, and if we analyze and send the new version video, it would be meaningless to send the first "Y" frames. The decision manager thus transmits the video from the "Y + 1th" frame after analyzing the current GOP and its associated frame streamed recently. For instance, if the "Y" frame is an "I" frame, thus decoding and reencoding are needed to search for the new version starting from the "Y + 1" frame. Therefore, the current GOP helps out in the delivery of required stream.

Here, the usage of RTP protocol helps in achieving this pattern of skipping frames and sending the current frame from the stored multiversion video file. This is one of the novel techniques used in our work that reduces both time to select the frame from stored video version and send or to generate them to the current bit rate level of the connected client through the already sending stream on the fly [8], by dropping packets which is multilayering done in adaptive streaming method. Client side communication is also done through this module.

The selection of stored video version or dropping of frames to match the limited bandwidth is decided here.

The input given is video file that client has requested along with the monitor's output as to which frame has been viewed recently. The output to this module generates a new compilation of video with dropped frames

**Input:** Identified video version from Server

**Output:** Frame to be played at the client

**Process:**

(1) If current bit rate of client matches stored video version then check frame number, seek to that frame and send from that frame onwards.

(2) Else, drop layers of B frames from the current video stream, seek to the requested frame number and send from that frame onwards.

*3.2. Tracker.* The cooperation provided by the peers by uploading to the fellow peers with incentivized pattern as seen in [18, 19, 38] is added to motivate the peers as jubilant uploaders. This further achieves better focus on servicing requesters rather than spending time in identifying providers at the tracker. In our system, we consider all peers are cheerful in providing as they are serviced with greater download speed for each completed (100% data received by the peer requester) or partially completed (40% to 90% data received by the peer requester) uploads. The combination of these destined methods at the tracker provides novelty to this approach with which we deem to achieve far more than that was intended with great impact on client satisfaction. The basic processes at the tracker for all the combined effort to be put into the router or a separate dedicated peer node for this tracking should include the following modules: (1) Stream Onlooker, (2) P2P Client helper, and (3) Server-side Helper.

*3.2.1. The Stream Onlooker.* The Stream Onlooker (SO) is quite similar in a way to the analyzer module of the QPMVM. This keeps a watch on the incoming requests from the client to the media server which notes down the request information in the tracker table (LUT). Also, it saves the video segment information stored by the media server during push stream which is used for managing further requests among the peer groups. Among all the other things, it keeps watch on the stream request generated within the peer group, by looking for further requests to be made by the same requester after assigning another peer to serve for the request currently made by the requesting peer.

For example if peer "A" has requested a video segment "i," then the tracker could receive only two more request, for the same video segment from the same peer "A." By this we guarantee that the request has either been serviced by another providing peer for the better or the request has not been served by the provider peer to whom the request was directed to, for the worse. In which case, the tracker manages to route it to another peer and the media server simultaneously. By this way the tracker fixes the servicing time for the requesting peer and looks on for another reliable provider with options open within the local peers of high profile range groups.

This part of the work at the tracker to look for capable service providers in the peer group satisfies the requesting peer to get serviced before the exhaustion of its buffered video content. Moreover the agreed peer provider who would not serve the requesting peer client would get less benefit in terms of speed and video quality as the agreement has been futile with no services at the client end. After agreement, the nonproviding peer node's information could be captured with the same request generated by the requesting client who has timed out its earlier request. This information on the nonprovider becomes fruitful at the same time to look for any further requests from the non-provider as such to look in for network-related problems like link congestion, attack-driven malicious nodes and so on, and if there are no such problems, the information would be used for negative incentives.

**Input:** Streaming requests to the server/tracker from the peer clients

**Output:** Manage client uploading/downloading pattern send information to tracker

**Process:**

(1) Maintain database

(1a) Requested video file, user id of the requester, number of requests by same user for the same content and/or different content, servicer details and so forth,

(1b) For uploading within peer groups and the rate of upload.

(1c) For downloading within peer groups or from server and the rate of upload.

(1d) Place the most uploading peer at the top and the remaining in consecutive order of their sharing ability.

(2) Update tracker with the managed database for every request serviced.

(3) Provide entity on the PM and CM to PCH.

*3.2.2. P2P Client Helper.* The P2P Client Helper (PCH) manages and provides information for Ant-based routing at the tracker. The Stream Onlooker (SO) updates and sends the information on each peer based on the services required as well as services provided among peers to PCH. The PCH holds the latest information on the clips cached at each peer in all the peer nodes as a table.

**Input:** From Stream Onlooker

**Output:** Analyzed video segments and policies

**Process:**

(1) Places the pushed video segments from the media server into the peers, by queue maintained by the SO.

(1a) The top of the queue has the maximal provider and the rear the least provider in the peer group-This data subjectively used for pushing data as well as incentive at the peer.

(1b) Identifies and updates route for Ant based optimization. (Any peer node is not selected for an optimal route but with the information from the maximal queued.)

This achieves high delivery to the client side by combining these several solutions put together.

*3.2.3. Server-Side Helper.* The Server-side Helper (SH) mainly concerns with scheduling the peer group requests mostly at the tracker and very few at the Media server. The Server-side Helper maintains hash table to provide the requesting peer map with video segment needed from the various peers stored across peer group. The PCH and SO input the SH with the needed information for managing the admission control dynamically altering the choice when the provider or the receiver is no more worthy of doing any of it due to network-related problems. The topology of the peer group connected and maintained as Earliest Reachable Merge Tree (ERMT) with Static Full Stream Scheduling (SFSS) [11, 16, 39] gains more momentum in multicast streaming for the peers as scheduled by the SH. The SH provides another most prominent work to server that helps it to place the pushed video content at the peers based on the ability of each peer of its bandwidth speed and buffering size which is to serve others. This is to identify the best supplier (peer) for the current request. By best suppliers we mean those peer nodes that can stream the requested video as fast as possible which is achieved by tracking the reduced load at those peers to serve the requester.

**Input:** Split video segments

**Output:** Place video segments among the clients

**Process:**

$m$: total number of peers in the system

$N$: total number of segments a video is split.

$A$: average bandwidth of all $m$ peers

bi: bandwidth of ith peer "Pi".

ph: high bandwidth peers

pl: low bandwidth peers

Npl: Total number of segments to be allocated for low bandwidth peers.

Nph: Total number of segments to be allocated for high bandwidth peers.

Categorize the $m$ peers into

ph where, bandwidth $> A$ for all nodes in this set and

pl where, bandwidth $< A$ for all nodes in this set

for i = P1 to Pm

Nph = $(N * (\text{bi}) \text{ PeerHigh}) / (A * m)$

Npl = $N -$ Nph

Ni = $(N * \text{bi}) / (A * m)$

Ni video segments are distributed to Pi.

The video segments that have been obtained by splitting the video file are distributed among the various peers in the system based upon their utility bandwidth. More precisely, the peers having more bandwidth receive more segments and vice versa. It is also possible for the end user to view a specified segment in the video. This improves user interactivity of the system by providing for random-seek option.

*3.3. Peer Node.* Distributed P2P network is more advantageous in the setup of VOD system [17, 40]. Each peer node is equipped to handle a factor of the overall load from the server by serving with its buffered content. Peers are henceforth clients with mini version of the server. The requests arise from the clientpart of the peer node, and the streaming service comes from the server part of the peer. Here, we call it as service provider. Not only does the service provider is quite complicated but then it needs to have the current information on the video segments held at each peer and also needs to combine with existing multicast streams to service in a better way. The user interface at each peer contains the Audio Video receiver (AV receiver) which is used for Real Time Protocol (RTP) video reception. Processing at each Peer node comprises of the (i) Client Side Processor (CSP) (ii) Cache Manager, and (iii) Peer Manager which we detail in the following sections.

*3.3.1. Client Side Processor.* The Client Side Processor (CSP) acts as the heart of every peer node in the network. Because, it keeps track as well as informs the most important matters happening within the peer to the tracker as well as neighbor peers. CSP attends to the various tasks that take place at the peers.

The most important of all and several other individual peer subtasks like decoding the video and connecting with Cache Manager and Peer Manager. The various factors leading to proper uploading and henceforth the incentives maintenance very much depends on CSP's informed data. Here, the sender does not magnify the sending details but the receiver notifies it to the tracker. Which means in the normal sense as the sender might not fulfill or rather provide to the custom peer node but inform to the tracker that it had completed its service; this magnification by the sender is clarified by the receiver's CSP information. The components of the CSP meticulously work with the Media server, tracker and other peers to provide them with optimized content delivery. The Cache Manager (CM) and the Peer Manager (PM) also inputs content stored to the CSP.

**Input: Various tasks at the peers**

//∗ multicast stream upload, multicast stream download, buffer scanner, network bandwidth connectivity, video segments buffered, services provided and services received, services disrupted, subtasks like decoding the video and connecting with Cache Manager for knowing high priority task and Peer Manager and so forth, ∗//

**Output:** Information supplied to tracker and neighbor peers

**Process:**

(1) Inform on each task performed at a peer to the tracker.

(2) The stream shift that takes place for the different version of the same video file, is also provided by the CSP during uploading to fellow peers.

(3) CSP maps the tracked frame number when rewind, fast forward and pause buttons are clicked.

(i) It uses the aspects like bit rate control, buffer control and frame positioning method.

(ii) It induces request appropriate to the Cache Manger's data on the cached content within the playback time.

*3.3.2. Cache Manager.* Video content buffered at the peer is managed by Cache Manager (CM). The importance of cache manager towards bringing out efficient working of peer groups sharing is explained in this subsection. There are two cache, used in each peer. Ones completely dedicated for the server to push its video content called the *serve-cache* is and the other is used by the peer for its client application called the *playback cache/buffer*. In our work, we presume there is enough cache to hold at the client so there is no need for cache replacement policy.

The serve-cache would also slowly accustom to fill in the user's/client's interest. The user's interest is expected to lie within any two of all the videos at hand. The Cache Manager acts as a monitor of the playback system wherein the playback time of the current clip is used to fill in the playback cache. If the cache is almost empty and does not get filled up within the stipulated playback time,

**Input:** Playback time indicator from the player.

**Output:** CM raises high priority issue to the CSP.

**Process:**

(1) CM does not wait until the last bit of the cached content is viewed rather as in several of the Media players' cache warning takes place at half of 1/4th of the cached content [28] which is fixed as the threshold here too

(2) Sets high priority bit.

(3) Halt all other services that it was carrying out and perform for getting the next needed video segment.

(4) For normal play back, the cached content in the playback buffer is managed to hold the viewer's interest through (1)–(3) steps.

(5) For VCR requests, if the server-push content is different from the interest of the user it would not benefit from the cached content.

(6) % of cache hit is required to be greater than the cached content at each client. Hence, the non-interest cached content would alter its position from *serve-cache* to another of the interest filled *serve-cache* peer node.

For any VCR request like seek Fast-forward generated within the client, the playback buffer keeps playing the latest video segment that it is currently viewing for 20 msecs and immediately switches over to the kth video segment held in the playback cache. Here, $k$ is the random number generated that lies within the currently viewing clip number to half of the maximum segment number. The maximum segment number lies as the last clip in the limit of the existing playback buffer content [40].

If the VCR request is for seek Fast-backward, the same as Fast-forward with reverse action carries on here. But, if the currently viewing clip is the first clip in the playback cache, this particular seek request grabs the clip segment previously to currently viewing clip minus kth (any integer value for $k$ that matches the *serve-cache* content) clip from the *serve-cache* if present (which is highly possible); else it grabs the clip within next hop of its neighbors processed by CSP within 2 secs (maximum).

For any other VCR actions like pause, continue lies within the previously explained two types. For request of any new video, while the currently viewing video is different, CM searches for video in *serve-cache* instantly without looking in the *playback cache*. It then accepts clips from a different multicast stream through CSP and retains the currently viewed clip in the *serve-cache*.

*3.3.3. Peer Manager.* Communication between other peer nodes to receive and to send video streams happens through this Peer Manager (PM).

The speed at which the receiver receives the video streams gets increased by few kbps based on the incentivized value prescribed by the tracker; the senders have minimal upload bandwidth in which the receiver combines channels as it receives from many nodes at the same time quantifying the incremental speed for its service already provided.

**Input:** Tracker information and bandwidth correlation map.

**Output:** communication and incentives by interacting with the groups intimated by the tracker through CSP in order to supply at incentive sufficed speed.

**Process:**

(1) For each video shared among a group, it maintains a leader (higher bandwidth within group);

The PM accurately maps with all other servicing peer nodes at the same time and balances upload among this specific group for the specified client.

*Incentive Algorithm*

*Peer Up-Loaders*

(1) Tracker stores the up-loader's peer information for incentive management.

(2) Critical case-exhaustion of buffer content,

(i) Uploading peers can agree on uploading for critical case.

(ii) Once agreed,

(a) No more high priority bit is set by the requestor

(b) Else, look for Congestion, Malicious nodes. If found, then do not mark the agreed up-loader for negative incentive (reduce download speed by 10 kbps per negative score per agreed peer)

(c) Else, mark the agreed up-loader for negative incentive

(d) Else, uploaded successfully (100%) or partially (40–90% may happen due to external failures), provide positive incentive counter set (increase download speed by 25 kbps per positive score by combining multiple agreed up-loaders)

*Peer Down-Loaders*

(1) Incentivized peers are tracked by the tracker. PM present at each peer verifies the incentive received.

(2) Achieving incentives (at tracker, Peer):

(i) If Incentive queue managed at the tracker has counter set and if the counter is not more than can be serviced then provide full incentive.

(ii) Else, divide the counter by half and check whether half the incentive can be serviced by individual Up-loader or by combining multiple up-loaders,

(iii) Repeat the above step until incentive serviceable.

(iv) If Negative incentive, update incentive queue per peer basis at tracker and that peer.

*Combining Multiple Up-Loaders*

This is done as one of the group member say "Pi"' who has maximal bandwidth say at "$q$" kbps, but currently provides at the rate of "$q$-$x$" kbps (since it in reality, complete bandwidth is underutilized) this "$x$" kbps would be compensated by another peer in the group by sending at its current bit rate "plus" the left out bit rate that is, if supposedly it sends at "$r$" kbps and its maximal is below "$r + x$" kbps it compensates the complete streaming rate. If not, the "$x$" kbps would be shared among the existing group to provide incremental support bandwidth which is obtained by adding current sending rate plus 2 kbps by all until receiver is fulfilled with the added bit rate. This requires enormous client communication management to support the incentive set. Hence, with PM the feasibility to provide to peers with ultimate caution and care has been obtained.

## 4. Performance Study (METRICS) and Experimental Setup

The experimental setup for the simulation run is shown in Table 1. The simulation is done using Java code for the entire system. Here, the simulated system is compared with Bittorrent-based Gnutella system and the normal client-server system and we call the Gnutella system, as the traditional P2P system as they form the base for any P2P network system for file sharing. In traditional P2P system, the segments are equally distributed among the peers without the tracker scheme. The components of a social network are simulated as our peer network groups.

Firstly, we evaluate the main subsystem in VoD setup, which is our QPMVM that ensures video quality does not gets deteriorated as the user bandwidth reduces below the video bit rate. We evaluate this subsystem present at the server by calculating the loss of packets and compare the loss of effective frames in packets to the existing multiversion and multilayer systems. We consider this as the parameter for QoS. Normally, when frame drop occurs, the receiver's capacity is lesser than the senders. A simulation setup that performs for the frame drops in three different systems are handled here. It shows, I and P frames lost with single version and many multiple version system is relatively high than our QPMVM system. Though it stores very few multiple version it rarely drops effective frames. In QPMVM we look for B frames that are to be dropped first, and lastly we drop the P frames and finally I frames during the packet transmission. This of course is not handled in both multilayered as well as multiversion systems where the frames in a packet are

Table 1: Experimental values.

| Simulation setup | |
|---|---|
| Total number of peers | 100 |
| Total number of videos | 5 |
| Min-Segment size | 1 sec |
| Max-Segment size | 25 secs |
| Serve-cache size of each peer | 200 secs or 180 segments |
| Play buffer size | 30 segments |
| Upload bandwidth | 32 kbps–512 kbps |
| Download bandwidth | 32 kbps–512 kbps |
| Video bit rate | 64–512 kbps |
| Length of video | 180 secs |
| Tracker bandwidth | 32 kbps–512 kbps |
| Simulation time | 3600 secs |
| Reservation buffer size | 4 segments |
| Incentive increment per count | 25 kbps |

Table 2: Video Samples.

| S. no | Video name | Time length (min:sec) | Data rate (kbps) |
|---|---|---|---|
| 1 | v_1 (Core) | 0:50 | 4086 |
| 2 | v_2 (Bodyguard) | 3:30 | 4971 |
| 3 | v_3 (3idiots) | 4:40 | 4209 |
| 4 | v_4 (Magadheera) | 5:40 | 5204 |
| 5 | v_5 (Inception) | 6:40 | 3466 |

dropped without its relevance consideration as studied in [12].

Secondly, we evaluate the proposed work by considering the entire system as a whole with the server and the tracker along with 100 peer nodes. The simulation run for one hour with request arrival rate using random distribution and its corresponding factors such as start-up latency, seek latency, network throughput, simultaneous peers in action, and media server load are studied. Finally, every peer that contributes to our VoD system that increases its upload bandwidth as a profit has provided added multicast strength reducing load at server and solving penalized sending peer. The experimental study with the other two systems has been done with same peers reacting inefficiently to the same setup.

Table 2 describes the various video samples used for testing and its properties like time length and data rate. The number of segments for each video is fixed for the traditional system, whereas it is dynamically calculated for the proposed system with the available set of peer nodes formed in a group with its heterogeneous ability such as bandwidth rate and cache size as explained earlier. Dynamic splitting a video is done until the request for that video remains within the group. Table 3 provides the dynamic split of the same video v_1 stored at the same peer node at different bandwidth strength indicated as Sample 1 and Sample 2.

Table 3 gives the number of segments stored in each peer according to the algorithm present in the Server Helper (SH) which distributes segments based on peer's capacity.

Table 3: Segment split.

| S. no. | Storage peers | Sample 1 | | Sample 2 | |
|---|---|---|---|---|---|
| | | Bandwidth (kbps) | No. of segments | Bandwidth (kbps) | No. of Segments |
| 1 | P_B | 100 | 5 | 115 | 6 |
| 2 | P_C | 90 | 5 | 58 | 3 |
| 3 | P_D | 70 | 4 | 45 | 2 |
| 4 | P_E | 80 | 4 | 95 | 5 |
| 5 | P_F | 20 | 1 | 30 | 2 |
| 6 | P_G | 60 | 3 | 25 | 1 |
| 7 | P_H | 40 | 2 | 65 | 4 |
| 8 | P_I | 60 | 3 | 75 | 4 |
| 9 | P_J | 50 | 3 | 55 | 3 |

## 5. Results and Discussion

In this section, we put forth a test copy of the system that is suited for application such as VoD, based on set-top boxes as well as the multimedia content sharing at the social network systems. The fundamental approach to the reduction in service latency as well as providing optimal quality than any feasible solution is our concern. We prove this here with the several graphs and tables which provide proof to the system with the tracker and our QPMVM which is more advantageous than the systems that have very few factors working at the server end or the client end or at the proxy end. Since we combined all the factors from different viewpoint, this creates elements where the client side perception has more impact towards the betterment of the system as a whole. There are several results in the form of graphs and tables, but here we provide only a few of them to highlight our works aim.

Frames dropping is one concept to check for, in any service provider, such as our media server which services for video requests. On requests that come from peer clients, the sender's video bit rate should match that of the receiver's bit rate. If there is no match, frame dropping occurs when the sender's bit rate is more than the receiver's receiving capacity which happens due to congestion at the client network, and the bandwidth is clogged with the sender's data which the user could not process. This is present in all the server technology to drop frames and progress with streaming until it is uniformly completed in all multicast clients.

Here, in Figure 7, we see that the average number of frames dropped for video streams at 316 bits per second to 5 clients that receive at 56 kbps to 512 kbps. At 64 kbps, the client reception is very poor, and the loss of packets is comparatively more than that of 512 kbps reception, at which point all the clients receive the data packets without frames being lost. In Figure 7 it can be noted that the choking level is for lesser bit rate at client as seen from the graph. The table shows the difference in the three technologies discussed so far. Single version storage of video files drops few frames because it would be a lower version video with reduced picture quality that suffices even the least set of clients with very low connectivity. Also, it is seen that the

Table 4: Amount of memory used by different systems for different versioned files.

| Single version | Traditional multiple version | Our optimal solution |
|---|---|---|
| 609 KB | 2416 KB | 1403 KB |
| 10453 KB | 42113 KB | 22874 KB |

number of frames dropped by a single version is much less when compared to other multiple version systems. This is contrary to the fact that maintaining multiple versions is an optimal solution. The effectiveness of multiversioning is evident when we observe that the actual packets dropped are not the effective packets which we describe in the following paragraph.

In a single version system I, B, or a P frame could be dropped. But each one has a different level of importance. If the system drops a single I frame, then the corresponding full GOP is deemed to be useless. An important detail to note in Figure 3's graphical representation of the table is that the number of frames dropped is almost the same between the traditional multiversion scheme (that maintains every possible version for the client pattern) and our optimal solution. This indicates our optimal solution drops almost the same number of frames but it uses minimal storage as compared to the traditional scheme by setting a base version and few higher versions. It is also seen in Figure 3 that the frame drop in single version is less. To understand that the I, P, and B frames' corresponding priorities are taken into account in multiversioning, we plot a second graph as shown in Figure 4, which gives the effective number of packets dropped.

To understand the impact of the effective frame drop we must compare Figures 3 and 4. In Figure 3, we take the example where 3 frames are dropped in the single version (384 kbps file); we note that the corresponding effective frame rate is 28 frames. This is explained as follows:

Total no. of frames dropped: 3

Total no. of I frames dropped: 2

Total no. of P frames dropped: 1

Total no. of B frames dropped: —-

GOP : 12

If an I frame is dropped, none of the other frames can be decoded in the GOP, that is, dropping a single I frame effectively drops 12 frames. If a P frame is dropped, dependant B frames cannot be decoded. Hence,

No. of frames dropped because of I frames lost:

$2 \times 12 = 24$

No. of frames dropped because of P frames lost:

4 B frames

Total no. of effective frames dropped: 29 frames

Moreover, it is clear from Table 4 that a single version file requires lesser memory, but at the expense of video quality. Using all possible versions that satisfy the different client abilities in terms of its receiving minimal capacity, we find that this consumes enormous storage size for a single video file. Our optimized storage reduces the size for a single video content by approximately half of it and provides more or less same video quality similar to the multiversioned system. This is a major advantage of using the higher-versioned video at times of congestion to serve a slightly lower capacity client that is managed by our QPMVM system to drop off ineffective frames on the fly.

Table 5: Segment distribution in proposed P2P.

| Peer groups | Bandwidth (kbps) | No. of videos | Segments |
|---|---|---|---|
| A | 100 | 3 | v_1, v_2, v_3 |
| B | 80 | 3 | v_2, v_5, v_6 |
| C | 60 | 2 | v_1, v_4 |
| D | 45 | 1 | v_1 |
| E | 30 | 1 | v_5 |

Table 6: Segment distribution in t raditional P2P.

| Peer groups | Bandwidth (kbps) | No. of videos | Segments |
|---|---|---|---|
| A | 100 | 2 | v_1, v_2 |
| B | 80 | 2 | v_3, v_4 |
| C | 60 | 2 | v_1, v_3 |
| D | 45 | 2 | v_5, v_2 |
| E | 30 | 2 | v_4, v_5 |

Table 7: Total view time.

| No. of simultaneous users (name of peers) | Total viewing time(s) | | |
|---|---|---|---|
| | Proposed P2P | Client server | traditional P2P |
| 1 (B) | 481 | 469 | 640 |
| 2 (B,D) | 543 | 481 | 764 |
| 3 (A,B,D) | 584 | 806 | 1199 |
| 4 (A,B,D,E) | 723 | 1015 | 1254 |
| 5 (A,B,C,D,E) | 923 | 2634 | 1867 |

In traditional P2P system, the segments are equally distributed among the peers without the tracker system as shown in Table 4, and the proposed system has the distribution of clips based on the peer ability as shown in Table 5. For proposed, traditional, and client server systems, we measured the evaluation parameters given in Table 1.

*The Total Viewing Time of the Video for Peer Group B Is Observed as Follows.* From Tables 5 and 6, we have analyzed the total viewing time for 5 peer groups. As the number of simultaneous peers increases the time for viewing, the entire video increases. The variations among the three models are explained in the following section.

Table 7, we have analyzed the total viewing time for 5 peer groups.

*Total Viewing Time.* From the graph in Figure 5, we observe that when the percentage of simultaneous users at a point of time is very low (i.e., less than 40 users in this case),

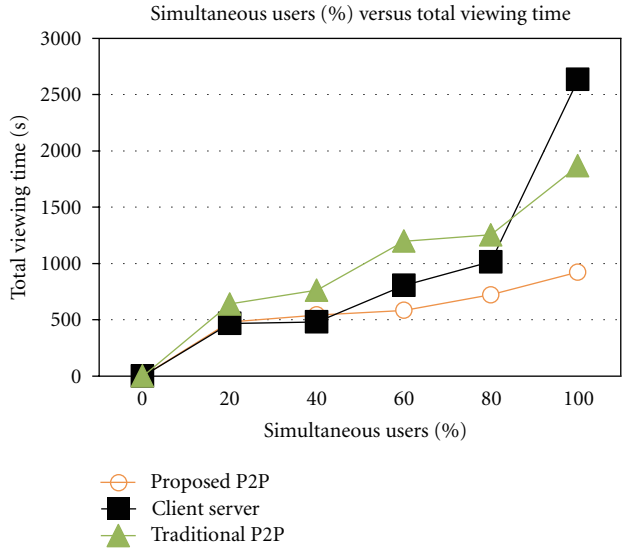


Figure 5: Percentage of simultaneous users versus total viewing time.

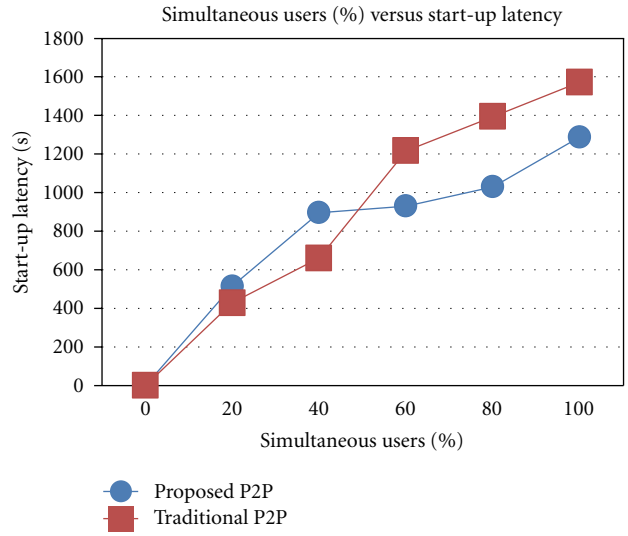


Figure 6: Percentage of simultaneous users versus start-up latency.

accessing all the segments from the server itself is preferable. This is because to access the segments from various peers, the user has to divide its bandwidth among the peers in the system. This initial joining time could be used to push video streams among the connected peers. In case of the client server model the entire bandwidth is dedicated to user. But as the number of users (simultaneous access) grows, the P2P model that we have proposed proves to be better. This can be explained as follows. As an example, 4 users request for viewing the video. (a) In client-server model, the server has to satisfy requests of all the clients in the network. This would obviously require more time. And also, the load on the server increases drastically. (b) Looking into the Proposed P2P model, the segments are shared among all the peers available. In this model, the bandwidths of all the peers are utilized. Thus, there is no overload problem in this case. This proves better for proposed system in comparison to the Traditional P2P in a way, because the segments are split according to the peer capacities the time for serving the requests for viewing is drastically reduced here.

*(1) The Start-Up Latency for % of Simultaneous Users Is Observed as Follows.* From the measured values, we observed that the start-up is comparatively less for our system than the traditional P2P model. Comparing the first two values, it is seen that there is decrease in the latency in the case of traditional P2P model. This is because the server has a bandwidth rate that is comparatively greater than a single peer in the network. Thus the time for serving a system decreases. This is the same with the startup latency as well. Figure 6 shows the startup latency for the various users occurring simultaneously.

*(2) The Seek Latency for % of Simultaneous Users Is Observed as Follows.* While measuring the seek latency, firstly, the proposed P2P (all users accessing different segments), we have peers from peer P_B request for the segment wk_7 from peer P_C. All other peers request for segments that are not in peer P_C. Secondly, the proposed P2P (2 users accessing same segments), peer P_B, and peer P_D request for the segment wk_7 from peer C simultaneously. All other peers request segments that are not in peer C. We could infer that seek latency is comparatively less for our system at the peak time. We efficiently offload the server at the peak time request for seek. The other peers have requests generated for various other clips that are not in sequence similar to P_B and P_D requests. Figure 7 shows the greater improvement of having tracker to manifest the VCR-based requests to locate and retrieve effectively in the proposed system.

Further, for explanations by looking at this case from a slightly different perspective, we have the users/clients requesting for a segment (same or each different) that is spatially ahead, and we find this method of combining multiple subsystems working together is more efficient for VCR (seek) than that given in [20, 29].

When the P2P Groups access different segments, multiple end users put forth seek requests simultaneously. But each end user request for a unique segment located at different peers. Thus, in this case the entire bandwidth of the storage peer that contains the requested segment is dedicated to that single requesting end user.

When the P2P Groups access different segments, multiple end users put forth seek requests simultaneously. But each end user request for a unique segment located at different peers. Thus, in this case the entire bandwidth of the storage peer that contains the requested segment is dedicated to that single requesting end user.

When the P2P groups access same segments, all the VCR requesting clients are made for the same segment at the same time. Thus the storage peer holding that requested segment has to share its bandwidth to serve the multiple end users. Utilizing the bandwidth of the many peers in the system model improves the performance when compared

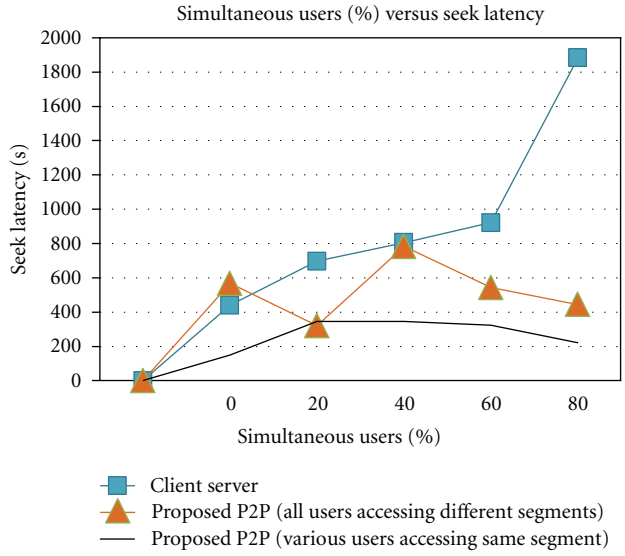


Figure 7: % of simultaneous users versus seek latency.

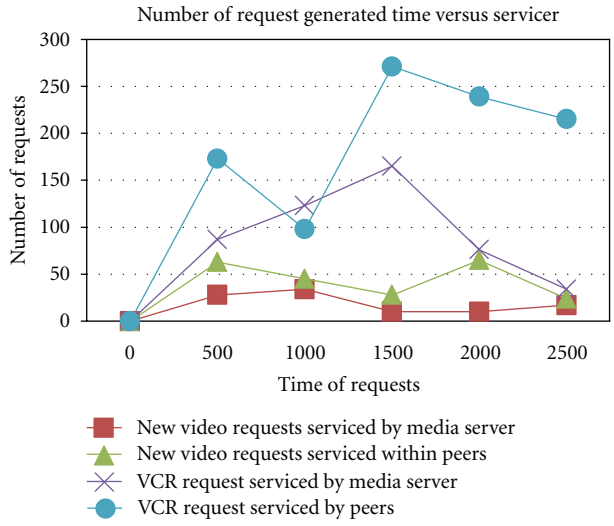


Figure 8: No. of request generated time versus service for the proposed system.

to dedicating (by dividing/sharing) the entire bandwidth of Server (Client-Server model) to all requesters (with others requests as well) [40].

In the traditional P2P system, the entire set of requests is sent to the server. This later identifies the clips present in the peer groups and reroutes them accordingly [29]. Be it requests for same segment or multiple segments, they have to be satisfied by the single system. Hence, there is a heavy increase in terms of load on the server. This in turn increases the seek time. In a nutshell, it deteriorates the performance of the complete model.

Figure 8, shows the overall requests as handled by our work which we mention here as the proposed system. The requests here are the requests for new video and VCR requests. The ability of this system to handle most of the

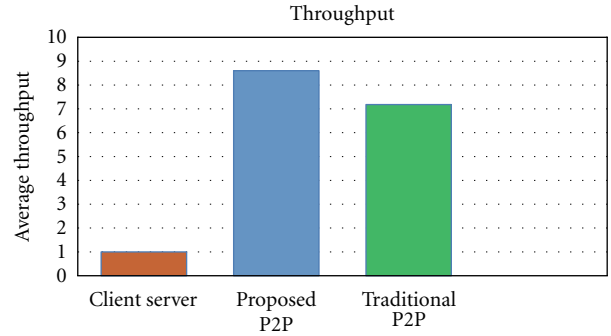


Figure 9: Throughput comparison.

requests with ease shows the utilization factor of the proposed work with the tracker and the optimized distribution at hand among the peers. The same when working with the traditional system showed 45% of the requests were handled by the media server whereas the proposed system made only 17% of the requests handled by the media server.

*Throughput Measure.* Performance tests for measuring throughput have been conducted, and the throughputs for the various models have been found to be as shown in Figure 9.

The graph in Figure 9 shows that the throughput of proposed P2P system is much better than a client server model and more similar to the traditional model. In client server model, since all requests are served only by the server, the other peers in the system are idle most of the time. In the proposed system, all the peers actively serve the requesting end user with the segments stored in it. Thus our system achieves high throughput and hence efficiently utilizes most of the peer resources. This is slightly greater than the proposed one as it leaves certain peers underutilized and several others overutilized. Hence the traditional P2P suffers a slight lower throughputthan our proposed P2P.

## 6. Conclusion and Future Work

Our proposed system with QPMVM, tracker in P2P VOD environment, has shown performance which utilizes various resource management techniques. It also understands performance limits in order to deploy adaptive QoS effectively in larger scaling systems.

*Major Contributions from This Work Are as Follows*

(i) A novel workable solution QPMVM for video streaming systems, especially for a VoD system, in place where there is provision of limited system resources and infrastructure. For example, systems that are possible only with lower bandwidth, less neighbor support and so on, could perform much better by employing our system (namely, e-learning with limited users).

(ii) Optimized usage of server bandwidth, reduced server load, storage, and prevalent better quality video at client by combining multiversion and multilayer system for videos at the server.

(iii) Stressing the video segment distribution combined with incentive mechanism solves idleness among peer sharers is another major advantage. This increases the usage of available distributed bandwidth among the social network groups with incentive benefits.

(iv) Design and integration of the entire system with performance study over various scenarios highlight the coordination and management rules provided by the various subsystems at the server, tracker, and the P2P nodes.

*Limitations of This Work*

(i) As this system is to provide better utilities for low profile consumers, sufficient video files that fit this category were considered for simulation. Hence, use of H.264 (SVC) is avoided/omitted.

(ii) If all the consumers request different videos at the same instant, then the peer structure would perform poorly, because there would be no possibility for initial sharing and thereby increasing the seek latency.

(iii) The cache store and replacement policies worked with are optimal in their usage but could further be altered to suit the peer sharing ability with their profiles in mind. Peer churns must be handled for real time efficiency.

This work supports all MPEG file formats including MPEG-4 part 10 (AVC); future enhancements could include usage of H.264 (SVC) files that could provide remarkable solutions that can be studied. Advanced caching policies could be fitted into the framework to upgrade the system.

## References


[1] C. Zhijia, L. Chuang, and W. Xiaogang, "Enabling on-demand internet video streaming services to multi-terminal users in large scale," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1988–1996, 2009.

[2] A. Raghuveer, N. Kang, and D. H. C. Du, "Techniques for efficient streaming of layered video in heterogeneous client environments," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM '05)*, vol. 1, pp. 245–250, December 2005.

[3] Y. Yang, A. L. H. Chow, L. Golubchik, and D. Bragg, "Improving QoS in BitTorrent-like VoD systems," in *Proceedings of the IEEE International Conference on Computer Communications (IEEE INFOCOM '10)*, San Diego, Calif, USA, March 2010.

[4] B. Giovanni, S. Thomas, and A. Luigi, "Theoretical models for video on demand services on peer-to-peer networks," *International Journal of Digital Multimedia Broadcasting*, vol. 2009, Article ID 263936, 8 pages, 2009.

[5] A. G. Nemati and M. Takizawa, "Application level QoS in multimedia peer-to-peer (P2P) networks," in *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications Workshops/Symposia (AINA '08)*, pp. 319–324, March 2008.

[6] L. Bo, C. Yanchuan, C. Cui Yi, X. Yuan, Q. Fan, and L. Yansheng, "Minimizing service disruption in peer-to-peer streaming," in *Proceedings of the IEEE Computer Communications and Networking Conference (CCNC '11)*, pp. 1066–1071, 2011.

[7] Y. Lingjie, G. Linxiang, Z. Jin, and W. Xin, "SonicVoD: a VCR-supported P2P-VoD system with network coding," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, pp. 576–582, 2009.

[8] H. Cheng-Hsin and H. Mohamed, "Optimal coding of multilayer and multiversion video streams," *IEEE Transactions on Multimedia*, vol. 10, no. 1, pp. 121–131, 2008.

[9] G. M. Muntean, G. Ghinea, and T. N. Sheehan, "Region of interest-based adaptive multimedia streaming scheme," *IEEE Transactions on Broadcasting*, vol. 54, no. 2, pp. 296–303, 2008.

[10] H. Yu, E. C. Chang, W. T. Ooi, M. C. Chan, and W. Cheng, "Integrated optimization of video server resource and streaming quality over best-effort network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 3, pp. 374–385, 2009.

[11] L. Jiangchuan, B. Li, and Z. Ya-Qin, "Adaptive video multicast over the internet," *IEEE Multimedia*, vol. 10, no. 1, pp. 22–33, 2003.

[12] S. McCanne, M. Vetterli, and V. Jacobson, "Low-complexity video coding for receiver-driven layered multicast," *IEEE Journal on Selected Areas in Communications*, vol. 15, no. 6, pp. 983–1001, 1997.

[13] L. Jiangchuan, B. Li, and Y. Q. Zhang, "Optimal stream replication for video simulcasting," *IEEE Transactions on Multimedia*, vol. 8, no. 1, pp. 162–169, 2006.

[14] T. C. Thang, J. W. Kang, J. J. Yoo, and Y. M. Ro, "Optimal multilayer adaptation of SVC video over heterogeneous environments," *Advances in Multimedia*, vol. 2008, Article ID 739192, 8 pages, 2008.

[15] C. Yan, F. Toni, and Y. Nong, "QoS requirement of network applications on the Internet," *Proceedings of Information, Knowledge, Systems Management*, vol. 4, no. 1, pp. 55–76, 2004.

[16] R. A. X. Annie and P. Yogesh, "VoD system: providing effective peer-to-peer environment for an improved VCR operative solutions," *Communications in Computer and Information Science*, vol. 106, no. 2, pp. 127–134, 2010.

[17] F. V. Hecht, T. Bocek, and B. Stiller, "B-Tracker: improving load balancing and efficiency in distributed P2P trackers," in *Proceedings of the 11th IEEE International Conference on Peer-to-Peer Computing (P2P '11)*, pp. 310–313, 2011.

[18] C. Liang, Z. Fu, Y. Liu, and C. W. Wu, "Incentivized peer-assisted streaming for on-demand services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 9, pp. 1354–1367, 2010.

[19] T. Guo and Y. Zhang, "Research of incentive mechanisms in P2P-based Video on Demand System," in *Proceedings of the 2nd International Conference on Networking and Distributed Computing (ICNDC '11)*, pp. 340–343, 2011.

[20] J. M. Dyaberi, K. Kannan, and V. S. Pai, "Storage optimization for a peer-to-peer video-on-demand network," in *Proceedings of the ACM SIGMM Conference on Multimedia Systems (MMSys '10)*, pp. 59–70, February 2010.

[21] D. Tursun and W. Liejun, "Adaptive stream multicast for video in heterogeneous networks," *Information Technology Journal*, vol. 8, no. 2, pp. 246–249, 2009.

[22] D. Wang and J. Liu, "Peer-to-peer asynchronous video streaming using skip list," in *Proceedings of the IEEE International*

*Conference on Multimedia and Expo (ICME '06)*, pp. 1397–1400, July 2006.

[23] E. Tan and C. T. Chou, "Frame rate control for video streaming," in *Proceedings of the 36th Annual IEEE Conference on Local Computer Networks (LCN '11)*, pp. 163–166, 2011.

[24] S. F. Chang and A. Vetro, "Video adaptation: concepts, technologies, and open issues," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 148–158, 2005.

[25] F. Takaya, E. Rei, M. Kei, and S. Hiroshi, "Video-popularity-based caching scheme for P2P video-on-demand streaming," in *Proceedings of the 25th IEEE International Conference on Advanced Information Networking and Applications (AINA '11)*, pp. 748–755, March 2011.

[26] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts," *Multimedia Systems*, vol. 9, no. 2, pp. 170–184, 2003.

[27] H. Byun and M. Lee, "A tracker-based P2P system for live multimedia streaming services," in *Proceedings of the 13th International Conference on Advanced Communication Technology: Smart Service Innovation through Mobile Interactivity (ICACT '11)*, pp. 1608–1613, February 2011.

[28] C. Jia Ming, L. Jenq Shiou, C. Yen Chiu, W. Hsin Wen, and S. Wei Kuan, "MegaDrop: a cooperative video-on-demand system in a Peer-to-Peer environment," *Journal of Information Science and Engineering*, vol. 27, no. 4, pp. 1345–1361, 2011.

[29] I. Radulovic, P. Frossard, and O. Verscheure, "Adaptive video streaming in lossy networks: versions or layers?" in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '04)*, vol. 3, pp. 1915–1918, Taipei, Taiwan, June 2004.

[30] P. Seeling and M. Reisslein, "Video transport evaluation with H.264 video traces," *IEEE Communications Surveys and Tutorials*, no. 4, pp. 1–24, 2011.

[31] L. Tionardi and F. Hartanto, "The use of cumulative inter-frame jitter for adapting video transmission rate," in *Proceedings of the Confernce on Covergent Technologies for the Asia-Pacific Region (IEEE TENCON '03)*, pp. 364–368, October 2003.

[32] R. Mahindra, R. Kokku, H. Zhang, and S. Rangarajan, "MESA: farsighted flow management for video delivery in broadband wireless networks," in *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS '11)*, pp. 1–10, January 2011.

[33] D. Gangadharan, H. Ma, S. Chakraborty, and R. Zimmermann, "Video quality-driven buffer dimensioning in MPSoC platforms via prioritized frame drops," in *Proceedings of the IEEE 29th International Conference on Computer Design (ICCD '11)*, pp. 247–252, 2011.

[34] A. A. Sofokleous and M. C. Angelides, "DCAF: an MPEG-21 dynamic content adaptation framework," *Multimedia Tools and Applications*, vol. 40, no. 2, pp. 151–182, 2008.

[35] J. Annesley, G. Bäse, J. Orwell, and H. Sabirin, "An extension of the AVC file format for video surveillance," in *Proceedings of the 3rd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC '09)*, pp. 1–8, September 2009.

[36] P. Amon, T. Rathgen, and D. Singer, "File format for scalable video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9, pp. 1174–1185, 2007.

[37] T. L. Lin, J. Shin, and P. Cosman, "Packet dropping for widely varying bit reduction rates using a network-based packet loss visibility model," in *Proceedings of the Data Compression Conference (DCC '10)*, pp. 445–454, March 2010.

[38] C. Hu and C. Tu, "Research on P2P incentive mechanism," in *2010 International Forum on Information Technology and Applications (IFITA '10)*, vol. 1, pp. 47–50, July 2010.

[39] Y. W. Wong, J. Y. B. Lee, V. O. K. Li, and G. S. H. Chan, "Supporting interactive video-on-demand with adaptive multicast streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 2, pp. 129–141, 2007.

[40] Q. Wei, T. Qin, and S. Fujita, "A two-level caching protocol for hierarchical peer-to-peer file sharing systems," in *Proceedings of the IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA '11)*, pp. 195–200, 2011.