Implementation of VoD P2P System Based on the LCBBS Module

Nima Bina

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
January 2012
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Muhammed Salamah
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asoc. Prof. Dr. Işık Aybay
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Isik Aybay
2. Assoc. Prof. Dr. Muhammed Salamah
3. Asst. Prof. Dr. Gürcü Öz

ABSTRACT

In a common VoD system, a large bandwidth is required for the distribution of videos over the Internet. One promising solution is to use a “peer to peer” Video on Demand (VoD) system, in which peers cooperate in the distribution of videos. This solution decreases the server traffic; so the system works faster in a peer to peer VoD architecture.

Video Locality Based Buffering Mechanism (LCBBS) is using this idea in an efficient methodology to get better performance in peer to peer VoD systems. In this thesis, a modified version of a LCBBS module is implemented and tested over a real test bed VoD application. The details of this implementation and the results of related experiments are also provided.

This implementation has two major modules; TCP and UDP. The TCP module is implemented to work as a standalone video server on the Internet to distribute video segments to all clients. The TCP module has a buffer to store video segments on the local machine of each client. The UDP module helps system to distribute video segments over a high speed LAN connection without any interaction with server.

Keywords: Video streaming, Video on Demand, Client side buffering, Peer to Peer VoD**,** Distributed VoD, LCBBS.

ÖZ

Bir Video İstemci Sisteminde (VoD) videoların sunucu (server) bilgisayarlar tarafından Internet üzerinden dağıtılabilmesi için geniş bir bant gerekir. Bu konuda ümit verici bir çözüm, eşdüzey ögelerin (peers) birbirine destek olacağı bir VoD sisteminin kurularak kullanılmasıdır. Bu sistemde eşdüzey istemci (client) bilgisayarlar, birbirlerine video sağlamada destek olurlar. Böylece sunucu (server) üzerine binen trafik yükü azaltılabilir.

Videoların mekana bağlılığına dayanan bir tampon bellek sistemi olan LCBBS sistemi, bu fikri eşdüzey ögelerin bulunduğu VoD sistemlerinde daha etkin olarak uygulamak için geliştirilmiş bir yöntemdir. Bu tezde, bir LCBBS modülünün orjinal öneriye gore bir ölçüde değiştirilmiş bir uyarlamasının gerçek bir VoD deneme ortamında fiziksel uygulaması ve testi yapılmıştır. Uygulamanın detayları ve ilgili deneylerin sonuçları da sunulmuştur.

Bu uygulamada iki ana modül kullanılmıştır. Bunlar TCP ve UDP protokolleri için hazırlanan modüllerdir. TCP modülü, kendi başına çalışabilecek ve Internet üzerinden video segmentlerini tüm kullanıcılara iletebilecek bir video sunucusu olarak tasarlanmıştır. Bu modülün her eşdüzey bilgisayarın üzerinde video segmentlerini depolayabilecek bir tampon belleği yer almaktadır. UDP modülü ise, sistemin video segmentlerini her hangi bir sunucu ile etkileşim gerektirmeden, yüksek hızlı bir yerel ağ üzerinden (LAN) dağıtmasını sağlamaktadır.

Anahtar kelimeler: Video akışı, Video istemci sistemi, İstemci tarafı tampon bellek, Eşdüzey ögeler arası VoD**,** Dağıtık VoD, LCBBS.

To My Family

ACKNOWLEDGMENT

I would like to dedicate this thesis to my mother, Mahin, for her love understanding and support during this entire process. She has been a source of encouragement and personal inspiration to me and her determination to make me realize my full potential throughout has made this possible.

I would also like to express my deep appreciation and gratitude to my supervisor Dr. Isik Aybay, for giving me the opportunity to be a part of such exciting research. His time, patience, and support were crucial in getting this thesis completed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Peer – to – peer (P2P) VoD system

Video on demand (VoD) is a popular internet service for watching videos in a real time manner. With increasing bandwidth of Internet access, Video on demand (VoD) is becoming extremely popular. For example, YouTube, a video distribution server that uses Video on demand (VoD) is visited by millions of people every day [1].

The traditional VoD system architecture is based on a server and a number of clients that are connected to the server for downloading the video through the Internet. However when a large number of concurrent users are requesting videos from the server, a high bandwidth is required to stream videos to all the users in real – time. Using a peer – to – peer (P2P) VoD system is one solution for real – time video streaming. It can handle a large number of users by letting peers to cache some parts of the video. In this system, peers can provide on demand service to other clients. So each peer is not only a receiver, receiving videos from the server, but is also a provider, providing the buffered video content to other peers.

In a peer – to – peer (P2P) VoD system, each client helps the server by sharing video segments, and streaming these segments to other peers through the network to decrease the TCP traffic of server. Peer – to – peer (P2P) VoD systems improve the performance of video sharing over the Internet by adding peers to help the server to

share the video files. For example, this system can also be applied in a distance education field in which the transfer rate of video and file sharing between users and server is very high.

Many ideas are suggested for peer – to – peer (P2P) VoD mostly with the same core system, but with some differences on, how to segment the video and share it through the network. The Video Locality Based Buffering Mechanism (LCBBS) [2] is one of the techniques that can be used to setup a peer – to – peer (P2P) VoD system. The LCBBS module is simulated and has been shown to perform well in terms of startup latency, stops and wait times during the video clip play.

The LCBBS module is based on a peer – to – peer (P2P) VoD system in which other clients (peers) help system to reduce the server traffic. In a peer – to – peer media streaming system, a part of media should start playing while the rest of it is downloading. More specifically, in a media sharing system, like LCBBS clients (peers) have some media segments and they share the segments with other requesting clients (peers); on the other hand, the requesting peers playback and store the media data segments during the streaming session.

In this thesis, an implementation of a modified LCBBS module is developed. Some modifications are done to make the software applicable to real cases. In simulation, there is no limitation of using a TCP connection or a UDP connection, Video player limitation and etc. However, in physical implementation, some limitations are required to integrate the system in an acceptable way. Perhaps the reminder of this thesis is organized as follows. Chapter 2 describes some similar work to our software and it is compared with LCBBS main idea. In Chapter 3 the LCBBS module is

described in details. The details of our implementation are also provided in this chapter with some advantages of our methodology. Chapter 4 describes the experimental studies of the implemented software in both the TCP and UDP modules with some comparison between the theoretical aspects. The test bed of the system is also provided in this chapter. Chapter 5 is the conclusion section that concludes our implementation and the idea.

Chapter 2

LITRETURE SURVEY STUDIES

2.1 Overview of Related Works

A significant surveys of peer – to – peer systems is given in [3] [4] [5] [6]. In [3], "Y.Zhang and H.Wang" proposed a basic model for a large – scale P2P VoD system that includes a large number of peers with a server. They proposed a system model in which a file is divided into number of segments, and encoded for playback at a specific rate. In their system, the server stores all segments of the file with the assumption that the file is able to meet any needs from peers. They also assumed that, time is slotted, so during each time slot, a peer can simultaneously provide a limited number of upload connections at most. Each peer in their system is not only downloading data, but it can also upload data for other peers.

Simulation assumptions of [3]are:

1. All peers have identical configuration parameters.

2. Peers participate with the system and perform a complete in – order playback, and leave the system once finishing.

3. Playback rate is assumed to be 500Kbps.

The results showed that peers assisting each other in the system and they can handle the system when the server traffic is large. Also they showed that the file duration brings a minor effect on the system.

In another study [4], "Y.He and L.Guan" proposed a system architecture in which some "helper" contribute in the system to decrease the server traffic. "A helper is a node who is not interested in watching the video but is willing to use its idle bandwidth to download the video at a small rate and then forward the received rate to other peers" [4].

The proposed system in [4] has 3 types of elements: 1) the server, 2) the peers and 3) the helpers. Their overall architecture is given in Figure1, below.

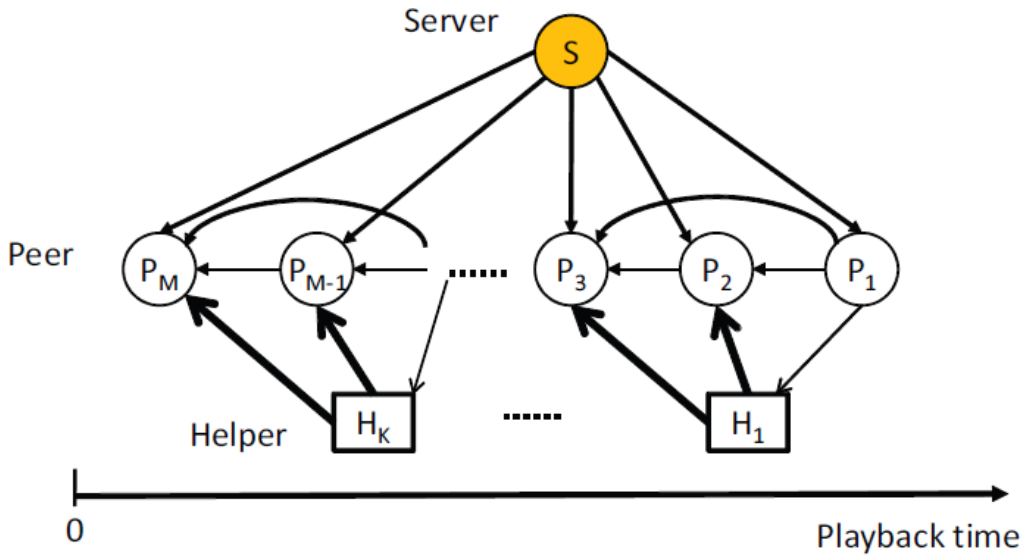

Figure 1. Illustration of proposed system architecture of [4]

Figure 1 shows that each peer has a connection to the server without any intermediate node. The helper is a node or a peer that is in an idle status so it is a candidate to help the system for video sharing.

Simulation assumptions of [4] are:

1. Server upload capacity is 3 Mbps.

2. Two class of peers 1) cable/DSL and 2) Ethernet peers

3. 85% of whole system peers are assumed that connected by cable/DSL with download rate between 0.9 – 1.5 Mbps and upload rate between 0.3 – 0.6 Mbps.

4. 15% of the whole system is assumed that connected by Ethernet connection with download and upload rate between 1.5 Mbps and 3.0 Mbps.

5. The number of the helpers in system is assumed to be 10% of nodes.

With the above assumptions they simulated their system with different conditions and comprised the obtained results. They proved that the helpers utilize the upload bandwidth of the whole system, and they showed that their system improved the streaming capacity comparing to a peer – to – peer VoD system without any helpers.

In another study by "K.Wang and C.Lin" [5], an architecture of a peer – to – peer VoD system is proposed with the following assumptions:

1. Each peer is asked to contribute a fixed amount of hard disc storage. The entire viewer population thus forms a distributed P2P file system.

2. A movie is divided into a number of media chunks, and initially the chunks are made available at an origin content server.

3. A peer, when start to watch a movie, obtains from the tracker a list of peers currently have that file. The peer then establishes neighborhood relationships with the peers on the list.

4. The peer asks its neighbors for their Chunk Bitmaps telling which chunks a peer has. Based on this information, the peer requests video chunks from its neighbors and from the server in case there is not enough replicas.

5. Once a peer has obtained a chunk, it informs its tracker that it is replicating that movie; conversely, a peer also tells its tracker when it no longer holds a movie in its cache.

The PPLive system has the following major components:

1. A bootstrap server that help peers to find a suitable tracker and to perform other bootstrapping functions.

2. A set of video servers as the source of content.

3. A set of trackers that help peers connect to others to share the same content.

The PPLive system has some other sub – server that the tracking tasks for video segments are handled by “Tracker Server”.

Figure 2 shows the system architecture schema of the PPLive system:

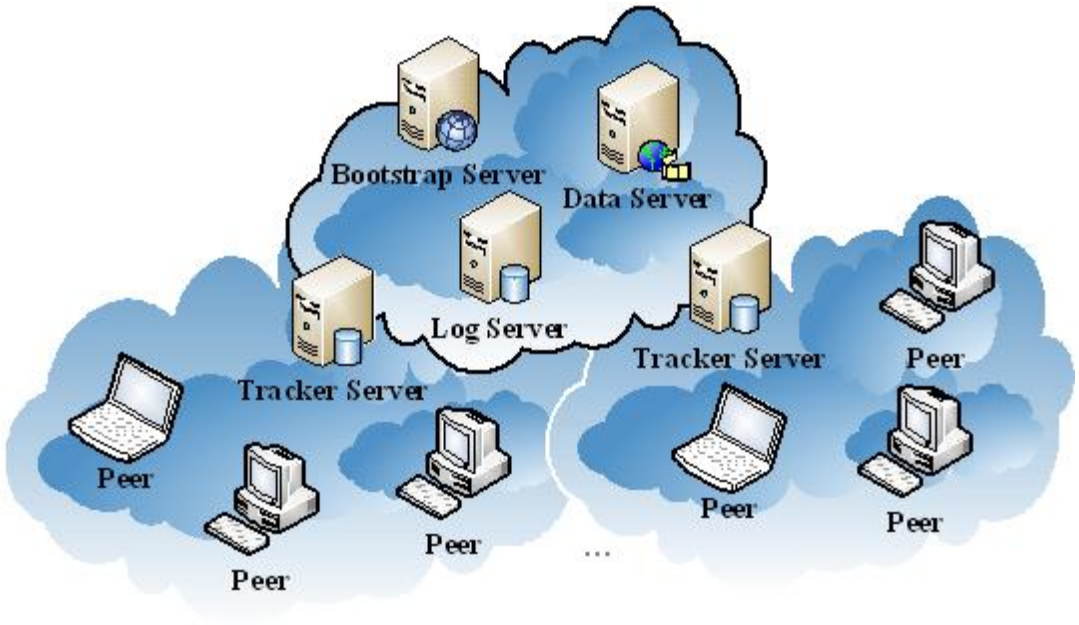

Figure 2. PPLive P2P – VoD system architecture. [5]

They have simulated their system and obtained some results that showed the system's performance by comparing the upload bandwidth in two cases 1) if a client connect to the server without any help of downloading form other peers and 2) some peers are helping to upload videos to the requested client.

In another study by "Z.Lu, S.Zhang and J.Wu" [6], an architecture of a peer – to – peer VoD system is proposed with the following characteristics:

1. Unicast IP is used at network layer.

2. Asynchronous connections of clients are handled by some utilizations methods.

3. A variable FIFO buffer size is designed for each client.

4. By checking the out – band bandwidth, each client can forward the media file segments to a new client.

Figure 3 shows the system architecture of P2P – VoD – File Segment Selection with just one central VoD server:

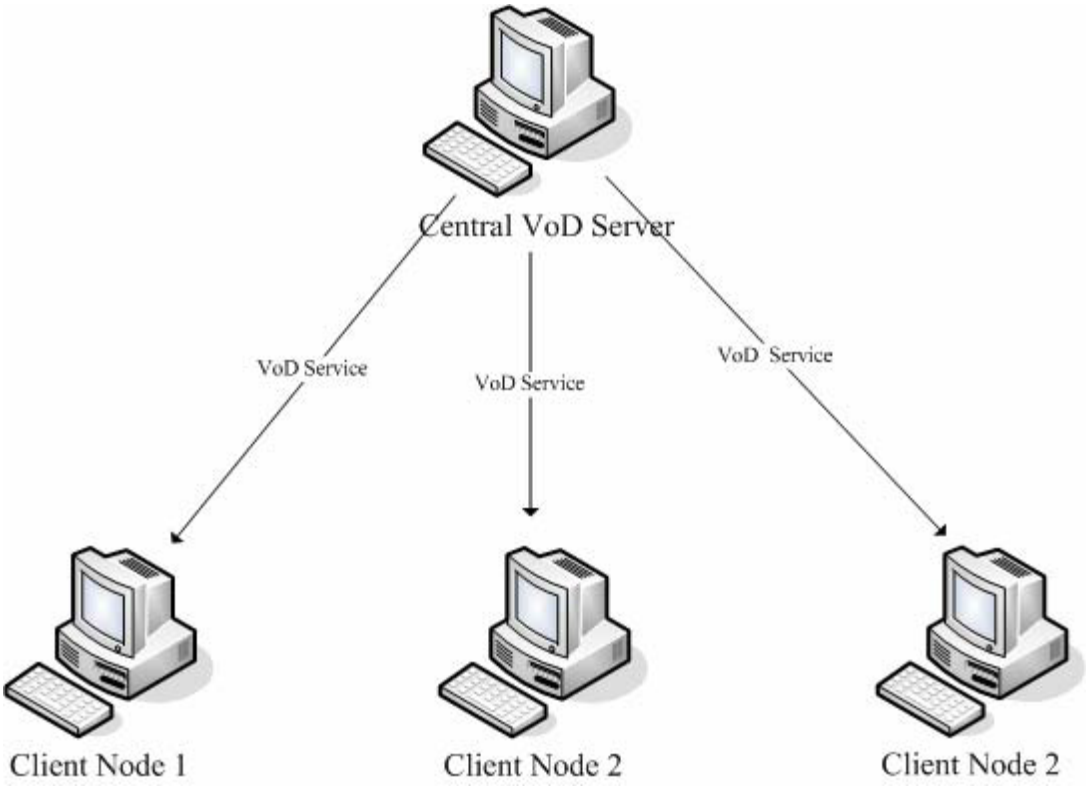

Figure 3. The P2P – VoD – FSS system architecture with the VoD central server. [6]

Figure 3 indicates that the clients are connected with a dedicated connection to the server.

The other node that is shown in Figure 4 is “Strong Node” that works like a smaller server, which just provides media file, not requests and consumes media files.

Figure 4 shows the system architecture of “P2P – VoD – FSS” with strong nodes:

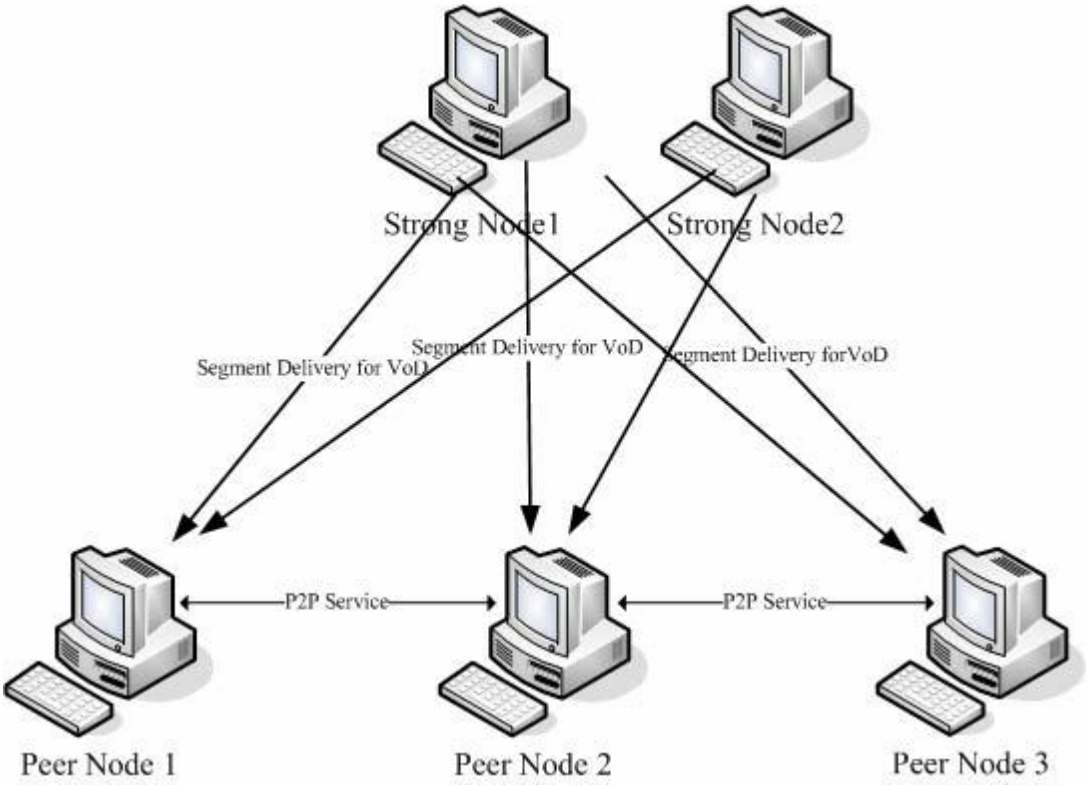

Figure 4. The P2P – VoD – FSS system architecture (with strong nodes). [6]

They implemented their system by applying a video segmentation with the size of 200 KB and 100 Mbps of LAN bandwidth. The results showed that by increasing the number of peer nodes in the P2P VoD system the performance increases comparing to a VoD central server.

2.2 Comparison of Similar Works with the LCBBS

The proposed system in [3], is based on a peer – to – peer video on demand system in which a video file is segmented to a specific number of segments, and then encoded according to the media playback rate. However in LCBBS, as it is illustrated in Chapter 3, the segmentation of video files is based on the binary size of each segment, so there is no need for encoding and decoding segments. Our methodology can be compared with [6], in which a media file is segmented to 200 KB while in our LCBBS system the size is 500 KB for each video segment.

In [4], some helpers are introduced which are used as an idle peer to help the server for video distribution to other clients. Our LCBBS system also uses a similar structure in which some idle clients share video segments with other peers.

Unlike [5 and 6], where some other sub – server helps the whole system for tracking or sharing video segments to other clients, in the LCBBS system and [4], the task of tracking segments and peers is done with a central server (video server).

[3, 4 and 5] just support their idea with simulation software, but in this thesis and also in [6], respective system are implemented in a real test – bed and the simulation results obtained are compared with the physically implemented software results.

Chapter 3

LCBBS MODULE

3.1 LCBBS Module Specification

The LCBBS is a buffering module that improves the performance of a peer – to – peer VoD system. The main LCBBS concept is that video files are divided to a number of segments, which are stored on the server located on the Internet and also at peers. Each peer, before requesting a video from the server checks the following

1. Its own buffer for segments of a video. If there are some segments, they are not requested from the peers or the server.

2. Other peer's buffers for needed segments of a video. If there are such segments, they are requested from the peers who own them.

If both of the above conditions fail, the client will request video segments from the server. The simulation results of LCBBS module [2] indicated that the performance of the resulting VoD system is improved, especially in terms of startup latency.

3.2 LCBBS Overview

LCBBS module is both a client and a server side module which runs on each client (peer) machine and performs two tasks,

1. It has a protocol in which the local machine search is carried out. If the requested video is not available, first the peers are checked and then the contact with the server on the Internet is provided.

2. The LCBBS module caches the video segments that are previously downloaded and displayed, for a possible later request of other peers.

Figure 1 shows the LAN client of the LCBBS module;

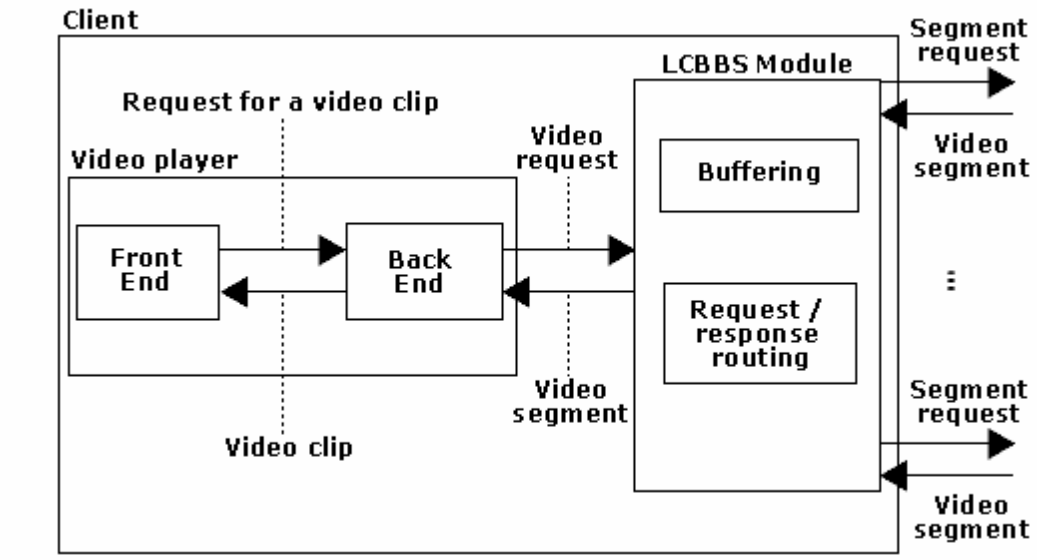

Figure 5. LAN Client with LCBBS Module [2]

The LCBBS module takes place on the back – end of the video player, and it is responsible for routing the video segments.

Generally, storing whole video files needs a very large volume of secondary memory. The LCBBS module divides a video file to a number of segments with a fixed size and stores some leading segments of a video clip in client buffers.

3.3 Implementation

Some ideas that occur logically in simulation may not be able for implementation in reality. In this section, first implementation problems are illustrated with the solution methodology, and then detailed information about the real system is provided.

3.4 Segmentation Problems

One of the common video streaming systems is Real Time Messaging Protocol (RTMP) [7], which was initially developed by Macromedia [8] for streaming audio and video over the Internet between Flash player [8]and server. In these systems, the segmentation of the video file is done automatically by the system and the video player should be capable to read the stream from the server in such a way that the stream is created by the server. However, when the streaming method of the server is unknown, identifying the stream in client side and segmenting it to a desirable segment is almost impossible. The solution of this problem is to create the own video streaming server which can be done in two possible ways:

1. Decoding the video file and segmenting the video according to video frames and encoding the video and saving the segment.

2. No decoding and encoding, but segmenting the files based on their size.

Video files such as MP4, FLV, AVI, etc. are encoded and compressed with different algorithms. If the first option is chosen, for each format, a complex encoding and decoding should be done before segmenting a video. Also, video players are not playing a video segment that does not have a correct header for each segment, so for each segment the header should be added.

The other choice, (2) is an easier method, but it has a big problem that is discussed on the next section.

3.5 Player Compatibility

After finding a solution for segmenting a video file and making it ready to be transferred over the network, the player should be compatible with our methodology to play the segmented video in a correct manner.

Most of the video players accept either a streaming server or a complete file as input. Since our system is dividing video files to segments, the client should gather segments from different locations so, we are not able to give one video streaming server to the player or a complete video file while the whole file is not accessible at the beginning of play.

As a solution to all above problems it is better to segmenting video files without considering their format, and hence, eliminating waste of time on decoding and encoding. Also a fixed length of size should be selected for all segments on the server machine and each segment cannot be played separately. On the client machine, after requesting the segments, a video file with known format and size will be created and it is left as empty. The address of the created empty file is then given to the video player, while the segments are simultaneously downloaded from the

server or other peers. To avoid any crash of the video player, the total number of segments should be known and the player should be paused before reaching the end of downloaded segments.

3.6 Advantages of our methodology

1. No need for complex encoding and decoding algorithms.

2. No need to add header and meta data for each segment.

3. Much faster since less metadata is kept by segments.

4. The file format is not important for the system, so all video and audio formats can be played on the client side if the video player is capable to play the format

The only problem is, while the video player is reading the binary data of the video file, the system fills the file in advance with segments. This condition needs a video player that can accept a video file as input to read the content while the other process is writing to the file.

We tested the Windows Media Player [9] as our first try to see if the player works with an opened file or not. Windows Media Player opens a file and does not let any other process to work with the file, so it was not applicable. Our second player was the VLC media player [10] that gives an ability to access the file which is opened by the player.

VLC is a well-known media player that is capable of playing the following files [10]

Table 1: Files acceptable by VLC [10]

MPEG (ES,PS,TS,PVA,MP3)
AVI
ASF / WMV / WMA
MP4 / MOV / 3GP
OGG / OGM / Annodex
Matroska (MKV)
Real
WAV (including DTS)
DTS, AAC, AC3/A52
Raw DV
FLAC
FLV (Flash)
MXF
Nut
Standard MIDI / SMF

As long as our methodology doesn't relate to a special kind of file format, all above formats can be shared and played in our system.

3.7 Implementation Details

The implemented LCBBS module can be divided into two major parts (sub - modules), TCP and UDP. Both TCP and UDP modules are located at the back end of the video player and completely control and handle all tasks of the system. The task of the TCP module is to make a connection to server that is located on the Internet while it should be able to send and receive packets from server to client and vice versa. The TCP module separately should be able to handle a client request and it caches the segments and plays the video file on the client. The TCP module is working like a Real Time Messaging Protocol (RTMP) system (which is illustrated

in the beginning of chapter 3). In our methodology; it means that a client can request a video clip, then video segments are completely transferred to the client through the TCP connection, and the video appears on the screen.

In both the client and the server, the same packet structure is used, which handles all information that is needed in each step of the process. The following figure shows the structure of a packet.

Figure 6: Packet Structure

1. moviesList (populated with server from database)
2. localSegments (populated by client - available segments on the client disk -)
3. neededSegments (populated by client)
4. availableSegments (populated by server)
5. SegmentData (populated by client and server for sending and receiving binary data of a segment)
6. ClientIP (populated by client)
7. ProcessStep (populated by client and server- indicates the level of processing)

This packet is a class that is shared by the client and the server. To minimize the size of the packet, each time the required information needed, will be populated by the server or the client and the other information is left as null.

The following figure shows the class packet parameters:

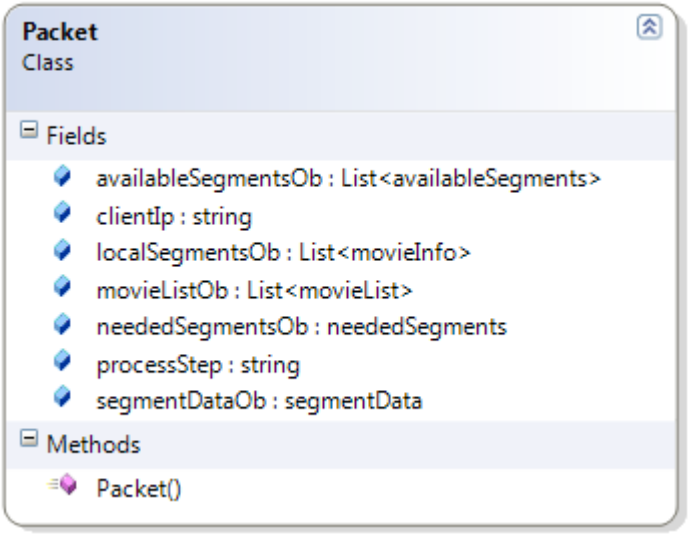

Figure 7: Packet Class

The packet, itself is a class that makes some new objects from other classes with their fields according to server and client information.

In the following figures, each Subclass is illustrated.

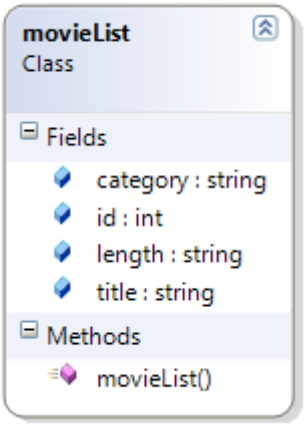

Figure 8: movieList Class

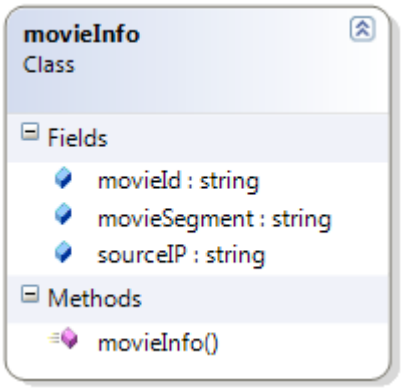

Figure 9: movieInfo Class

The “movieList” class contains the list of all movies that are stored on the server database. This class is populated by the server and when a client tries to connect to the server, after handshaking, this class (“movieList”), will be seriallized and sent as a part of the packet to the client. When the client recieves the packet, it deserializes the packet to check the contents. As long as the system is “asynchronous”, both sides of the connection are not waiting to get a response from the other side. The class “ movieInfo” is populated by client, with the information of all available segments on the secondary memory, and it will be sent to the server. The server stores all information of each connected clients in memory untill the user is connected to the system.

The “neededSegments” class is populated by the client when a user clicks on a video clip to be shown on the screen. The LCBBS system checks the available segments on the local machine, and findout the missing segments of the video file.

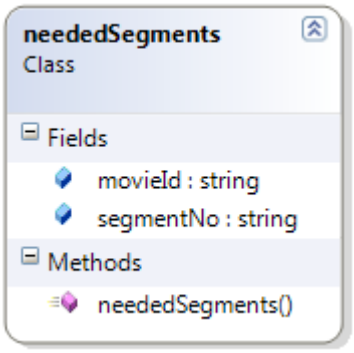

Figure 10: neededSegments Class

After determining the requested segments of the client on the server side, a check of all connected client's segments will be done by the server and preferable clients LanIP with movie ID and segment number will be populated to the "availableSegments" class and sends to the client as a packet.

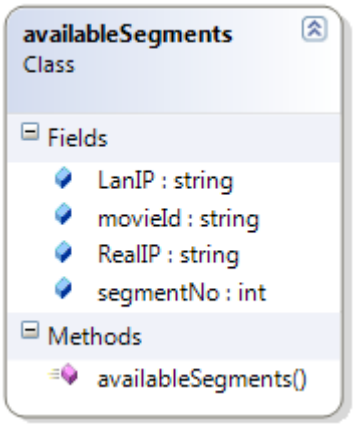

Figure 11: availableSegment Class

The "availableSegments" class is a class that is populated by both the server and the clients. On the client side, when the "availableSegments" is known, the segments are requested from the source IP one by one. The "segmentData" class will be sent from the server with all the information about the video clip to the other side of the connection. This class can be used as a TCP or UDP connection. If IP of the needed segment is located on UDP, the client sends "segmentData" packet with the empty

“segmentByte” field and waits for the binary data of the file from the other peer. In case of any failure, The same class will be sent to the server with an empty “segmentByte” field after some timeout.

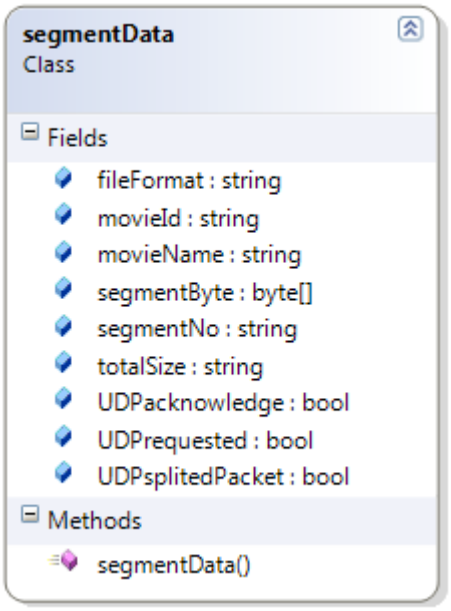

Figure 12: segmentData class

The process step is checked at the receiving point (client or server) in order to understand what information is handled with the packet. As an example, in the first send and receive between the server and the client, the server populates and sends “movieList” with process step as one packet to the client. On the client side, after checking the “ProcessStep” and understanding that packet has the movie list, the “movieInfo” is populated with “ProcessStep” and the packet is send back to the server.

3.8 Serializing and deserilizing

For sending a class over a TCP or UDP connection, the other side should have the same class schema for recognition. As mentioned before, our packet is a class which is created and shared by both the server and the client. This class is a serializable class that can be easily changed to an array of bytes to be transferred over the network. On the other side, by deserializing the array of bytes, the class is readable.

3.9 TCP module

The TCP module is created in such a way that a server can serve all video clips to any clients that are connected to the system separately. The TCP module is capable of accepting up to one hundred clients. The limitation of the acceptance of clients is related to the Internet bandwidth, so it can be increased or decreased according to the actual Internet speed.

The TCP module assumes the server is a standalone video server that can work like other RTMP servers in another methodology, to share videos over Internet.

The following pseudo-code illustrates the functionality of the client and the server in the TCP module.

1) Client lists the available segments on disk (before making connection to server).
2) Server trims the video files and starts to listen.
3) Client creates asynchronous TCP connection and waits to get acceptance from the server

   If (server accepts)

   {Client receives the moviesList and sends its own movieInfo.}/

4) User clicks “play a video” from the list of available videos. Client module checks **movieInfo** and creates **neededSegments**

   (**neededSegments** = all Segments – **movieInfo**)

   If (first segment of video is found)

   {Player plays the first segment while the rest is downloading from server by populating **neededSegments** and sending to server by another thread.}

   Else {Client populates the **neededSegments** and sends to server}.

5) On the server side, the available segments are checked on all the clients and the availableSegments is populated and send back to the client.
6) The Client sends requests to get each segment one by one, by populating and sending the SegmentData.
7) The Server populates SegmentData with bytes of the video segment and sends them back to the client.
8) After receiving the first segment on the client, another thread starts playing the video file.

The following flowchart shows how the TCP module works.

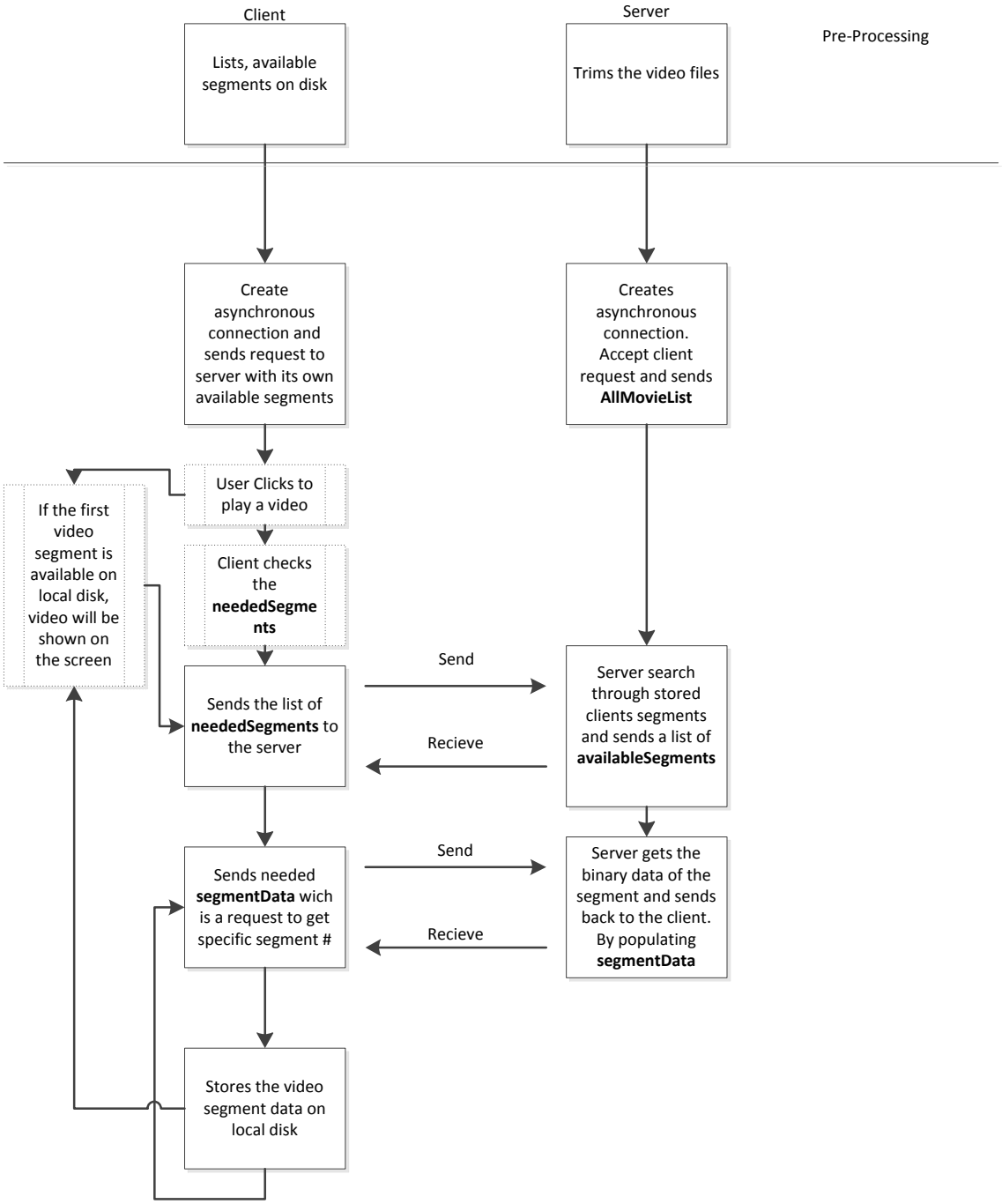

Figure 13: TCP module

3.10 UDP module

The role of the UDP module is to work like a server and a client module at the same time. This module is capable of receiving a request for a video segment and sending the corresponding segment to the requested peer like a server. The module's other capability is that it can receive one segment of a video and store it on secondary memory, in addition to playing the video like a client.

By creating the UDP module, all segments that are available in other peers can be transferred over the UDP connection with higher speed.

Below, some properties which should be satisfied by a UDP module are listed:

1) By creating a UDP module, each client works like a server for other peers
2) Each client doesn't have any information about the other peer's available segments, unless server gives the information to client.
3) Clients cannot request for any segment directly from other peers.

The conditions listed above are needed to get a better performance from the system. These conditions help the system to have less traffic on the network in such a way that, the server can decide about the idle peers and let the client request a segment from idle peers.

The following steps explain how the system works when other peers have some video segments.

1) When a video request sends to the server from a client, the server checks its own available clients (clients that are currently connected to the server) buffer.
2) The Server checks which peer has segments.
3) The server sorts peers in descending order according to their activity in the system (a peer with less activity is more preferable to share its own segments to other peers).
4) The server populates the preferred clients in a list of **availableSegments** and creates the packet and sends it to client.
5) In the client machine which requested the video a packet will be received which contains all peer's UDP IPs with the information about what segment each peer has.
6) In this level, the task of the server is completed and no other communication is needed between the client and the server over TCP.
7) Now, the client requests a needed segment data from the **availbaleSegments** packet from a peer, since it knows the peer IP's.
8) The peer is chosen as a seed node gets the segment data from hard disk and populates the **segmentData** class of the packet and sends it back to the client. (In UDP connection, we have more limitation on the size limit of each packet. This is illustrated in the "Limitation of UDP connection" section).
9) In case a peer has a problem (crashing, switching off, etc...); the client will automatically request the segment from the server over the TCP module.

The following flowchart shows a UDP connection, when working as a client, to request a segment from another peer.

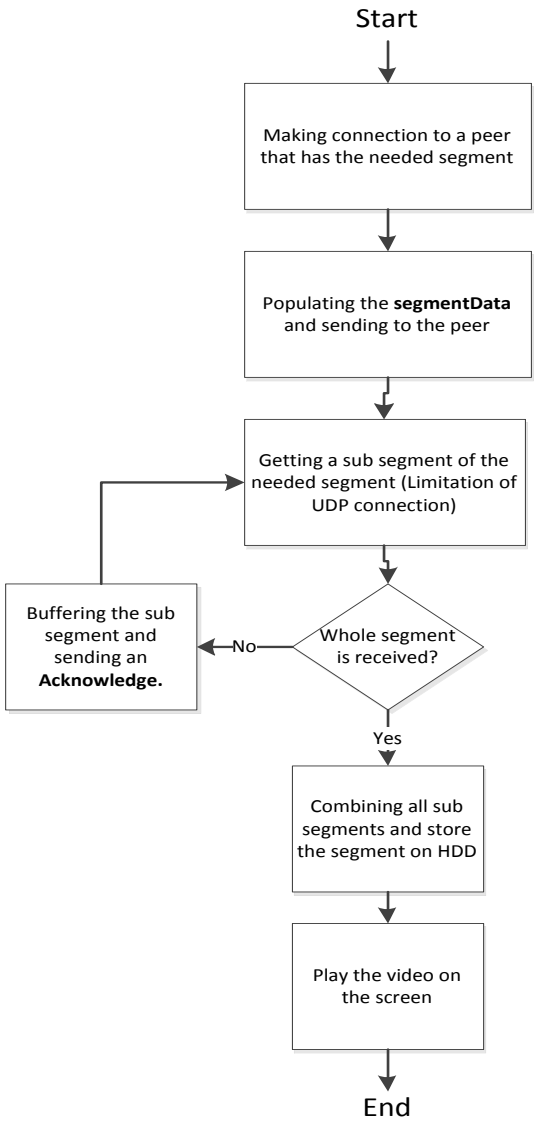

Figure 14: UDP connection (requesting a segment)

The following flowchart shows a UDP connection when working as a server to respond a requested a segment from another peer.

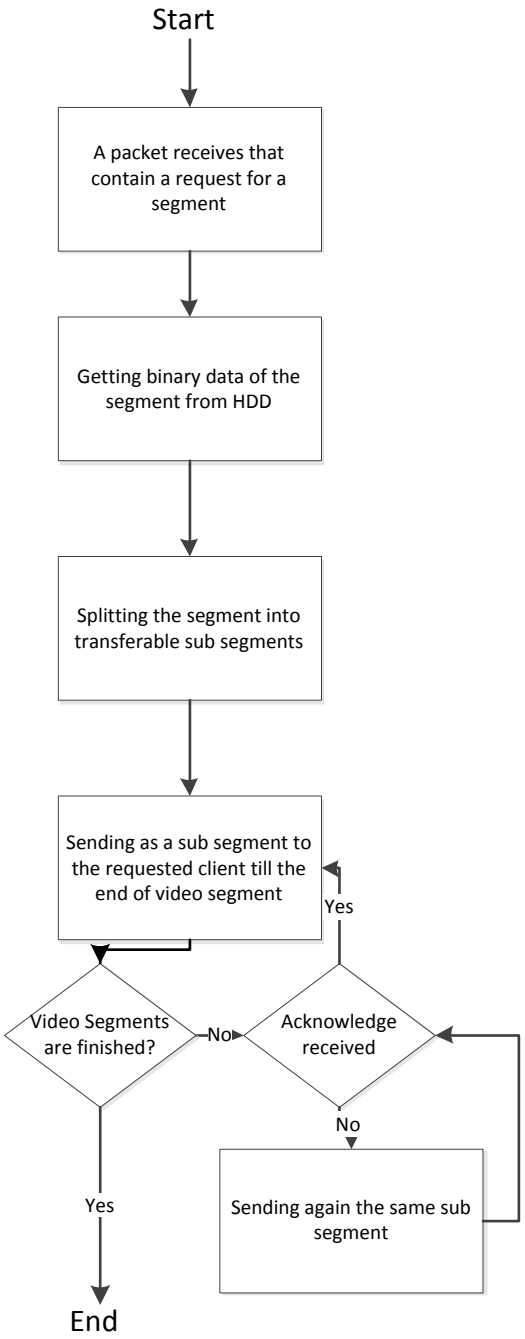

Figure 15. UDP Connection (Responding a Request)

3.11 Limitation on UDP connection

In implementation of the software we have found out that, UDP has a size limit for transferring a packet. This limit according to [11] is between 0 – 65535 bytes. In our experiment we chose a 65500 bytes for each packet to be transferred through the UDP connection. Since we have segmented our video files to 512 KB, a re – segmenting is required. The UDP size limit is very for video file. For example, a video file with size of 10 MB will have approximately 160 segments. To overcome this problem, we created a function that automatically gets the segment number requested from the other peer, reads the binary file from the local hard disk, and re – segments the file into 65500 bytes. Then it sends all packets one by one. On the other hand, as long as a UDP split segment (65500 bytes) is not readable as our system packet, the client continues sending "UDPacknowledge" for each received UDP's segments until it gets "finished", as the last packet "ProcessStep**"** and understanding that no more packets is needed for the segment. (For example, a 512 KB packet will be split to (8 * 65500) and (1*288) bytes. For each successive receive, the client sends a "UDPacknowlege" and waits for the next split segment).

In requested client side, when all packets are received they are buffered into one buffer (this buffer has a readable packet schema) and are checked to figure out the information handled within the packet.

3.12 TCP and UDP modules

This system is completed by combining the TCP and the UDP modules. By combining these, the system is capable of playing video files either getting video segments from the server or from other peers.

The following pseudo – code shows how the system works with both modules (TCP and UDP):

1. After receiving a list of videos on the client side, the user clicks to watch a video (a function checks the nrequired segments of the video and populate the respective part of the packet).
2. Server handles the request of the client and checks where the segments are. Then it sends a list that contains the IP address of the peers that has the segments to the requesting client.
3. The client attempt to get the segments form the indicated peers whose IP's are listed in the packet. (In case, a peer which is listed as a supplier of a segment has crashed, after a period of timeout, the client will request it from the server).
4. Segments are buffered in the client machine, and the video player plays the file until the last segment. During the playing if the rate of play is higher that the downloading rate, the player will be automatically paused until the missing segment is downloaded.

The following flowchart shows how the whole system is working with both modules.

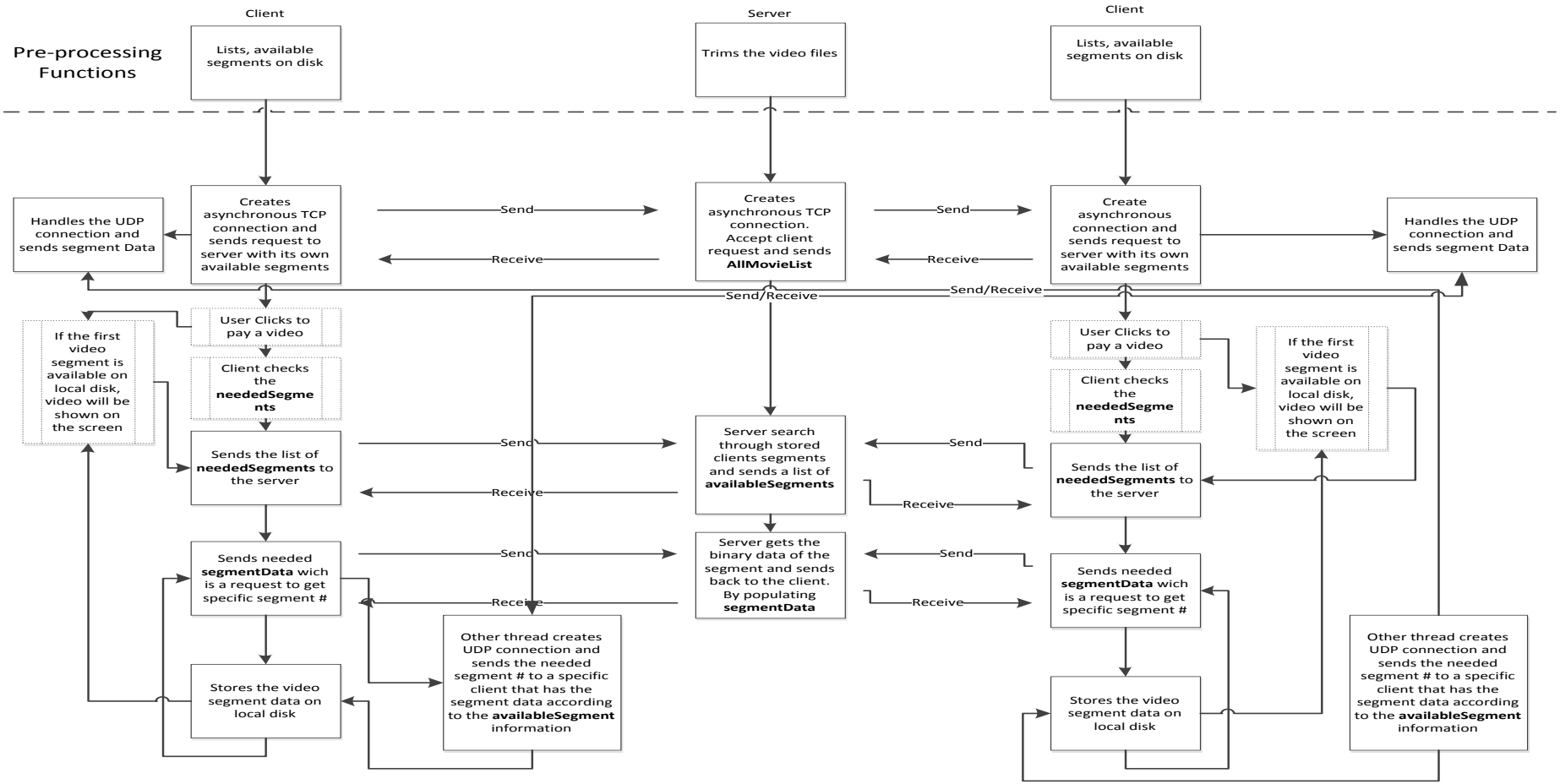

Figure 16. The TCP and The UDP modules.

Chapter 4

EXPERIMENTAL STUDIES

This software was implemented using the VLC ActiveX plugin [10] for playing videos on ".Net" framework. The program is implemented in C# language in .Net framework. NetLimiter [12] is used to limit the Internet bandwidth to a specific speed for the TCP module.

The Internet bandwidth is limited to 1Mbps, 2 Mbps and 3 Mbps for the TCP module testing and the LAN bandwidth is 10 Mbps for testing the UDP module. The hardware configuration of the server located on the Internet is: CPU- Pentium(R) Dual – Core 2.6 GHz /Memory 4 GB / HardDisk 160G.

First client hardware configuration is: CPU- Pentium(R) D 3.0 GHz /Memory 1 GB / HardDisk 80G.

Second client hardware configuration is: CPU- Core i5-750 2.67 GHz /Memory 3 GB / HardDisk 80G.

The whole system architecture is shown in the following figure

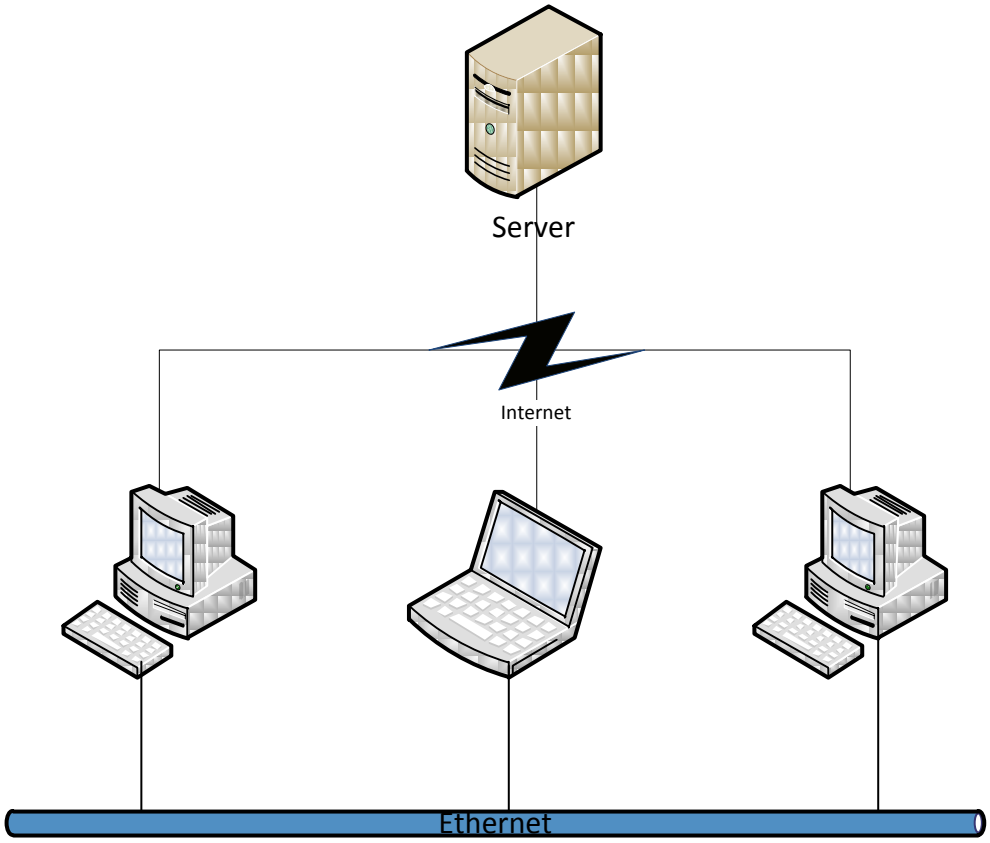

Figure 17: Implemented LCBBS architecture

Each segment of a video is trimmed to 512 KB units and stored on the server. The size and quality of the selected sample video files are illustrated in the following table:

Table 2: Video Samples

Item	**File Size (Byte)**	**Video Quality**
1	3,940,191	240p (320x240)
2	5,623,906	240p (320x240)
3	6,078,237	360p (640x360)
4	72,467,347	360p (640 ×360)
5	91,266,485	480p (640 ×480)

In our experiment, we chose different sizes video frame, 240p (320x240), 360p (640x360) and 480p (640 ×480) in constructing sample video files to test our software.

In order to have a smoother play, a metadata reader is implemented to get the 'data bit rate' of each video. By taking into consideration the 'data bit rate', the system adds a variable startup latency, to be sure that large enough video time is captured. Before startup according to the main LCBBS idea, users prefer to waiting to start at the beginning, instead of having more stop time while watching a video [1]. After some seconds of loading time, the client crates a video file on the local client machine and segment's data are appended to it.

The following table shows the duration of video in seconds and the data bit rate of sample videos. The first video file is the item 2 in table4 that is a “flv” video file with the size of “5,623,906” byte, second one is the item 3 that is a “flv” video file with the size of “6,078,237” and the third one is the item 5 that is a “mp4” video file with the size of “91,266,485”.

Table 3: Dependency of data bit rate and length of video

Video Quality of video samples	**Approximate data Bit Rate (1 sec) of video**	**Duration of video for each 512 KB packet**
240p (320x240)	25.25 KB/sec	20.27 Sec
360p (640x360)	83.75 KB/sec	6.1 Sec
480p (640 ×480)	120.5 KB/sec	4.2 Sec

This table shows the dependency between the data bit rate and the length of a video clip. A 512 KB of a video with quality of 240p has 20.27 sec length of the video. It has 6.1 sec of video length with quality of 360p, and 4.2 sec of video length with 480p quality. The above data are obtained from the meta data of the video files.

By applying the variable startup latency, a 480p (highest video quality in our experiment) video file can be played smoothly without any stop during the play time. (For example, for a video with quality of 480p and stopping the player for 2sec the first 4.2 sec will be reached to the client, so the rest of the video can be downloaded while another thread is playing the file.)

4.1 TCP connection speed test

The following table shows some parameters that are tested with the TCP module of software. This table gives information on the video clips that are transferred over a TCP connection between client and server with different quality and file sizes. The result obtained for each video clip is compared with the theoretical calculation. Our methodology is independent of video file formats; however the FLV and Mp4 files are tested in our system.

Table 4: Results of the TCP module.

Item	File Size (Byte)	Video Quality	Data Bit Rate (single frame)	File Format	Time with 1 Mbps Internet tested by software (Seconds)	Theoretically time needed with 1 Mbps Internet	Time with 2 Mbps Internet tested by software (Seconds)	Theoretically time needed with 2 Mbps Internet	Time with 3 Mbps Internet tested by software (Seconds)	Theoretically time needed with 3 Mbps Internet
1	3,940,191	240p (320x240)	25.25 KB/sec	FLV	32.10	30.06	15.84	15.03	11.08	10.02
2	5,623,906	240p (320x240)	42.75 KB/sec	FLV	43.84	42.91	22.03	21.45	14.44	14.31
3	6,078,237	360p (640x360)	83.75 KB/sec	FLV	47.75	46.37	23.43	23.18	16.03	15.45
4	72,467,347	360p (640 ×360)	71.25 KB/sec	MP4	553.08	552.88	279.31	276.44	185.66	184.29
5	91,266,485	480p (640 ×480)	120.5 KB/sec	MP4	670.23	696.30	350.14	348.15	233.60	232.10

From table results, we can conclude that the system works efficiently in sending and receiving segments. By comparing it to the theoretical result, we see that there are some small amounts of differences.

In the following figure, results of item 2, 3 and 5 in table 4 are compared.

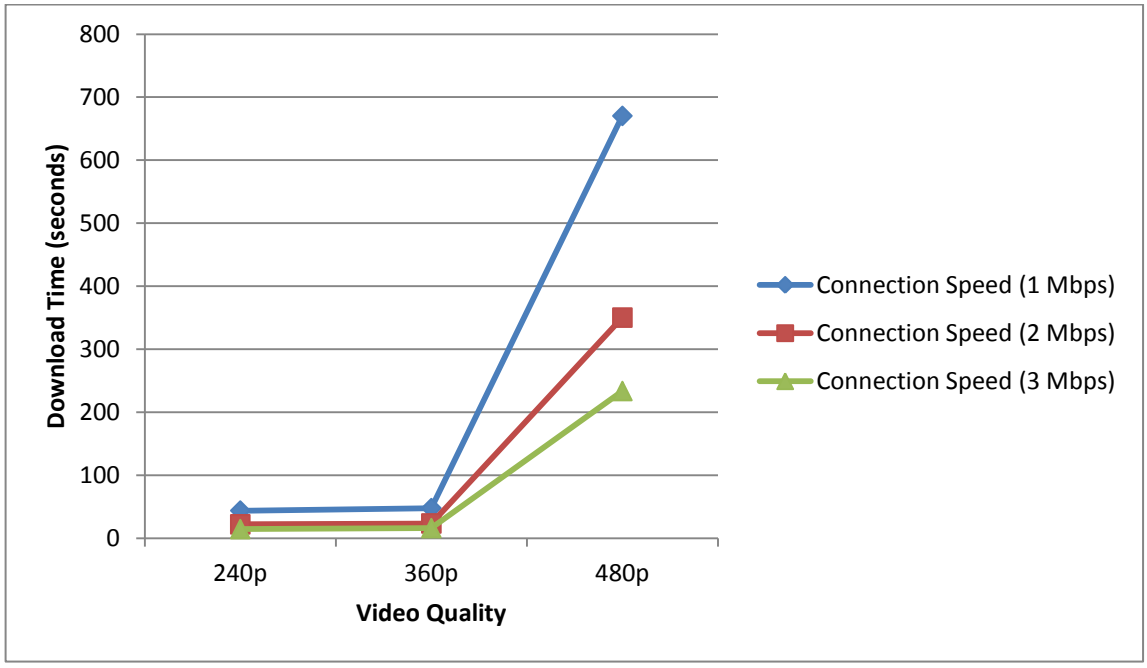

Figure 18. Comparison of the tested TCP modules.

In Figure 18, the horizontal axis plots the video quality according to the size of the video file and the vertical axis plots the time in seconds. This figure shows that, when the Internet speed is 1 Mbps, larger files take longer time to reach the other side, this difference is much smaller by a faster (3 Mbps) Internet connection speed.

Figure 18 also shows that, as the video quality increases, the time differences becoming larger.

4.2UDP connection speed test

The following table shows some statistics that are tested using the UDP module of software.

Table 5: Results of the UDP module.

Item	File Size (Byte)	Video Quality	Data Bit Rate (single frame)	File Format	Approximate time needed for each packet with size 512 KB Bytes	Time needed for all packets (for all video segments)
1	3,940,191	240p (320x240)	25.25 KB/sec	FLV	461 ms	3.22 Sec
2	5,623,906	240p (320x240)	42.75 KB/sec	FLV	461 ms	4.93 Sec
3	6,078,237	360p (640x360)	83.75 KB/sec	FLV	461 ms	5.30 Sec
4	72,467,347	360p (640×360)	71.25 KB/sec	MP4	461 ms	64.07 Sec
5	91,266,485	480p (640×480)	120.5 KB/sec	MP4	461 ms	80.67 Sec

The following figure shows the comparison of items 2, 3 and 5 in table5.

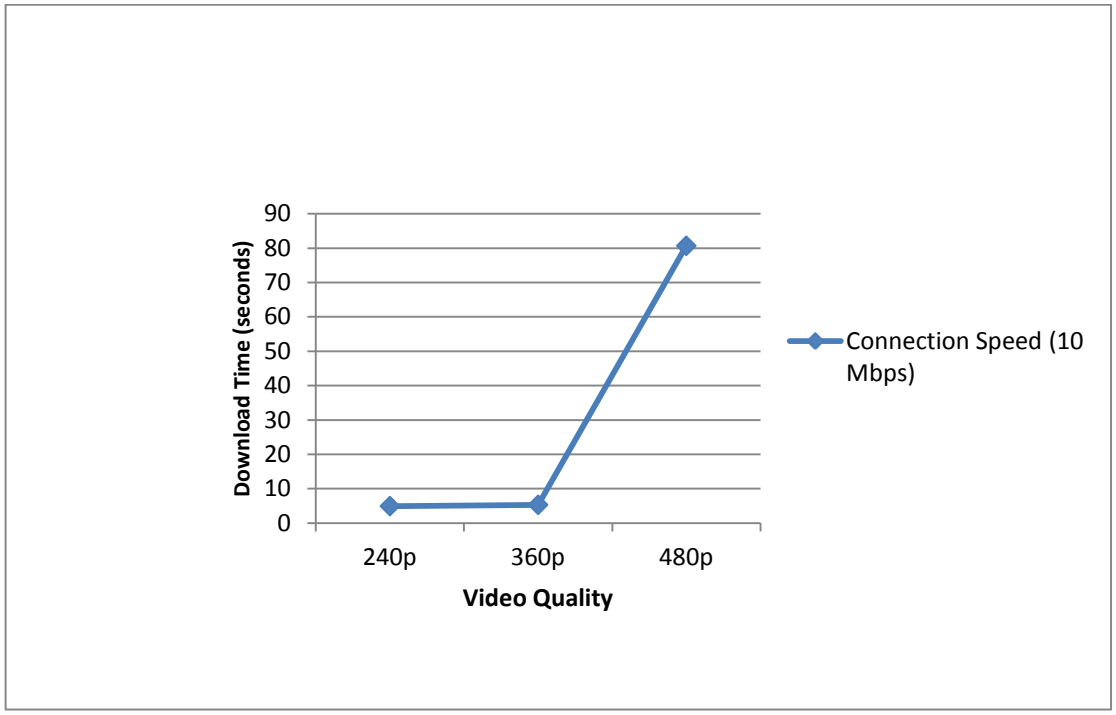

Figure 19. Comparison of the tested UDP modules.

The figure 19 shows the approximate needed time for videos with different quality to reach the destination. (For example, item 2 in table 5 reaches the destination in less than 10 seconds while the time for item 5 is around 80 seconds.)

4.3 UDP and TCP comparison

The following figure shows how much speed of UDP (LAN connection) can help the system.

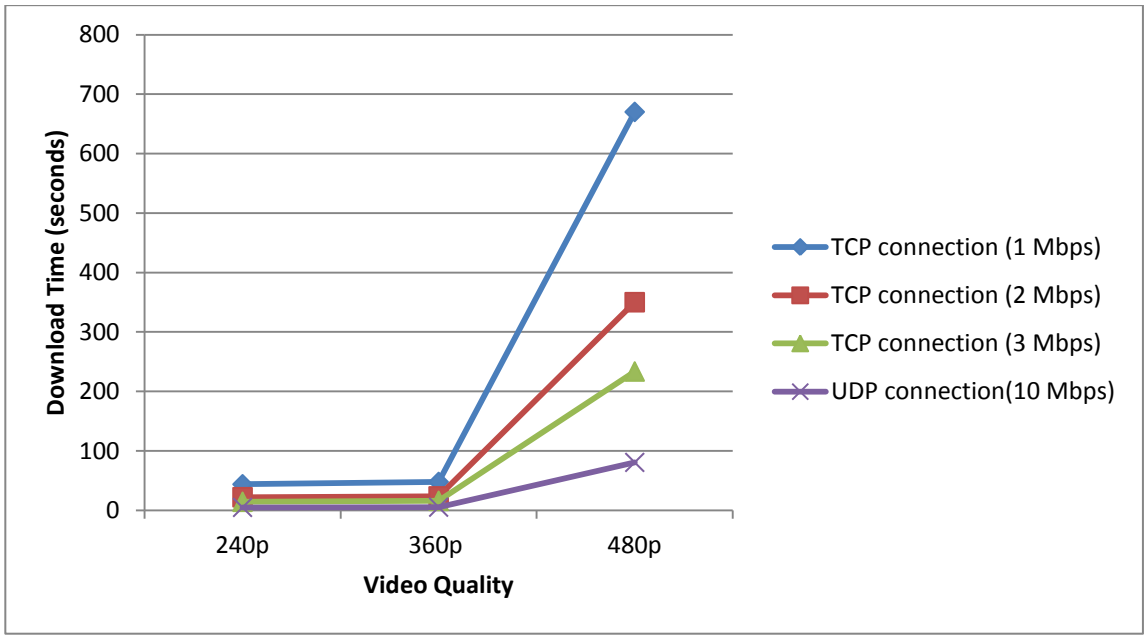

Figure 20.TCP and UDP speed comparison.

It is supposed that all segments of a video file items 2, 3 and 5 are available on other peer machines, so with a 10 Mbps LAN speed, the whole video file with the highest tested quality (480p) reaches to requested client machine in less than 100 seconds compared to a TCP connection with speed of 3 Mbps with more than 200 seconds.

We tested our software with two clients and a server and obtained some experimental results. The results showed that the system works well with variable quality video files. Since peers share their video segments in their idle time (clients that are not transferring data with the server are supposed as idle), when the number of clients goes larger, the

system would work with better performance. Thus, a test with two clients is considered to be sufficient to measure the worst – case performance.

4.4 Comparison of experimental work with other studies

The segment size in the LCBBS simulation is chosen as 128 KB, video data bit rate is chosen as 350Kbps (43.75 KB/sec as constant) and internet speed as 120 Kbps. In our system segments size are 512 KB because the internet speed 2012 which is faster than the time 2007 when the LCBBS module is simulated. we chose the Internet speed to be 1 Mbps, 2 Mbps and 3 Mbps.

The following table shows the different parameters that are used in this study, LCBBS simulation [13] and P2P – VoD – FSS [6] simulation.

Table 6: Implementaion and simulation parameters comparison

Parameters	**LCBBS Implementation**	**LCBBS Simulation**	**P2P – VOD – FSS Simulation**
Server Specification	Segmenting videos based on a fixed size using its own video server	Segmenting videos based on a fixed size using a streaming server	Segmenting videos based on a fixed size using a streaming server
System Architecture	A server and clients	A server and clients	A server, a strong node and clients
File Segments	512 KB	128 KB	200 KB
Internet Speed	1 Mbps, 2Mbps, 3Mbps	120 Kbps	Variable (the speed is not mentioned)
LAN Speed	10 Mbps	10 Mbps	100 Mbps
Video Files	Variable bitrate tested	Fixed bitrate tested	Fixed bitrate tested

Chapter 5

CONCLUSION

By increasing the bandwidth offered by Internet Service Providers (ISPs), the Video on Demand (VoD) services becoming popular. However giving service to large number of users in a concurrent access requires huge link capacity. Our implementation is based on the Video Locality Based Buffering Mechanism (LCBBS) improves VoD systems.

Based from the result of the tested software, a peer – to – peer video on demand system in which the peers help the system to distribute videos worked better than traditional video on demand systems. Our implemented software can be used as a standalone computer program to distribute video segments over the internet, in addition if the connected clients are located in a Local Area Network (LAN), the video files can be buffered and distributed over a higher speed connection to other clients. This implementation achieved a quicker start-up and a faster buffering in client machines. The results showed that, videos played smoother and the stop and wait time is smaller than a common VoD system. The stop and wait time decreases when the number of available segments in other peers is increases. The comprised results in chapter 4 showed that peers improve the overall performance of the system.

Our software can distribute video files and play those that are supported by the VLC media player. Other capability of our software is that it can be used as a file server

located on the Internet to distribute files such as any kind of video files, audio files, document files and etc., to clients without any consideration of file formats.

As future work, a function can be added to the software to periodically check the available clients on the system in a specific timeout. This function can help the server to update the list of available segments on the connected clients and decrease the probability of a wrong addressed of a peer in the case that a crash might happen. Also the software is tested with two clients and a server for worst – case behavior. The number of clients can be increased to get more realistic results.

REFERENCES

[1] C.Huang, J.Li and K.W. Ross, "Peer-assisted VoD: Making internet video distribution cheap," *In Proc. of IPTPS,* Feb, 2007.

[2] Hasan Sarper and Isik Aybay, "A video Locality Based Buffering Mechanism for VoD Systems," *IEEE Transaction on Cunsumer Electronics,* vol. 53, no. 4, pp. 1521-1528, 2007.

[3] Yuabo Zhang, Hui Wang, Pei Li, Zhihong Jiang and Chao Gao, "How can peers assist each other in large-sclae P2P-VoD systems," *International Conference on Multimedia Information Networking and Security,* pp. 198-202, 2010.

[4] Guan, Yifeng He and Ling, "Improving the streaming capacity in P2P VoD systems with helpers," *IEEE International Conference,* pp. 790-793, 2009.

[5] Lin, Kai Wang and Chuang, "Insight into the P2P-VoD system: performance modeling and analysis," *IEEE Multimedia,* pp. 1 - 6 , 2009.

[6] ZhiHui Lu, ShiYoung Zhang, Jie Wu, WeiMing Fu and YiPing Zhong, "Design and implementation of a novel P2P- based VOD system using media file segments selecting algorithm," *IEEE International Conference,* pp. 599 - 604, 2007.

[7] "Adobe RTMP," [Online]. Available: http://www.adobe.com/devnet/rtmp.html. [Accessed 8 October 2011].

[8] "Adobe," [Online]. Available: http://www.adobe.com/. [Accessed 8 October 2011].

[9] "Windows Media Player," [Online]. Available: http://windows.microsoft.com/en-US/windows/products/windows-media. [Accessed 2 Novemebr 2011].

[10] V. M. Player, "VideoLan," [Online]. Available: http://www.videolan.org/vlc/features.html. [Accessed 11 November 2011].

[11] B. Mitchell, "CompNetworking," [Online]. Available: http://compnetworking.about.com/od/networkprotocolsip/g/udp-user-datagram-protocol.htm. [Accessed 5 November 2011].

[12] "NetLimiter," [Online]. Available: http://www.netlimiter.com/. [Accessed 25 November 2011].

[13] H.Sarper and I.Aybay, "Improving VoD performance with LAN client back-end buffering," *IEEE Multimedia*, vol. 14, no. 1, pp. 48-60, January 2007.