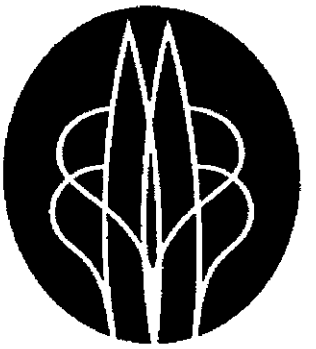


UNIVERSITI
TEKNOLOGI
PETRONAS

Peer to Peer Video Streaming Application

By:
Muhammad Ilham Faez b. Shafri

3758

Information Technology

2008

Abstract

This final year project is entitled Peer-to-peer video streaming. Peer-to-peer video streaming is an alternative method of video streaming besides the client-server video streaming. The program will allow the client to share the network resources such as bandwidth in order to stream the video. Each user, while downloading, is also uploading, thus contributing to the overall available bandwidth. The video quality of the channels typically depends on how many users are watching; the video quality is better if there are more users. This project will use a multi sender method in a peer-to-peer network environment. We are going to use a multicast method on the top of an arbitrary multi-sender method so that all requesting peers receive almost the same expected bit-rate. The program will be done using Java environment and its algorithms.

The principle used in doing the project is sharing the computer resources and the idea increasing the scalability according to the number of receivers.

Table of Contents

1. Introduction

1.1 Background Study

Increasing penetration of high-speed Internet access (e.g., ADSL) among the users of peer-to-peer (P2P) networks enables deployment of real-time multimedia delivery schemes over them, in addition to file sharing – their traditional application. The differences between peer-to-peer video streaming with traditional client server application is that peer-to-peer network are ordinary nodes with limited bandwidth.

By employing a multi-sender method, the limited bandwidths of the sender peers do not impose a serious restriction on streaming quality. In fact, MSMC can be integrated into any existing multi-sender scheme to provide a scalable multicast solution. Also, by considering the availability of senders, the quality of the streamed media is improved.[1]

Another advantage of being receiver-driven is that the multicast trees are made in a distributed manner. Each receiver makes its multicast tree itself. Also in the proposed method, the joining operation is managed by previously joined receivers except for the first receiver which is managed by the senders. By being a multicast scheme, a large number of receivers can receive multimedia from a limited number of senders without stressing the P2P substrate or the senders.

From a distribution point of view, Peer-to-Peer technologies facilitate better and more targeted distribution because sharing can take place within communities having common interests, communities that would typically already exist to take advantage of the content. When used as a method of distributing video using files (downloads), it can greatly enhance distribution because files are downloaded more quickly

Time-sensitive applications, such as streaming media, gain popularity and real-time data is expected to compose a considerable portion of the overall data traffic traversing the Internet. These applications generally prefer timeliness to reliability. Real-time video streaming, in particular, calls for strict requirements on end-to-end delay and delay variation. Furthermore, reliability parameters, such as packet loss and bit errors, usually compose an impairment factor, since they cause perceptible degradation on video quality .Unlike bulk-data transfers, video streaming seeks to achieve smooth playback quality rather than simply transmit at the highest attainable bandwidth.

Such stringent requirements necessitate explicit management techniques in order to preserve the fundamental Quality of Service (QoS) guarantees or video traffic. In this context, Internet Engineering Task Force (IETF) attempted to facilitate true end-to-end QoS on IP networks by defining Integrated (IntServ) and Differentiated Services (DiffServ) models. IntServ follows the signaled- QoS model, where the end-hosts signal their QoS need to the network, while DiffServ works on the provisioned QoS model, where network elements are setup to service multiple classes of traffic with varying QoS requirements. However, both models are associated with high implementation costs and limited applicability; hence, they have not yet received wide appeal from the majority of users. Essentially, most end- users still rely on the best-effort services of the Internet which strives to meet the high demands of the merging multimedia applications.

1.2 Problem Statement

Time dependent requirements are generally express as Quality of Service (QoS) requirements. These requirements describe what is needed from the underlying distributed system and network to ensure that, for example, the temporary relationship in a stream can be presenrved. QoS concern for the data stream mainly concern **timeliness**, **volume** and **reliability.[3]**

Over 40% extra data overhead compared to unicasting. To bypass the QoS issue, P2P networks have multiple peers send multiple traffic to other peers, introducing extra data overhead for retransmits communication and redundancy. Dutch ISP's have calculated the traffic needed to send a P2P stream to a number of users, and measured 40% additional traffic usage compared to unicasting an RTSP stream to a similar audience. Alternately, Multicasting (although not a widespread internet technology) is even more efficient than unicasting, allowing one stream to feed a virtually unlimited number of viewers. Synchronization and buffering will also produce a challenge in developing the system.

1.3 Objective and Scope of Study

The primary objective of this project is to create a java coded program that could steam video using peer-to-peer technology. Hence the project target is to make the file distribution faster and guarantee a reliable stream.

In order to achieve those goals, knowledge about network topology and data transfer through out the network is necessary (link bandwidths or physical proximities of the neighbors of each node). This approach decreases the initial streaming delay.

We also need to be able to design an algorithm using Java in order to fulfill the requirement of the project.

We also need to understand the basic architecture of the Peer-to-peer networking and how the data is transferred in the networking topology. Through the understanding of the Peer-to-peer network then we would be able to manage the file transfer throughout the system to all the users and the server. With further understanding also we would be able to optimize the data transfer of the system and also minimize the resource usage of the system so that it would run properly.

2.0 LITERATURE REVIEW AND THEORY

The main objective of creating a peer-to-peer video streaming is that to demonstrate that Peer-to-Peer downloading can save substantial amounts of bandwidth costs because many peers are likely to be within a single ISP (Internet Service Provider) and hence "interconnection" costs are reduced.

If a receiver R requests a certain multimedia, a set of candidate senders (determined by a location protocol) having the desired media, signal their readiness to transmit data to R. The receiver can simply connect directly to the senders and start downloading. However, as the download bandwidth of a typical node is considerably larger than its upload bandwidth (e.g., for ADSL the ratio is 8 to 1), the simple scheme of direct connection leads to selfish usage of the network resources: no other nodes can use the senders from which R is receiving the media. Also R cannot provide the media to any other receiver at the same bit-rate it is receiving the media.[1]

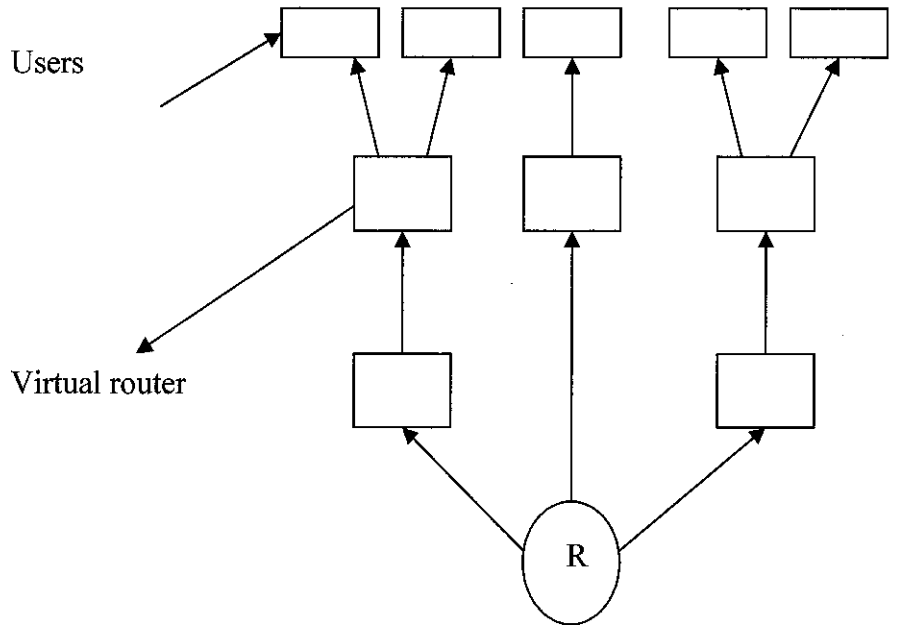


Fig. 1. A sample multi-sender structure. Virtual routers (a.k.a. "forwarders") route data to the receiver over the P2P substrate.[1]

The proposed multicast method is illustrated in Fig. 1. Nodes users are active senders (i.e., have the media and are transmitting data to a receiver) to R. Nodes virtual which are routing the stream in this topology, have a partial content of the streamed media. Our idea is to use these routers as temporary senders to forward copies of packets destined for R to another requesting node such as R. Intuitively; one can observe that using this approach the EBR of R can be increased without pressuring the bandwidths of the primary senders. Moreover, by addition of each receiver to this topology, a number of new temporary senders appear that can serve even more new receivers. Thus, the proposed method is scalable with the number of receivers.[1]

2.1 Protocol Used

2.1.1 RTSP (real time streaming protocol)

The real time streaming protocol (RTSP) is developed by the IETF and is a protocol used for streaming media system which allow client to remotely control the streaming media, issuing commands such as "play" or "pause" and also allowing time-based access file on a host computer.

Most of the RTSP host uses the standard-based RTP (real time transport protocol) as the transport protocol for the actual audio/video data acting somewhat as a metadata channel. The set of standards that include RTSP and RTP are unfortunately not sufficiently complete or specific to ensure the interoperability and each client/server implementation tends to be a little different.

2.1.2 RTP (real-time transport protocol)

The real-time transport protocol (RTP) defines a standardized packet format for delivering audio and video through the internet. RTP does not have a standard TCP or UDP port on which it communicates, the only standard that it obeys is that UDP communications are done via even port and the next higher odd port is used for RTP control Protocol (RTCP) communications. Although there are no standard are assigned, RTP is generally configured using ports 16384-32767. The fact that RTP uses a dynamic port range makes it difficult for it to traverse firewalls.

The RPT was originally designed as a multicast protocol, but has since been applied in many unicast applications. It is frequently used in streaming media systems as well as video conferencing and push to talk system and the latest is the development of VoIP.

The services provided by the RTP protocol include:

- Payload-type identification which indicate what kind of content is being carried by the system
- Sequence numbering which allow PDU sequence numbering
- Time stamping which allow synchronization and jitter the calculations.

The position of RTP in the protocol stack is somewhat strange. It was decided to put RTP in user space and have it (normally) run over UDP. It operates as follows. The multimedia application consists of multiple audio, video, text, and possibly other streams. These are fed into the RTP library, which is in user space along with the application. This library then multiplexes the streams and encodes them in RTP packets, which it then stuffs into a socket. At the other end of the socket (in the operating system kernel), UDP packets are generated and embedded in IP packets. If the computer is on an Ethernet, the IP packets are then put in Ethernet frames for transmission.As a consequence of this design, it is a little hard to say which layer RTP is in. Since it runs in user space and is linked to the application program, it certainly looks like an application protocol. On the other hand, it is a generic, application-independent protocol that just provides transport facilities, so it also looks like a transport protocol. Probably the best description is that it is a transport protocol that is implemented in the application layer.

2.2System architecture

2.2.1 Peer-to-peer (P2P)

A **peer-to-peer** (or "**P2P**") computer network exploits diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application. Peer-to-peer networks are typically used for connecting nodes via largely *ad hoc* connections. Such networks are useful for many purposes. Sharing content files (see file sharing) containing audio, video, data or anything in digital format is very common, and realtime data, such as telephony traffic, is also passed using P2P technology.

A pure peer-to-peer network does not have the notion of clients or servers, but only equal ***peer*** nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. A typical example for a non peer-to-peer file transfer is an FTP server where the client and server programs are quite distinct, and the clients initiate the download/uploads and the servers react to and satisfy these requests.

An important goal in peer-to-peer networks is that all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. This is not true of a client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users.

The distributed nature of peer-to-peer networks also increases robustness in case of failures by replicating data over multiple peers, and -- in pure P2P systems -- by enabling peers to find the data without relying on a centralized index server. In the latter case, there is no single point of failure in the system.

When the term peer-to-peer was used to describe the Napster network, it implied that the peer protocol was important, but, in reality, the great achievement of Napster was the empowerment of the peers (i.e., the fringes of the network) in association with a central index, which made it fast and efficient to locate available content. The peer protocol was just a common way to achieve this.

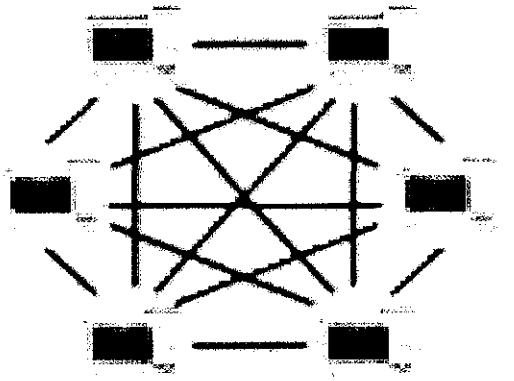


Figure 2.2.1a the network architecture of a peer to peer environment

2.2.2 Client Server Architecture

The most commonly used architecture used in the networking environment is the client server architecture. It consist of network connected computers and server. Computing architecture which separates a client from a server, and is almost always implemented over a computer network. Each client or server connected to a network can also be referred to as a node. The most basic type of client-server architecture employs only two types of nodes: clients and servers. This type of architecture is sometimes referred to as *two-tier*. It allows devices to share files and resources.

Each instance of the client software can send data requests to one or more connected *servers*. In turn, the servers can accept these requests, process them, and return the requested information to the client. Although this concept can be applied for a variety of reasons to many different kinds of applications, the architecture remains fundamentally the same.

These days, clients are most often web browsers, although that has not always been the case. Servers typically include web servers, database servers and mail servers. Online gaming is usually client-server too. In the specific case of MMORPG, the servers are typically operated by the company selling the game; for other games one of the players will act as the host by setting his game in server mode.

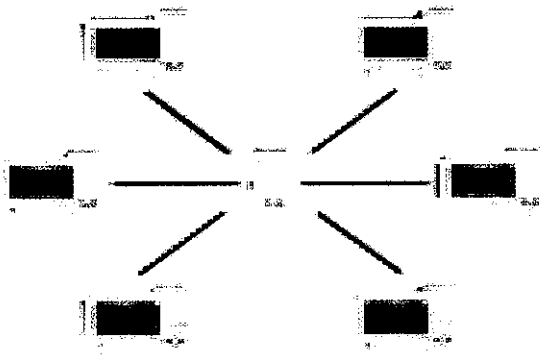


Figure 2.2.2 the network architecture of a client server environment

2.2.3 Comparison between Peer-to-peer architecture and client-server architecture

The main difference between peer-to-peer architecture and the client server architecture is that each computer in a peer-to-peer network environment acts as a client and a server at the same time. This means that each individual personal computer can host and share a file with other personal computer via network. Where else in a client server environment a server is hosting all the file sharing and storage in the network.

An important goal in peer-to-peer networks is that all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. This is not true of a client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users.

The distributed nature of peer-to-peer networks also increases robustness in case of failures by replicating data over multiple peers, and -- in pure P2P systems -- by enabling peers to find the data without relying on a centralized index server. In the latter case, there is no single point of failure in the system.

When the term peer-to-peer was used to describe the Napster network, it implied that the peer protocol was important, but, in reality, the great achievement of Napster was the empowerment of the peers (i.e., the fringes of the network) in association with a central index, which made it fast and efficient to locate available content. The peer protocol was just a common way to achieve this.

While the original Napster network was a P2P network the newest version of Napster has no connection to P2P networking at all. The modern day version of Napster is a subscription based service which allows you to download music files legally.

Where else in client server architecture in most cases, client-server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network. This creates an additional advantage to this architecture: greater ease of maintenance. For example, it is possible to replace, repair, upgrade, or even relocate a server while its clients remain both unaware and unaffected by that change. This independence from change is also referred to as *encapsulation.*

All the data are stored on the servers, which generally have far greater security controls than most clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data. Since data storage is centralized, updates to those data are far easier to administer than would be possible under a P2P paradigm. Under a P2P architecture, data updates may need to be distributed and applied to each "peer" in the network, which is both time-consuming and error-prone, as there can be thousands or even millions of peers.

3.0 METHODOLOGY

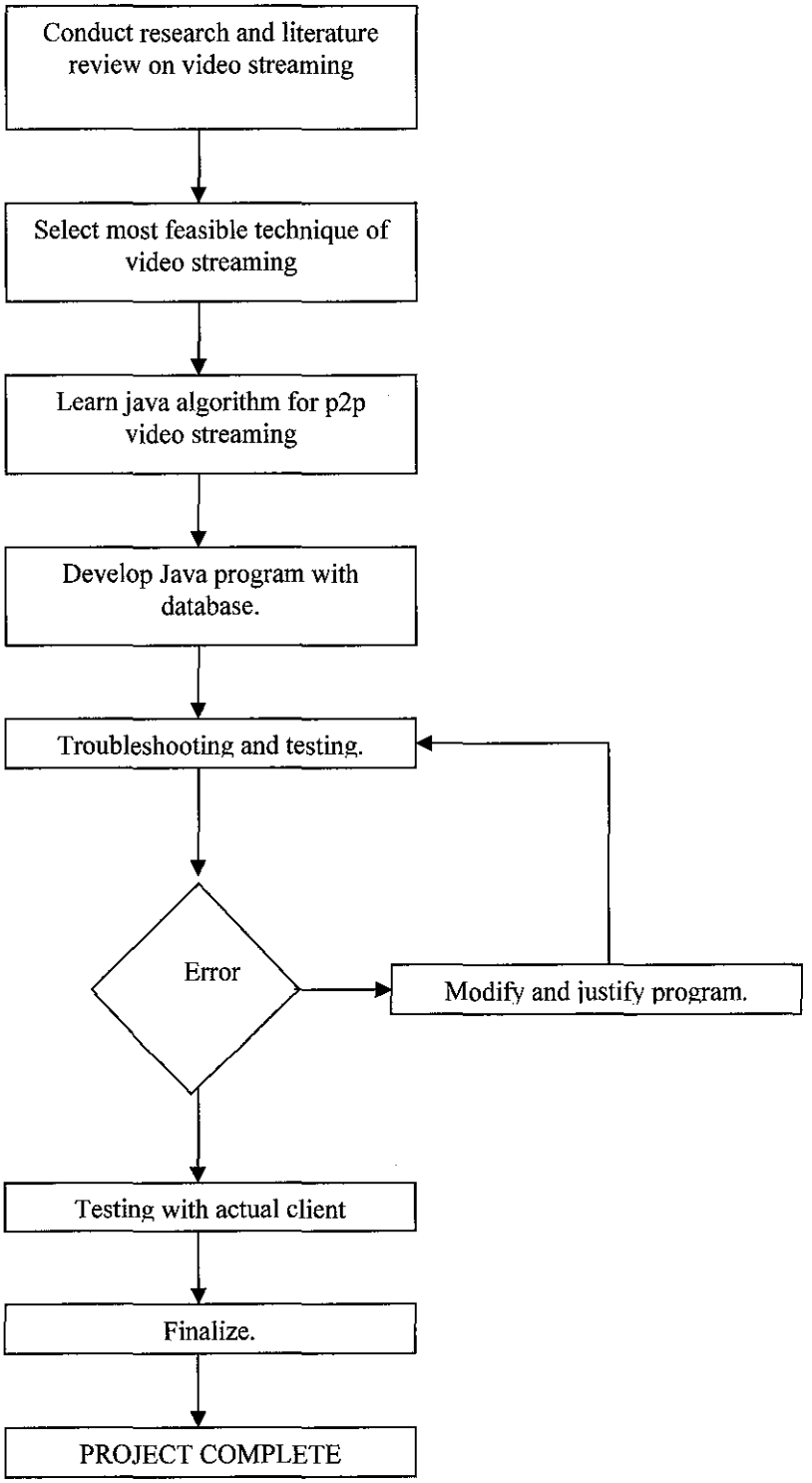
The project will start with intensive research and studies of techniques of video streaming. Then the most feasible and best method will be selected as basis for this project. The information is obtained by researching and reading of papers, journals, websites and books.

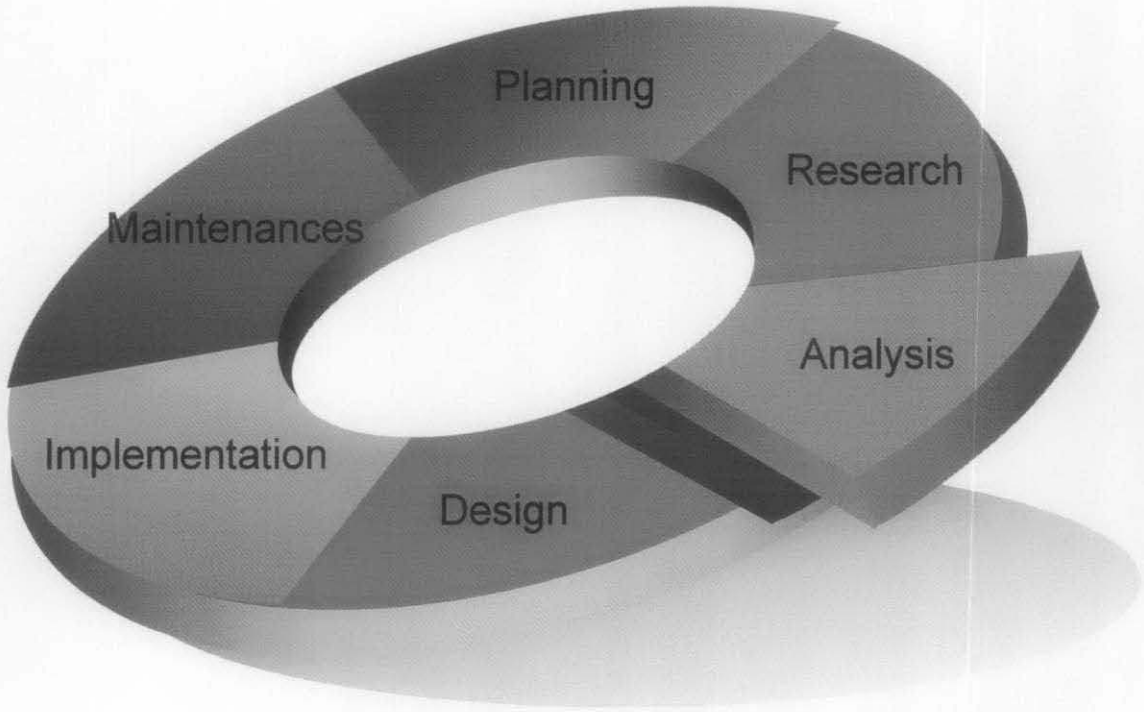
A thorough understanding and knowledge of the Java software commands is needed to develop the algorithm to be used for the selected method. The theories and functions will be constructed in Java environment and will be tested and enhanced should the need occur.

The design phase of the Java algorithm would require much learning and supervision. A complete understanding of the method of face recognition chosen is needed. Much troubleshooting would be done to connect the different commands of the Java into a working program.

After the development of the program, the interfacing of the the program is done. Further research is needed to add commands to the program so further enhancement regarding the file transfer and video stream will be applicable.

The figure below shows the project flowchart.





The figure shows the methodology used in designing the system

4.0 RESULTS AND DISCUSSION

4.1 Below is the Prototype of unicast streaming coding

```
package streaming.server.manager.transport.unicast;

import streaming.helper.error.ErrorLog;

import javax.media.NotRealizedError;
import javax.media.MediaLocator;
import javax.media.RealizeCompleteEvent;
import javax.media.ControllerListener;
import javax.media.ControllerEvent;
import javax.media.Format;
import javax.media.NoProcessorException;
import javax.media.Processor;
import javax.media.Manager;
import javax.media.ConfigureCompleteEvent;
import javax.media.EndOfMediaEvent;
import javax.media.protocol.PushBufferDataSource;
import javax.media.protocol.PushBufferStream;
import javax.media.protocol.ContentDescriptor;
import javax.media.protocol.DataSource;

import javax.media.control.FormatControl;
import javax.media.control.TrackControl;

import javax.media.format.AudioFormat;
import javax.media.format.VideoFormat;
import javax.media.rtp.RTPManager;
import javax.media.rtp.SessionAddress;
import javax.media.rtp.SendStreamListener;
import javax.media.rtp.SendStream;

import java.io.IOException;
import java.net.InetAddress;
import javax.media.rtp.event.NewSendStreamEvent;
import javax.media.rtp.event.StreamClosedEvent;
import javax.media.rtp.event.SendStreamEvent;
```

```
public final class Unicast_rtp implements ControllerListener, SendStreamListener
{
 private Processor processor;
 private String url;
 private DataSource ds = null;
 private SendStream mySendStream = null;
 private int prepare_track;

 private int local_rtp;
 private InetAddress destIP;
 private int dest_rtp;

 private RTPManager[] mgr;

 private boolean endofMedia = false;

 public Unicast_rtp(String file, InetAddress d_IP, int l_rtp, int d_rtp, int track)
 {
  url = file;
  local_rtp = l_rtp;
  destIP = d_IP;
  dest_rtp = d_rtp;
  prepare_track = track;
 }

 private void myEx(Exception ex, String f)
 {
  f += " :";
  f += ex.getMessage();
  new ErrorLog(f);
 }

 public boolean createMyProcessor()
 {
  final String e = "RTP_Stream createMyProcessor";

  try
  {
   processor = Manager.createProcessor(new MediaLocator(url));
   processor.addControllerListener(this);
```

```
    processor.configure();
   }
   catch(IOException ex)
   { myEx((Exception) ex, e);  return false; }
   catch(NoProcessorException ex)
   { myEx((Exception) ex, e);  return false; }
   return true;
  }

  public void controllerUpdate(ControllerEvent p0)
  {
   if(p0 instanceof ConfigureCompleteEvent)
   {
    Format format;
    boolean encodingOK = false;

    TrackControl track[] = processor.getTrackControls();
    ContentDescriptor cd = new ContentDescriptor(ContentDescriptor.RAW_RTP);
    processor.setContentDescriptor(cd);
    format = track[prepare_track].getFormat();

    if(format instanceof VideoFormat)
    {
     VideoFormat v = (VideoFormat)track[prepare_track].getFormat();
     encodingOK = setMyVideoFormat(v, track[prepare_track]);
    }
    if(format instanceof AudioFormat)
    {
     AudioFormat a = (AudioFormat)track[prepare_track].getFormat();
     encodingOK = setMyAudioFormat(a, track[prepare_track]);
    }

    if(encodingOK)
    {
     for(int i=0; i<track.length; i++)
     {
      if(i != prepare_track)
      { track[i].setEnabled(false); }
     }
     processor.realize();
    }
```

```
  }

  if(p0 instanceof RealizeCompleteEvent)
  {
   try
   {
    ds = processor.getDataOutput();
    createMyRTPManager();
   }
   catch(NotRealizedError ex)
   { myEx(null, ex.getMessage()); }
  }

  if(p0 instanceof EndOfMediaEvent)
  { closeMyStream();  endofMedia = true;  }
}

public void update(SendStreamEvent p0)
{
 if(p0 instanceof NewSendStreamEvent)
 { startMyStream();  }
 if(p0 instanceof StreamClosedEvent)
 { closeMyStream();  }
}

private boolean setMyVideoFormat(VideoFormat v, TrackControl track)
{
 boolean found = false;
 if(v.isSameEncoding(VideoFormat.MPEG))
 {
  ((FormatControl)track).setFormat(new VideoFormat(VideoFormat.MPEG_RTP));
  found = true;
 }
 if(v.isSameEncoding(VideoFormat.JPEG))
 {
  ((FormatControl)track).setFormat(new VideoFormat(VideoFormat.JPEG_RTP));
  found = true;
 }
 if(v.isSameEncoding(VideoFormat.MJPG))
 {
  ((FormatControl)track).setFormat(new VideoFormat(VideoFormat.JPEG_RTP));
```

```
    found = true;
   }
   track.setEnabled(found);
   return found;
  }

  private boolean setMyAudioFormat(AudioFormat a, TrackControl track)
  {
   boolean found = false;

   if(a.isSameEncoding(AudioFormat.MPEG))
   {
    ((FormatControl)track).setFormat(new AudioFormat(AudioFormat.MPEG_RTP));
    found = true;
   }
   if(a.isSameEncoding(AudioFormat.MPEGLAYER3))
   {
    ((FormatControl)track).setFormat(new AudioFormat(AudioFormat.MPEG_RTP));
    found = true;
   }
   if(a.isSameEncoding(AudioFormat.LINEAR))
   {
    ((FormatControl)track).setFormat(new AudioFormat(AudioFormat.DVI_RTP));
    found = true;
   }
   if(a.isSameEncoding(AudioFormat.ULAW))
   {
    ((FormatControl)track).setFormat(new AudioFormat(AudioFormat.ULAW_RTP));
    found = true;
   }

   track.setEnabled(found);

   return found;
  }


  private boolean createMyRTPManager()
  {
   PushBufferDataSource pbds = (PushBufferDataSource)ds;
```

```
   PushBufferStream pbss[] = pbds.getStreams();

   mgr = new RTPManager[pbss.length];

   for(int i=0; i<pbss.length; i++)
   {
    try
    {
     mgr[i] = RTPManager.newInstance();
     mgr[i].addSendStreamListener(this);
     SessionAddress localAddr = new SessionAddress( InetAddress.getLocalHost(),
local_rtp);

                                                                    SessionAddre
ss  destAddr = new SessionAddress( destIP, dest_rtp);
      mgr[i].initialize( localAddr);

                                                                    mgr[i].addTar
get( destAddr);
      mySendStream = mgr[i].createSendStream(ds, i);
     }
     catch (Exception  e)
     { myEx(e, "RTP_Stream createMyRTPManager"); return false; }
   }

   return true;
  }

  private boolean startMyStream()
  {
   try
   {
    mySendStream.start();
    processor.start();
   }
   catch(IOException ex)
   { myEx((Exception) ex, "RTP_Stream startMyStream");  return false; }
   return true;
  }

  private void closeMyStream()
```

```
{
 processor.close();
 processor.deallocate();
 mySendStream.close();
 for(int i=0; i<mgr.length; i++)
 { mgr[i].dispose(); }
}

public void startStreamAgain()
{
 try
 { mySendStream.start(); }
 catch(IOException ex)
 { myEx((Exception) ex, "RTP_Stream startStreamAgain"); }
}

public void pauseStream()
{
 try
 { mySendStream.stop(); }
 catch(IOException ex)
 { myEx((Exception)ex, "RTP_Stream pauseStream"); }
}

public void teardownStream()
{
 if(!endofMedia)
 {
  pauseStream();
  closeMyStream();
 }
}

public boolean getMediaState()
{
 return endofMedia;
}

public void run()
{
 createMyProcessor();
```

```
  }
}
```

4.2 Below is the prototype of the multicast streaming

```
*/
package streaming.server.manager.transport.multicast;

import org.w3c.dom.Document;

/**
*/
public final class MulticastStream extends Thread {

        private Document doc;

        public MulticastStream(Document d) {
                doc = d;
        }

        public void run() {
                super.run();
        }

}
```

4.3 Below is the prototype for RTSP manager

```
package streaming.server.manager.rtsp;

import streaming.helper.error.ErrorLog;
import streaming.protocol.rtsp.ServerRtsp;
import streaming.protocol.rtsp.RTP_Ports;


import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;


public class RTSP_Manager
{

 private int serverport;
 private int base_rtp_port;
 private ServerSocket server;
 private RTP_Ports rtp_ports;
 private int port;

 public RTSP_Manager(int port, int base_port,int max)
 {
  serverport = port;
  base_rtp_port = base_port;
  rtp_ports = new RTP_Ports(max,base_port);
 }

 public final void startRtspServer()
 {
  try
  {
   server = new ServerSocket(serverport);
   while(true)
   {
    Socket client = server.accept();
    port = rtp_ports.getPort(false,0);
    ServerRtsp rtsp = new ServerRtsp(client, port, rtp_ports);
    rtsp.start();
```

```
    }
   }
   catch(IOException ex)
   { myEx((Exception)ex, "MyRTSP startRtspServer"); }
  }

  private final void myEx(Exception ex, String f)
  {
   f += " :";
   f += ex.getMessage();
   new ErrorLog(f);
  }
}
```

4.4 Simulations

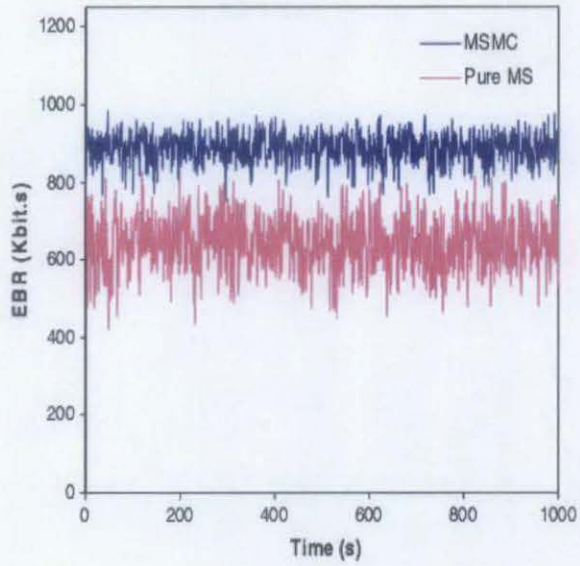


Figure 4.4.1 shows the Offered bit rate to the second receiver using multicast method and multi sender alone with 16 senders.

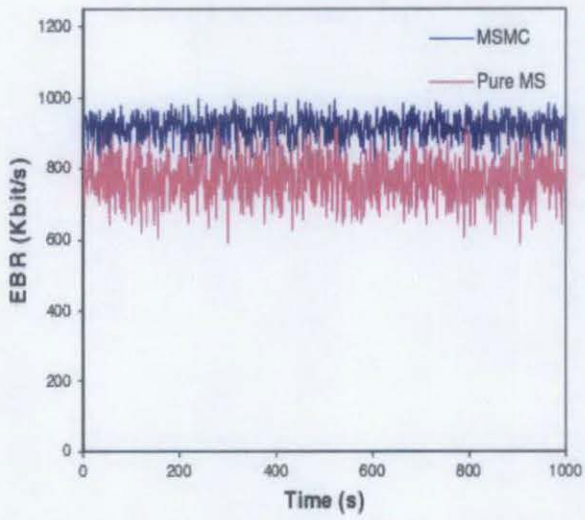


Figure 4.4.2 shows the offered bit rate to second receiver when senders are 32.

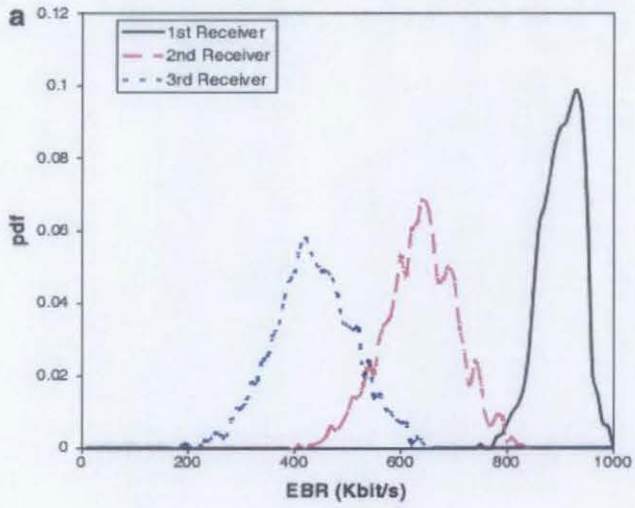


Figure 4.4.3 Histogram shows bit rate provided to first 3 receivers using the multicast method.

In this simulation a P2P network is simulated on a single 3.2GHz Intel Pc using JVM (Java Virtual Machine) technology. Each node has and uploads bandwidth of 300 kbps while the download bandwidth is also 300 kbps. This simulation shows the availability of nodes in short period of time (e.g 5 hours) is uniformly distributed between.

In Fig.4.4.1, the bit-rate provided to the second receiver overtime using the proposed method is compared to that provided using the pure unicast algorithm of run twice when 16 senders are contributing. It is observed that MSMC improves EBR of the second Receiver by 30%, rendering the same service quality as that given to the first receiver.

The effect of increasing the number of senders is shown in Fig.4.4.2 offered EBR to the second receiver in both algorithms is increased with the increased number of senders. Multicast out performs unicast again.

4.5 Discussion

4.5.1 Multicast

Multicast is the delivery of information to a group of destinations simultaneously using the most efficient strategy to deliver the messages over each link of the network only once, creating copies only when the links to the destinations split.

The word "Multicast" is typically used to refer to IP Multicast, the implementation of the multicast concept on the IP routing level, where routers create optimal distribution paths for datagrams sent to a multicast destination address spanning tree in realtime. But there are also other implementations of the multicast distribution strategy listed below.

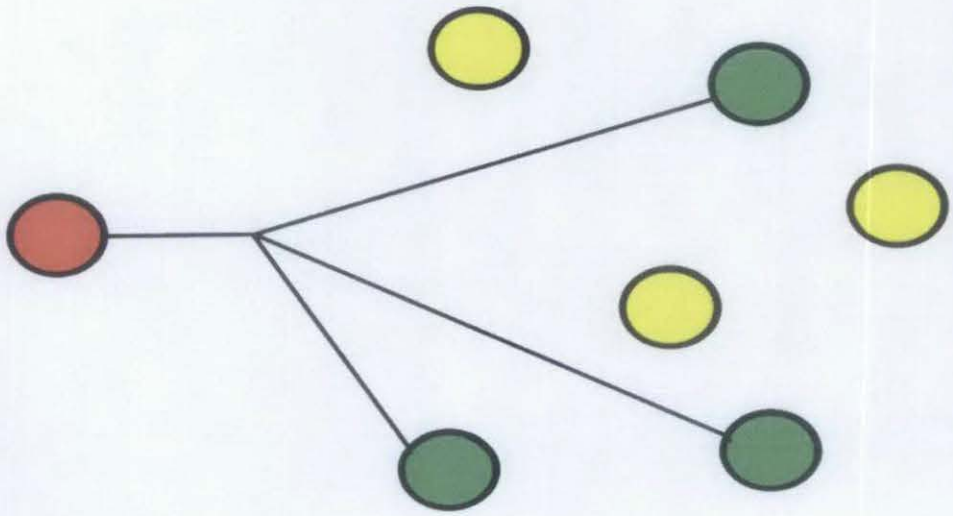


Figure 4.5.1a Figure is show the optimal distribution path for multicasting

4.5.2 Unicast

In computer networks, **unicast** is the sending of information packets to a single destination. "Unicast" is derived from the word broadcast, as unicast is the extreme opposite of broadcasting. In computer networking, **multicasting is used to regain some of the efficiencies of broadcasting.**

These terms are also synonymous with streaming content providers' services. Unicast servers provide a stream to a single user at a time, while **multicast servers can support a larger audience by serving content simultaneously to multiple users.**

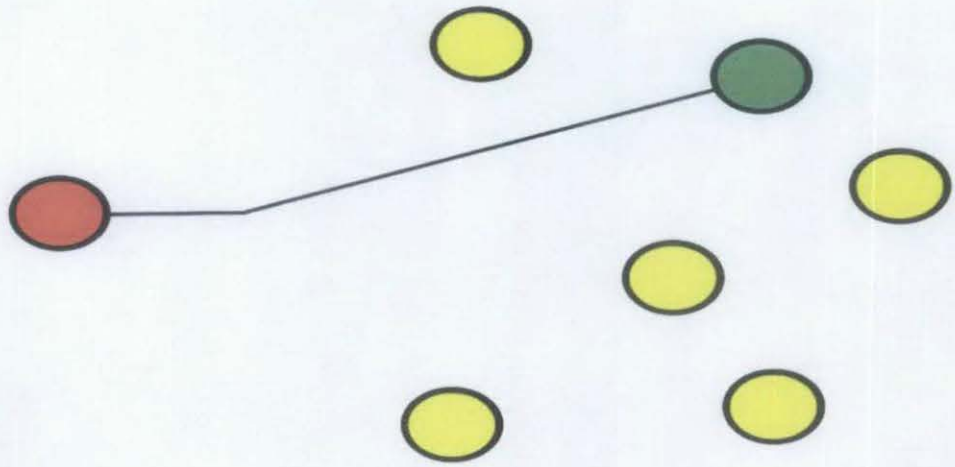


Figure 4.5.2a Figures shows the optimal distribution path for unicast

4.6 Measuring Performance

Good output was used to measure the overall efficiency of the system in bandwidth utilization. Good output is defined as:

Good output = original data/ connection time

Where original data is the number of byte delivered to the high level protocol at the receiver (excluding retransmitted data and overhead). The connection time is defined as the time taken for the data to be completely delivered.

The task of specifying the effect of network QoS parameters on video quality was challenging. Transmission fluctuations, increased delay, jitter and packet loss commonly deteriorate the perceptual quality or fidelity of the received video content. However, these parameters do not affect the quality in independent manner; they act in combination or cumulatively and ultimately only this joint effect is detected by the end user.

5.0 CONCLUSION

From this interim we can conclude that P2P architecture can help to minimize the usage of network resources. It also reduces the cost of hardware and software in implementation of the video streaming in network environment.

All existing multi-sender methods maximize the bit-rate provided to the first requesting node.When another node requests the media from the same senders, the algorithm must be run again. Since the limited senders' bandwidths are already committed to the first receiver, the bit-rates offered to the next receivers are likely to be unacceptably low. A multi-sender algorithm tries to maximize the quality for a single receiver, and to that end, it uses up all good sources.

In this paper, I proposed a method that overcomes this problem by using temporary senders–the peers between the senders and the receivers that inevitably have parts of the streamed media. The simulation results demonstrate that, using the proposed method, the bit-rate offered to the second peer is almost the same as that offered to the first receiver that is maximized by the underlying multi-sender.

6.0 REFERENCES

1- http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6TYP-4NVH7WM-
A multi-sender multicast algorithm for media streaming on peer-to-peer networks.

2. - http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6WMK-4P2M71K -Special issue: Resource-aware adaptive video streaming
3. Distributed system Principle and Paradigms, Andrew S.Tanenbaum
4. http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6WJ0-4NJWP0V – Performance analysis of multimedia based web traffic with QoS constraints

5. J.Gross,J.Klaue,H.Karl,A.Wolisz,Cross-layer optimization of OFDM transmission systems for MPEG-4video streaming, Computer Communications27(11)(2004)1044–1055.

6. S.Kulkami,J.Markham,Splite and merge multicast: live media streaming with application level multicast, IEEE International Conference on Communication 2(2005)1292–1298,16-20.