ONLINE TUTORING USING MOBILE PLATFORM

Srividhya Gopalakrishnan
B.E., Anna University, India, 2006

PROJECT

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

SPRING
2011
ONLINE TUTORING USING MOBILE PLATFORM

A Project

by

Srividhya Gopalakrishnan

Approved by:

__________________________, Committee Chair
Ahmed Salem, Ph.D.

__________________________, Second Reader
Meiliu Lu, Ph.D.

__________________________
Date

iii
Student: Srividhya Gopalakrishnan

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit to be awarded for the project.

__________________________, Graduate Coordinator
Nikrouz Faroughi, Ph.D.                     Date

Department of Computer Science
Abstract

of

ONLINE TUTORING USING MOBILE PLATFORM

by

Srividhya Gopalakrishnan

Mobile applications are one of the fastest growing technologies in today’s global market. Mobile applications run inside a mobile device and provide a rich and fast user experience in the palm of the user. Online tutoring has progressed leaps and bounds and considered a preferred means of tutoring. In the recent past, there is a remarkable increase in research around e-learning and their use in education industry. With an audio/video capturing/transmitting mobile device any room can become an interactive classroom.

The aim of this project is to implement an online tutoring service using a mobile platform called Android. Android mobile device was used as a video capturing and transmitting device. Unique username and password are provided to students to access the live sessions using a web interface. The tool consists of three main components:

- Client Program (Android Device)

  The client program is responsible for capturing image frames and audio packets and transporting it to the server.
• Network Server

The network server is responsible for receiving, encoding and transmitting the audio/video to the Red5 server.

• Streaming Server: Red 5 Server

The Red5 server is responsible for streaming the content to web browsers.

_____________________, Committee Chair
Ahmed Salem, Ph.D.

_____________________
Date
To my Father
ACKNOWLEDGMENTS

I would like to thank Dr. Ahmed Salem, my project advisor for the guidance that he has provided me through this project.

I would also like to thank Dr. Meiliu Lu for giving me her time and input as second reader.

I would like to thank my husband, Manikandan, who has given me the time and support to finish this project. I couldn’t have done it without him. In addition, I would like to thank my mother and my in-laws for their support throughout the coursework.

I would also like to thank my son Raghav for giving me the time to finish this project and my course.

My special thanks to my colleague Tami Shaw for reviewing my project report. Also my special thanks to Whitney Clark, my workshop instructor from Student center for helping me with project formatting.

Also I would like thank all my professors, who taught me. Without their assistance and guidance, my Master’s degree would not have been completed successfully.
TABLE OF CONTENTS

Dedication ........................................................................................................................................ vii
Acknowledgements ..................................................................................................................... viii
List of Figures .................................................................................................................................. xiii

Chapter

1. INTRODUCTION ......................................................................................................................... 1
2. BACKGROUND ............................................................................................................................ 3
   2.1 Need for online tutoring using mobile ............................................................................... 3
   2.2 Scope of the project ............................................................................................................. 3
   2.3 Audio Video Basics .......................................................................................................... 4
       2.3.1 Flash player .............................................................................................................. 4
       2.3.2 Flash video format ................................................................................................. 4
       2.3.3 Encoding and decoding ......................................................................................... 5
       2.3.4 Image compression techniques .............................................................................. 5
3. TECHNOLOGY ............................................................................................................................. 7
   3.1 Mobile platform ................................................................................................................... 7
   3.2 Java ..................................................................................................................................... 7
   3.3 Asp.net ............................................................................................................................... 7
   3.4 Other components ............................................................................................................... 8
3.4.1 Xuggler – FFmpeg ................................................................. 8
3.4.2 Red 5 .................................................................................. 8

4. ARCHITECTURE AND PROGRAMMING CONCEPTS ............. 10
4.1 Methodology used ..................................................................... 10
4.2 System flow ............................................................................. 11
4.3 Programming concepts ............................................................ 13
  4.3.1 Multi-threading .................................................................. 13
  4.3.2 Optimized collection ........................................................ 13
  4.3.3 Encoding and decoding ..................................................... 14
  4.3.4 Network sockets and streams ......................................... 14

5. SYSTEM COMPONENTS AND IMPLEMENTATION .................. 15
5.1 Network server ........................................................................ 15
  5.1.1 Listen for client connections ........................................... 15
  5.1.2 Receive media packets from client .................................... 16
  5.1.3 Transfer the media packets to transport for assembling .... 16
  5.1.4 Encode the assembled audio and video packets .......... 17
  5.1.5 Stream the encoded packets to red5 server .................... 17
  5.1.6 Apply network latency corrections during encoding ....... 17
5.2 Android Client ................................................................. 18
  5.2.1 Provide an UI for the user to start and stop transmission of media .... 18
5.2.2 Acquire control of the device camera and capture image frames .................... 19
5.2.3 Acquire control of the device microphone and capture audio packets ............ 20
5.2.4 Encode the YUV frames from the camera to compressed JPEG images ....... 20
5.2.5 Transfer the frames images and audio data to the transport layer ............. 21
5.3 Streaming server ..................................................................................................... 21
5.3.1 Web interface ................................................................................................... 21
5.4 Transport layer ........................................................................................................ 23
5.4.1 Provide a store for clients to offload data to be disassembled and transmitted................................................................. 23
5.4.2 Provide a store for clients to retrieve back the assembled data .................. 24
5.4.3 Provide a mechanism for disassembling and assembling media packets .... 25
5.4.4 Provide the actual network implementation .................................................... 26
5.5 Implementation constraints ..................................................................................... 26
5.5.1 Android client .................................................................................................. 26
5.5.2 Java server ........................................................................................................ 27
5.5.3 Transport .......................................................................................................... 27
6. CONCLUSION AND FUTURE WORK ..................................................................... 28
6.1 Future work ............................................................................................................. 28
6.1.1 Optimization .................................................................................................... 28
6.1.2 To create a collaborative session .................................................................. 29
Appendix Source Code ................................................................. 30
Bibliography ........................................................................ 80
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Architecture diagram</td>
<td>11</td>
</tr>
<tr>
<td>Figure 2</td>
<td>System flow diagram</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Android screen</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Web interface - student login page</td>
<td>22</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Web interface to view live session</td>
<td>23</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

Mobile applications are a rapidly developing segment of the global mobile market. They consist of software that runs on a mobile device and performs certain tasks for the user of the mobile phone. Mobile applications have evolved to the point that they now give users a rich and fast experience.

Online tutoring has progressed leaps and bounds and has reached a stage where it is considered as one of the preferred means of tutoring. With an audio/video capturing/transmitting device any room can become a classroom. The method of delivering education by means of mobile devices is called as M-Learning [1]. M-Learning is the follow up of E-Learning. The rapid growth in the mobile and communication sector and recent innovations like iPhone and Android have made it possible to visualize new ways of tutoring. This was one of my motivations behind this project using Android, to try M-Tutoring which is the online tutoring system.

Using Android as a media capturing device, audio and video is streamed live to the server where students can view their live classes through the browser. The fascination about audio video streaming and technology used by YouTube to deliver video content was another motivation to do this project and is explained in Chapter 2. Chapter 2 also includes the audio video basics.

The need and the scope of the project is explained in chapter 2. The details about the technology and the programming concepts used to achieve successful audio and video streaming is explained in chapter 4. Chapter 5 explains the logic and methodology used
to meet the objective of the project. Chapter 5 also provides the constraints and the reasoning behind the design and implementation being used in the project. Chapter 6 concludes the learning experience with the future implementation plan.

The main objective of this project is the real time video streaming from any smart phone running Android OS [Version 2.2 or above] and video capture capability. Online Tutoring is used as the business case to demonstrate this project. Even though the project doesn’t cover all use cases of Online Tutoring, it helps to bring out the capabilities of live video streaming from Android and opens up the door for future implementation plans.
Chapter 2

BACKGROUND

With the immense growth in the mobile industry, there are many applications related to video streaming. There only a few applications like QiK [2] which does real time live video streaming, however it is for social media streaming purposes. QiK allows recording videos and sharing them instantly. But as mentioned it is used only for social media purposes.

2.1 Need for online tutoring using mobile

The future generation is moving towards mobile devices and tablets. The world is shrinking away from laptops to smart phones and tablets. This is true in education too, where education moved from distance learning (via postal) to E-Learning and now to M-Learning [1]. The latest innovations in technology can help the education industry to reach far more people. This project proposes one such solution for M-Tutoring, where tutors can conduct classes from their convenient location. The classroom is streamed live to server. The students can view the session through web browser. As a future work, we can create a collaborative session where students can also participate in the sessions using mobile where two way communications is enabled.

2.2 Scope of the project

The Scope of the project is to achieve the following:

- To use a smart phone as a one way streaming device (classroom to web)
- To achieve minimum of 7-8 frames per second
• To have an uninterrupted streaming for at least 30 minutes
• To develop an web interface to view the streamed content

2.3 Audio Video Basics

In today’s internet, media consumes most of the bandwidth and data utilization. Face book and twitter have progressed beyond being a social platform to very essential way of expression and communication. Audio and video are very important types of media. The audio and video can be transmitted in different formats like mp3, mp4, aac etc. Playing these formats requires the corresponding players. The main format used in this project is called FLV. A flash media player is required to play an FLV file [3].

2.3.1 Flash player

The flash player is a cross-platform software application and is available as plug-in for recent versions of browsers to view a wide variety of multimedia movies, animations, etc. This project uses Adobe Flash player for playing the flv format files. Flash Player is installed on far more computers that other media players, and even if the end user does not have the latest player installed, it is approximately 0.6 mb and installs very easily.

2.3.2 Flash video format

The FLV video format is one of the best available for presenting video clips on an internet website. It has good compression rate, and it is browser independent. The format is highly secure, in particular, it cannot be downloaded, edited or transcoded. For applications like online tutoring, where the lectures are highly secured and not for distribution, this format suits the need.
2.3.3 Encoding and decoding

Encoding is the process of transformation of information from one format to another and decoding is the reverse process. FFMPEG is a commonly used library to process and convert audio/video file to flv format. Audio/video data is stored as files on the web server. These files are also referred to as containers. The container has the header, packet and the trailer. An FLV format does not contain the trailers. The header is the first 9 bytes and indicates what streams are in the container, and the codec used. Codec tells how the data is compressed. Each packet has a timestamp based on when the computer decodes the packet and plays the file. FLV uses the key frame compression technique for compressing the video data. The player first reads the header for codes. Using the codec it configures the decoder. The audio packet is then sent to the audio decoder and video packets to the video decoder. Based on the timestamp, packets are played correspondingly [3].

2.3.4 Image compression techniques

Lossy compression technique

Lossy Compression is one of the encoding methods, which compress data by losing some of the data for larger transmission. This indicates, whether we can recover the original data completely after decompressing it. Lossy compression is generally used for video and sound, where most users will not detect a certain amount of information loss. The JPEG image file, commonly used for photographs and other complex still images on the Web, is an image that has lossy compression. Using JPEG compression,
the creator can decide how much loss to introduce and make a trade-off between file size and image quality.

**Lossless compression technique**

With lossless compression, every single bit of data that was originally in the file remains after the file is uncompressed. All of the information is completely restored. This is generally the technique of choice for text or spreadsheet files, where losing words or financial data could pose a problem. The Graphics Interchange File is an image format used on the Web that provides lossless compression.
Chapter 3

TECHNOLOGY

3.1 Mobile platform

The Android Mobile platform [Version 2.2 and above] from Google, is selected as the platform of choice for the following reasons:

1. It is one of the fastest growing platform in the mobile space
2. It is open sourced
3. It is supported across a wide variety of smart phones.
4. It provides an interface to develop in Java
5. It is backed by Google and the community [5]

3.2 Java

Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games and business applications. Java is an object oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web [6].

3.3 Asp.net

Asp.net is a web programming technology pioneered by Microsoft that allows developers to create dynamic web pages. Dynamic web pages are pages whose content is dynamically regenerated each time the web page is requested [7]. It is the successor to
Active Server Pages (ASP), which was Microsoft’s earlier dynamic web-page creation technology. Asp.net is built on the Common Language Runtime (CLR), allowing programmers to write the code using any supported .net language like C#, VB, C++, F#.

It is one of the most powerful and easiest technologies of the many available. This project utilizes the C#.net framework

3.4 Other components

3.4.1 Xuggler – FFMPEG

FFmpeg is an open source audio video encoding library. Xuggler is the Java wrapper for FFMPEG [9]. It has the capability to stream media to any streaming server. It is a free open-source library for Java developers to uncompress, manipulate and compress recorded or live video in real time. FFMPEG is a complete, cross-platform solution to record, convert and stream audio and video. It includes libavcodec – the leading audio/video code library [10]. FFMPEG can read and write images for each frame of a video sequence. FFMPEG supports different file and image formats through the libavformat library. The libavformat and libavcodec are the libraries that come with FFMPEG which allow us to access a large variety of video file formats. The libavformat deals with parsing video files and separating the streams contained in them and libavcodec deals with decoding raw audio and video streams [12].

3.4.2 Red 5

Red 5 [11] is the streaming server, a dedicated web server, or application installed on a server, that is designed purely for delivering video or audio files. Red 5 is an open source adobe flash media streaming server, that supports,
Streaming Audio/Video

- FLV
- MP3
- MP4 etc

Live Stream Publishing
Chapter 4

ARCHITECTURE AND PROGRAMMING CONCEPTS

4.1 Methodology used

The main components required for implementation are:

- Client Program (Android Device)
- Network Server
- Transport
- Streaming Server: Red 5 Server
- Web browser

The detailed explanations of these individual components are provided in chapter 5.

The overall architecture diagram with the main components for the Online Tutoring is explained in the figure 1 in next page. Also Figure 1 depicts the high level flow between the components. The Android device, the streaming client captures the images and transfers it to the network or the video server. The video server converts the images to flv movie using Xuggler and streams it to the Red 5 streaming server. The Red 5 server plays the live streaming content in the browsers from where students can view their live classes.
4.2 System flow

The two main flows of the system are:

1. Client flow
2. Server flow

The transport layer connects the client and server. The system flow and the tasks performed at each level are explained in the figure 2.
This diagram explains the flow of data through the system. Also it explains the high level sequence of steps required for the live video streaming. In the client end, the images frames and audio bytes are captured and compressed to JPEG images and are sent through the network as UDP packets. The transport layers acts as the data store and transfers the packets to the server. At the server end, the packets are received, assembled and converted into flv stream and streamed to the Red 5 server.
4.3 Programming concepts

4.3.1 Multi-threading

Threads are called as lightweight processes for enabling multi-processing. Multithreading enables programs to utilize the system resources efficiently. The main reasons for using Multi-Threading are:

1. Multi-threading helps the project to be responsive. As this is a networking project based on the internet, the system should be responsive for new connections, new clients, and new requests.

2. It keeps the processor busy and maximizes the resource utilization as mentioned above.

3. Simplifies the coding of interconnected tasks.

4.3.2 Optimized collection

Memory management is important challenge, with the amount of image frames generated per second. We overcome this challenge by using circular/ring buffer, which is an optimized collection.

Ring buffer

Ring buffer is an implementation of data structure where the last node points back to first node, making it as circular implementation. The data is loaded in the data structure in a linear fashion. When we reach the end of the structure, the data in the first position gets deleted and the new data gets loaded. Ring buffer is used, when the data is time sensitive and there is a need to manage memory efficiently. In this project, all the audio and video files are time stamped. We will not be able to play the file after that
particular time stamp, so we use ring buffer for storing the data. This helps in memory management.

4.3.3 Encoding and decoding

Video encoding is the process of preparing the video for output. The digital video is encoded to meet proper formats and specifications for recording and playback through the use of video encoder software, also referred to as video conversion. Video decoding is the reverse process, i.e. converting the encoded data into its original format. As explained in the previous chapters, we use FFMPEG for video encoding and decoding.

4.3.4 Network sockets and streams

UDP sockets are typically used in bandwidth-limited applications, where the overhead associated with resending packets is not tolerable. A good example of this is real-time network audio/video applications. If we are delivering packets of audio information over the network to be played in real-time, then there is no point in resending a late packet. By the time it gets delivered it will be useless, since the audio track must play continuously and sequentially, without backtracking.
Chapter 5

SYSTEM COMPONENTS AND IMPLEMENTATION

5.1 Network server

The Network server is an integral part of the application. The network server is a java application which performs the following functionalities:

1. Listen for client connections
2. Receive media packets sent by clients
3. Transfer the media packets to the transport for assembling
4. Encode the assembled audio and video packets and attach timestamp for playback
5. Stream the encoded packets to red5 server
6. Apply network latency corrections during encoding

5.1.1 Listen for client connections

Class Name: SocketServer.java

One of the main functions of the Network server is to listen for clients trying to connect to the server. The clients connect to the server through an UDP port. When a client tries to connect, the server starts a new thread and instantiates the UDP server class.

There are two possible network sockets available for clients to connect to the server. They are TCP sockets and UDP Sockets. TCP sockets provide a very reliable way for clients to connect to the server. The delivery of data transmitted is guaranteed through acknowledgment and re-transmission if required. The acknowledgements and re-transmission adds some overhead, but ensures the data reaches the server as long as the network is available.
UDP provides an unreliable but faster way to transmit data. Each UDP packet has a header which represents the sender, receiver and other information about the data being sent. When a sequence of UDP packets are sent through the network, each packet can travel through a different route and reach the destination out of sync. UDP doesn’t ensure service guarantee or re-transmission when a packet is lost. It is the responsibility of the receiver to identify the sequence and request packets which are lost.

UDP is being selected as the transmission methodology because in real time media communication, packets out of sync or lost cannot be used. It is important to receive the packets in time for encoding.

5.1.2 Receive media packets from client

Class Name: UDPServer.java

Once the connection is established between the client and the server, the client transmits the media packets. The server will have to receive the packets without any delay and transfer it to the transport layer for assembling. With the amount of clients connected to the server increasing, the delay in processing the data on the server adds a delay, which eventually affects the capability of the server to stream live content.

Based on local testing it is identified that each server can handle 10 clients without disrupting the live transmission.

5.1.3 Transfer the media packets to transport for assembling

Class Name: UDPServer.java

This is a handoff process. The key requirement here is to ensure the assembling process is handled by a separate thread. This ensures the receiving thread is available to receive requests from the client.
5.1.4 Encode the assembled audio and video packets

**Class Name: Imageencoder.java**

Encoding is the process where we transform data from one form to another. In our case we transform the images in jpeg format and the audio in amr format to FLV. Xuggler which is a wrapper to the famous FFMPEG library is utilized for this. The process takes the images in jpeg format, audio in amr format and the timestamp in milliseconds precision to create a FLV stream. This is a key functionality of the network server.

5.1.5 Stream the encoded packets to red5 server

**Class Name: Imageencoder.java**

Once the flv stream is created, there are two possible scenarios. One, the stream can be stored in the local file system to be retrieved later or two it has to be streamed live to a streaming server which can in turn transmit the media to all the clients listening the stream. Since we are implementing a live session, we transmit the stream to Red5 server which makes it available to the clients listening through a browser.

5.1.6 Apply network latency corrections during encoding

**Class Name: Imageencoder.java**

This is a parallel functionality performed by the network server. The Red5 server streams content based on the timestamp in the individual frames. When the frames receive delayed, the server interrupts the transmission and connection has to be restarted again. In order to prevent this, we need a mechanism which can adjust the timestamp and the frames per second depending on the data received from the client. This requires continuous monitoring of the data flowing through the UDPServer class and adjusting the encoding configurations.
5.2 Android Client

The Android client is the application which will be executed on the client device. We call this the client as it connects to the server and transmits the media packets as UDP packets. The key functionalities of the android client are as follows:

1. Provide an UI for the user to start and stop transmission of media
2. Acquire control of the device camera and capture image frames
3. Acquire control of the device microphone and capture audio packets
4. Encode the YUV frames from the camera to compressed JPEG images
5. Transfer the frames images and audio data to the transport layer for transmission

The client application continuously performs the above steps till the user stops the transmission.

5.2.1 Provide an UI for the user to start and stop transmission of media

Class Name: PreviewScreen.java, UMessagingActivity.java

The UI screen provides a mechanism to start and stop transmission of media. It is layed-out in a way so that the preview of the camera is visible to the user. The screen taken with the help of the simulator is shown figure 3. The video streaming starts when we click the Video button and the preview/streaming stops when the “leave” button is clicked. The other buttons invite, chat and snap are not used in this phase of the project.
5.2.2 Acquire control of the device camera and capture image frames

Class Name: PreviewScreen.java

Android provides an API to gain control of the device camera. When the application gains control of the camera, other applications will not be able to use them until the camera is released by the application. Android also provides a call back method to obtain the image frames from the camera as raw byte arrays. The configuration on the number of frames obtained every second and the dimensions of the images are set as configuration parameters to the camera. It is defined as configuration parameters in
IConstants.java. The FRAMERATE is set to 12 and width and height are set to 320 and 240 pixels respectively for the purpose of streaming

5.2.3 Acquire control of the device microphone and capture audio packets

Class Name: PreviewScreen.java

We need to gain control of the device microphone to capture audio packets which will be synchronized with the image packets at the server. Android OS provides an API to gain control of the device microphone and capture audio packets at a constant frequency. The audio data is received as RAW AMR. The packets are transmitted, as is, to the server. The server, with the aid of Xuggler, has the ability to transform the audio packets to MP3 and stream it as a part of FLV.

5.2.4 Encode the YUV frames from the camera to compressed JPEG images

The image frames obtained from the device camera are in Raw YUV format. The raw format provides a lot of details on the various characteristics of the images. The YUV format is large in size and cannot be transmitted through the network without delay. As a comparison, a typical network packet is 8K bytes. In a typical android device (say Motorola Droid 1), the size of a RAW 176 * 144 image is around 33K bytes. It will take 15 – 16 network packets to transmit a single image which is a serious performance hit. As a result, we transform the raw images to JPEG. JPEG image format is a lossy format which allows compressing an image to a small size by removing some of the details as explained in the previous chapters. A compressed JPEG image with 30% clarity will be around 2 – 3 K byes, which makes it possible for the image to be transmitted as a single packet.
Prior to version 2.2 of Android OS, the process of converting an image from YUV to JPEG was very tedious. The algorithm was time consuming and added a delay in transmission. With version 2.2 android provides an API to do the same.

5.2.5 Transfer the frames images and audio data to the transport layer

This is the final step in the process for media transmission for the client. It is a hand-off process. The client hands over the image and audio packets to the transport layer which is responsible for transporting the media packets to the server.

5.3 Streaming server

Red5 is used as the streaming server for this project. Red5 is an open source Adobe flash streaming server written in java. It provides a way for clients to connect to the server through a browser and view the stream which is being transmitted. Red5 provides the infrastructure for multiple clients to connect simultaneously and view the transmission. Red5 provides a network address (URL) for the stream to be transmitted. Clients can connect to the stream through a browser and view the transmission. The FLV file format was created by Adobe and it is made available in all flash plug-ins for browser. Since all browsers current days have the flash plug in preinstalled, it eliminates the need to create a media player to view the content.

5.3.1 Web interface

As mentioned earlier, students can view their session through the browser. The web interface screen shot is provided as below in the figure 4. The user can login through the provided username and password.
As can be seen in Figure 5, once a student has successfully logged into the web interface, they type in the session name provided by the tutor in the column “name,” to start viewing the session. They can pause the session at any point using the stop button, and click the start button to continue viewing.
5.4 Transport layer

The transport layer performs the very important functionality of disassembling media data to be transmitted across the network and then re-assembling it on the server so that it can be encoded. The key features provided by the transport layer are as follows:

- Provide a store for clients to offload data to be disassembled and transmitted
- Provide a store for clients to retrieve back the assembled data
- Provide a mechanism for disassembling and assembling media packets
- Provide the actual network implementation for transmitting data on the network

5.4.1 Provide a store for clients to offload data to be disassembled and transmitted

Class Name: DataStore.java, RingBuffer.java, IncomingDataStore.java
The two main clients to the transport are the android client and the network server. The android client generates data at a very high frequency (10 – 12 images and audio packets per second) and without a mechanism to offload the data, the client will not be able to consistently transmit data to the server. This poses another challenge that with the amount of data being collected there needs to be an efficient way to clear up unused memory to optimize resource utilization and maintain efficiency.

Transport achieves this by using a ring buffer implementation in java. A ring buffer or a circular buffer is an implementation where previously used memory will be overwritten to achieve reuse. This works very well with time sensitive data. When there is a delay in transmission, then the data is of no use and can be overridden. The size of the buffer defines the quantity of elements which will be written to the buffer before the first element is overwritten. In our case, the size is set to 20. This size should be determined based on the available resources and the time it takes to transmit the data to the server. In other words, the time after which the data becomes of no use. The ring buffer is implemented as an object array with 20 elements.

The transport allows multiple types of data like audio, video, etc. to be sent as part of a single network packet. The network packet has a header that qualifies the data present inside the packet. The transport has a separate instance of ring buffer for each type of media.

5.4.2 Provide a store for clients to retrieve back the assembled data

Class Name: TransportStore.java, ringBuffer.java, OutgoingStore.java
This is the reverse process of the previous step. When the data is assembled back to the original media, the server process must have a means to retrieve the media packet. The transport also takes care of sequencing the data which was assembled. As a result then the network server requests for a media packet, it receives the data in the order it was received from the client. The implementation is FIFO.

5.4.3 Provide a mechanism for disassembling and assembling media packets

Class Name: NetworkPacket.java, DataPacket.java, ProcessedData.java

In real time communication, network latency is the main factor which can break communication. It becomes very critical that we can transmit media packets with minimal network latency and failovers. The transport provides a mechanism to do this by constructing network packets which are always of same size. NetworkPacket.java represents a network packet which will be transmitted across the network. Each network packet will contain multiple data packets. Each data packets represent data of a particular media type. The network packet has a defined size and each data packet has a defined size based on the media type. The transport ensures the network packet is always packed as long as data is available to be transmitted. If a media of a particular type is unavailable, then the space is allotted to the next available media type with data. This ensures the assembling and disassembling process can be executed over and over with very little failover.

In order to speed up the assembling process, each network packet has a defined header which qualifies the data available inside the packet. On the server, when
disassembled packets are received the transport needs a mechanism to know if a packet is complete. This information is also obtained from the header.

5.4.4 Provide the actual network implementation

Class Name: Transport.java

This class provides the actual implementation of a UDP packet transport. This will be executed as a separate thread. The network thread continuously pulls network packets from the outgoing Store and sends it to the server. Java provides a standard implementation to perform network calls. The java.net package APIs are used for this.

5.5 Implementation constraints

As this project is all about networks and data which are time sensitive, there were many implementation constraints. This chapter explains these constraints at module level along with the reason for using the programming concepts explained in chapter 4. Additionally this chapter helps the reader to think about optimization in these areas.

5.5.1 Android client

- Real-time frame capture
  The image frames from the android camera are in YUV format. YUV format is a raw image format and is not suitable for network transmission. The YUV image should be converted to JPEG format, so that it can be compressed to images of smaller size to be transmitted across the network.

- Network Latency
  Since we are using internet for transmitting the images generated, there is no guarantee for the transmission time.
- Memory Management /Resource Utilization

Real time audio and video capture creates a lot of image arrays which needs to be efficiently handled to avoid resource exhaustion.

For reducing the size of images, we are using lossy compression, so that we can get the image in JPEG format. For efficient memory management and resource utilization we are using Ring buffers.

5.5.2 Java server

- Real time audio video synchronization
- Managing multiple clients
- Fault tolerance due to network latency
- Resource Utilization and threading
- Real time streaming to Red 5
- Adaptive audio/video encoding to handle network unpredictability

5.5.3 Transport

- Ring buffer implementation for optimized memory management
- Data packet splitting and reassembly to handle network impedance

The transport layer splits the data packets into smaller chunks of data and sends to server. The biggest challenge on the server is to recreate these packets to get back the actual data.
Chapter 6

CONCLUSION AND FUTURE WORK

In this project, real time video streaming in an Android device is achieved. The client application can be deployed in the market place, which can be downloaded by the tutors. The real time video streaming is used for Online Tutoring. As a conclusion to this report, we can say that this project has accomplished its primary goals and objectives as discussed in scope section 2.2 of Chapter 2. Also all the implementation constraints have been addressed and we are able to stream audio and video for more 2 hours. The streamed content maintains a consistent delay of 3 seconds which makes the video viewing experience pleasant.

6.1 Future work

The online tutoring application will be a first step in using mobile technology to enable M-Tutoring. There is a lot of future scope for improvement. Mobile to mobile streaming, will aid in viewing the stream on a mobile device. Also this project helped me learn more about networks and the above mentioned implementation constraints helped me to learn the art of more efficient ways of coding.

The other major areas for future work are:

6.1.1 Optimization

Echo elimination

An echo is the reflection of sound, arriving at the end user sometime after the direct sound. As it is the same audio, the packets will be having same timestamp. We need to identify them and eliminate the same.
Variable bit rate (VBR)

It is the encoding method used to achieve better audio/video quality. Practically even with 5 image frames, we can generate a video. But if we have more image frames, the quality will be better. So variable bit rate is adjusting the number of image frames dynamically according to the captured data.

6.1.2 To create a collaborative session

In this phase of this project, the students can only listen to the live video. In the future we can create a collaborative session, where the video will stream live from server to mobile and also from mobile to mobile as complete mobile application. This will include transmitting text based messages in real time and allowing the tutor to respond to the messages.
UMessagingActivity.java

```java
package com.android.client;
import com.android.client.R;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.graphics.Color;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;
/**
 * @author Vidhya
 *
 */
public class UMessagingActivity extends Activity {
    // handle to the camera preview surface view WHICH ALSO HANDLES BUTTON EVENTS
    private PreviewScreen mPreview;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Hide the window title.
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.umessagingactivity);
        // The main screen which will hold the title, preview and buttons
        LinearLayout mScreen = new LinearLayout(this);
        mScreen.setOrientation(LinearLayout.HORIZONTAL);
        /********
        * @author Vidhya
        *
        /
        public class UMessagingActivity extends Activity {
            // handle to the camera preview surface view WHICH ALSO HANDLES BUTTON EVENTS
            private PreviewScreen mPreview;
            @Override
            protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                // Hide the window title.
                requestWindowFeature(Window.FEATURE_NO_TITLE);
                setContentView(R.layout.umessagingactivity);
                /********
                * @author Vidhya
                *
                
```
// The container to hold the Title band
LinearLayout mTitleBand = new LinearLayout(this);
mTitleBand.setOrientation(LinearLayout.HORIZONTAL);
mTitleBand.setMinimumHeight(120);

// The textview to present the session title
TextView tTitle = new TextView(this);
// tTitle.setText("Session no -" + sessionId);
tTitle.setTextSize(20);
tTitle.setGravity(Gravity.CENTER);

// Layout to hold the preview screen
LinearLayout mLayout = new LinearLayout(this);
mLayout.setOrientation(LinearLayout.VERTICAL);
mLayout.addView(mPreview, new LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT, LinearLayout.LayoutParams.FILL_PARENT, 1.0f));

// Layout to hold the buttons on the bottom
LinearLayout mButtonLayout = new LinearLayout(this);
mButtonLayout.setOrientation(LinearLayout.VERTICAL);

// Buttons to be presented in the bottom
Button btInvite = new Button(this);
btInvite.setId(IConstants.BUTTON_INVITE_ID);
btInvite.setGravity(Gravity.CENTER);
btInvite.setText("Invite");
btInvite.setHeight(50);
btInvite.setWidth(80);
btInvite.setOnClickListener(mPreview);

Button btVideo = new Button(this);
bVideo.setId(IConstants.BUTTON_VIDEO_ID);
bVideo.setText("Video");
bVideo.setHeight(50);
bVideo.setWidth(80);
bVideo.setOnClickListener(mPreview);

Button btChat = new Button(this);
bChat.setId(IConstants.BUTTON_CHAT_ID);
bChat.setText("Chat");
bChat.setHeight(50);
bChat.setWidth(80);

Button btSnap = new Button(this);
bSnap.setId(IConstants.BUTTON_SNAP_ID);
bSnap.setText("Snap");
bSnap.setHeight(50);
bSnap.setWidth(80);
bSnap.setOnClickListener(mPreview);
// Button to present the Leave option
Button btLeave = new Button(this);
// set the id for later identification
btLeave.setId(IConstants.BUTTON_LEAVE_ID);
bttLeave.setText("Leave");
bttLeave.setHeight(50);
bttLeave.setWidth(80);
bttLeave.setOnClickListener(mPreview);

// add buttons to the button Layout screen
mButtonLayout.addView(btInvite,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));
mButtonLayout.addView(btVideo,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));
mButtonLayout.addView(btChat,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));
mButtonLayout.addView(btSnap,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));
mButtonLayout.addView(btLeave,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));

// set the screen settings for the layout screen
mButtonLayout.setGravity(Gravity.LEFT);
mButtonLayout.setMinimumHeight(60);
mButtonLayout.setHorizontalFadingEdgeEnabled(true);
mScreen.addView(mTitleBand,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.WRAP_CONTENT,
1.0f));
mScreen.addView(mLayout,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.WRAP_CONTENT,LinearLayout.LayoutParams.FILL_PARENT,
1.0f));
mScreen.addView(mButtonLayout,new LinearLayout.LayoutParams
(LinearLayout.LayoutParams.FILL_PARENT,LinearLayout.LayoutParams.FILL_PARENT,1.0f)
);

// add the container layout to the screen
FrameLayout fl = (FrameLayout)findViewById(R.id.layout);
fl.addView(mScreen);
} }
package com.android.client;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.net.Socket;
import java.util.zip.Deflater;
import com.network.transport.ByteBufferOutputStream;
import com.network.transport.TransportStore;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.ImageFormat;
import android.graphics.PixelFormat;
import android.graphics.Rect;
import android.graphics.YuvImage;
import android.graphics.Bitmap.CompressFormat;
import android.graphics.Bitmap.Config;
import android.hardware.Camera;
import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import android.os.Environment;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;
public class PreviewScreen extends SurfaceView implements SurfaceHolder.Callback, Camera.PreviewCallback, Camera.PictureCallback, OnClickListener, Runnable, IConstants {
    SurfaceHolder mHolder;
    Camera mCamera;
    boolean blnCreateFile = true;
    MediaRecorder recorder;
    byte[] rawImage = null;
    int counter = 0;
    DataOutputStream out;
boolean processData = false;
int threadCount = 0;
boolean dataAvailable = false;
boolean connectToServer = false;
boolean takeSnap = false;
Socket socket;
boolean isRecording = false;
ByteArrayOutputStream bos = new ByteArrayOutputStream();

int bufferSize = AudioRecord.getMinBufferSize(AUDIO_FREQUENCY, AUDIO_CHANNEL, AUDIO_ENCODING);
AudioRecord audioRecord = new AudioRecord(MediaRecorder.AudioSource.MIC, AUDIO_FREQUENCY, AUDIO_CHANNEL, AUDIO_ENCODING, bufferSize);
byte[] buffer = new byte[bufferSize];

int[] processedImage = new int[IMAGE_PREVIEW_WIDTH * IMAGE_PREVIEW_HEIGHT];

private final TransportStore dFunnel = TransportStore.getInstance(SERVER, PORT, false);

PreviewScreen(Context context) {
    super(context);
    // Install a SurfaceHolder.Callback so we get notified when the
    // underlying surface is created and destroyed.
    mHolder = getHolder();
    mHolder.addCallback(this);
    mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
}

public void surfaceCreated(SurfaceHolder holder) {
    // The Surface has been created, acquire the camera and tell it where
    // to draw.

    try {
        if (mCamera == null) {
            mCamera = Camera.open();
        }
        mCamera.setPreviewDisplay(holder);
        Camera.Parameters parameters = mCamera.getParameters();
        parameters.setPictureFormat(PixelFormat.JPEG);
        parameters.setPictureSize(IMAGE_PREVIEW_WIDTH, IMAGE_PREVIEW_HEIGHT);
        parameters.setJpegQuality(IMAGE_QUALITY);
        parameters.setPreviewFrameRate(FRAME_RATE);
    }
}
parameters.setPreviewSize(IMAGE_PREVIEW_WIDTH, IMAGE_PREVIEW_HEIGHT);
    mCamera.setParameters(parameters);
} catch (IOException ex) {
    Log.e("test", ex.getMessage());
    mCamera.release();
    mCamera = null;
}

catch (Exception exception) {
    Log.e("test", exception.getMessage());
}

Log.i("test", "surface created, preview on..");

public void surfaceDestroyed(SurfaceHolder holder) {
    // Surface will be destroyed when we return, so stop the preview.
    // Because the CameraDevice object is not a shared resource, it's very
    // important to release it when the activity is paused.
    mCamera.setPreviewCallback(null);
    mCamera.stopPreview();
    mCamera.release();
    mCamera = null;
    Log.i("test", "surface destroyed, camera released..");
}

public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    // Now that the size is known, set up the camera parameters and begin
    // the preview.

    try {
        mCamera.setPreviewDisplay(holder);
    } catch (IOException e) {
        Log.e("test", e.getMessage());
    }

    Camera.Parameters parameters = mCamera.getParameters();
    parameters.setPreviewSize(w, h);

    mCamera.startPreview();
    Log.i("test", "surface changed..");
}

@Override
public void onPreviewFrame(byte[] picture, Camera camera) {
    // set the image buffer to a array
    /*
* sample size of rawimage
  * 176 * 144 -- 38600
  * 320 * 240 -- 115200
  * No of frames received is 18
*/
rawImage = picture;
dataAvailable = true;
//Log.i("test","Raw image available;Size of raw array is ...");
rawImage.length);
}

@Override
public void onPictureTaken(byte[] picture, Camera camera) {
  /*Save the data to the store*/
dFunnel.storeDatatosend(TransportStore.DATA_TYPE_PICTURE,picture);
}

@Override
public void onClick(View v) {
  //implement the button click events here
  switch(v.getId()){
    case IConstants.BUTTON_LEAVE_ID : {
      processData = false;
mCamera.stopPreview();
      stopAudioRecording();
      Intent intent = new Intent();
      intent.setClass(getContext(), Sample.class);
      //do clean up
      getContext().startActivity(intent);
      break;
    }
    case IConstants.BUTTON_INVITE_ID: {
      break;
    }
    case IConstants.BUTTON_VIDEO_ID:{
      try{
        if(!processData){
          mCamera.setPreviewCallback(this);
processData = true;
          connectToServer = true;
          startAudioRecording();
          Thread worker = new Thread(this);

```
worker.setName("Thread -> " + threadCount);
worker.start();
break;
}
}
}
}
}
}
}
}
}

} catch(Exception ex){
    Log.i("test", "error in video button click...") + ex.getMessage());
}

}

}

case IConstants.BUTTON_SNAP_ID : {
    takeSnap = true;
}

}

}

@Override
public void run() {
    String name = Thread.currentThread().getName();
    try {
        while(processData)
        {
            if(dataAvailable){
                long millis = System.currentTimeMillis();
                ByteBufferOutputStream out = dFunnel.getOutputStream();
                YuvImage frameImage = new YuvImage(rawImage, ImageFormat.NV21, IMAGE_CROP_WIDTH, IMAGE_CROP_HEIGHT, null);
                Rect imageRect = new Rect(0,0,IMAGE_CROP_WIDTH,IMAGE_CROP_HEIGHT);
                frameImage.compressToJpeg(imageRect, IMAGE_QUALITY, out);
                byte[] compressedImage = out.getArray();
                dFunnel.storeDatatosend(TransportStore.DATA_TYPE_VIDEO,compressedImage);
                Thread.currentThread().sleep(50);
                Log.i("test", name + "Size of compressed image is " + compressedImage.length + ": end of operations.... " + (System.currentTimeMillis() - millis));
            }
        }
        try {
            Thread.currentThread().sleep(5);
        }
        try {
            Thread.currentThread().sleep(5);
        }
    }

    

}
Thread.currentThread().sleep(1000);

} catch (InterruptedException e1) {
    Log.e("test",name + " : There is some other error " +
    e1.getMessage());
}

} catch (Exception e) {
    Log.e("test",name + " : There is some other error " +
    e.getMessage());

/**
 * Starts a new audio recording.
 */
public void startAudioRecording() {
    try {
        audioRecord.startRecording();
        isRecording = true;
        Thread audioThread = new Thread()
            {public void run()
                
                while(isRecording){
                    dFunnel.storeDatatosend(TransportStore.DATA_TYPE_AUDIO,recordAudio());
                    try {
                        sleep(3);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        audioThread.start();
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}

/**
 * Stops a audio recording that has been previously started.
 */
public void stopAudioRecording() {
    try {
        audioRecord.stop();
        isRecording = false;
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}
IConstants.java

```java
package com.android.client;
import android.media.AudioFormat;
public interface IConstants {
    public static final String PROTOCOL = "HTTP://";
    //public static final String SERVER = "69.57.167.131";
    //public static final String SERVER = "10.9.16.97";
    public static final String SERVER = "130.86.207.170";
    public static final int PORT = 10099;
    public static final String APPLICATION_URL = "/vmc_product_account/php/iphone_ver1/";
    //public static final String AUTHENTICATE_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "authenticateUser.php";
    public static final String AUTHENTICATE_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "authenticateUser.php";
    public static final String GET_USER_DETAIL_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "getUser.php";
    public static final String GET_USER_SESSIONS_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "getSRs.php";
    public static final String GET_USER_GROUPS_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "getGroups.php";
    public static final String GET_SR_COMPONENTS_REQUEST = PROTOCOL + SERVER + APPLICATION_URL + "getComponents.php";
    public static final int SERVER_SCRIPT_ID_AUTHENTICATE_USER = 1;
    public static final int SERVER_SCRIPT_ID_PRIMARY_GET_USER = 2;
    public static final int SERVER_SCRIPT_ID_SET_PRIMARY_USER_ATTRIBUTE = 3;
```

```java
public byte[] recordAudio()
{
    try {
        int bufferReadResult = audioRecord.read(buffer, 0, bufferSize);
    }
    catch (Throwable t) {
        Log.e("AudioRecord","Recording Failed");
    }
    return buffer;
}
```
public static final int SERVER_SCRIPT_ID_GET_GROUPS = 4;
public static final int SERVER_SCRIPT_ID_GET_USER = 5;
public static final int SERVER_SCRIPT_ID_GET_SERVICE_REQUESTS = 6;
public static final int SERVER_SCRIPT_ID_CREATE_SERVICE_REQUEST = 7;
public static final int SERVER_SCRIPT_ID_SET_SERVICE_REQUEST_ATTRIBUTE = 8;
public static final int SERVER_SCRIPT_ID_GET_COMPONENTS = 9;
public static final int SERVER_SCRIPT_ID_ADD_SR_VIDEO_COMPONENT = 10;
public static final int SERVER_SCRIPT_ID_ADD_SR_CHAT_COMPONENT = 11;
public static final int SERVER_SCRIPT_ID_ADD_SR_ACTIVITY = 12;
public static final int SERVER_SCRIPT_ID_INVITE_USERS = 13;
public static final int SERVER_SCRIPT_ID_GET_CHAT = 14;
public static final int SERVER_SCRIPT_ID_ADD_SNAP = 15;
public static final int SERVER_SCRIPT_ID_GET_VIDEO_FILE_PATH = 16;
public static final int SERVER_SCRIPT_ID_GET_SNAP_FILE_PATH = 17;

/**
 * constants to define the XML tag
 */
public static final String XML_TAG_BEGIN_TAG = "z";
public static final String XML_TAG_ATR = "a";
public static final String XML_TAG_ATRV = "v";
public static final String XML_TAG_ATRN = "n";
public static final String XML_TAG_OLD_MESSAGE = "message";
public static final String XML_TAG_MESSAGE = "m";

public static final String XML_TAG_SNAP_FILE_PATH = "snap_file_path";
public static final String XML_TAG_VIDEO_FILE_PATH = "video_file_path";
public static final String XML_TAG_SR_COMP_ID = "sr_comp_id";
public static final String XML_TAG_SR_ID = "sr_id";
public static final String XML_TAG_GET_SR_REQUEST = "get_service_requests";
public static final String XML_TAG_GET_COMPONENTS = "get_components";
public static final String XML_TAG_COMPONENTS = "component";

public static final String XML_TAG_UID = "u";
public static final String XML_TAG_GID = "gid";
public static final String XML_TAG_GROUPS = "gs";
public static final String XML_TAG_GROUP = "g";
public static final String XML_TAG_GN = "gn";
public static final String XML_TAG_USER = "u";
public static final String XML_TAG_TEXT = "t";
public static final String XML_TAG_USERS = "us";
public static final String XML_TAG_SR = "sr";
public static final String XML_TAG_ACCOUNT_ID = "ai";

/**
 * Constants for button ID
 */
public static final int BUTTON_LEAVE_ID = 1;
public static final int BUTTON_INVITE_ID = 2;
public static final int BUTTON_SNAP_ID = 3;
public static final int BUTTON_CHAT_ID = 4;
public static final int BUTTON_VIDEO_ID = 5;
public static final int BUTTON_AUDIO_ID = 6;
public static final int BUTTON_CANCEL_ID = 7;
public static final int BUTTON_INVITE_CONFIRM_ID = 9;
public static final int BUTTON_USER_DETAILS_ID = 10;

/**
 * Sample image size
 *
 * 176 * 144
 * 320 * 240
 */

/*Properties related to images*/
public static final int IMAGE_THRESHOLD_COUNT = 1;
public static final int IMAGE_PREVIEW_WIDTH = 320;
public static final int IMAGE_PREVIEW_HEIGHT = 240;
public static final int IMAGE_CROP_WIDTH = 320;
public static final int IMAGE_CROP_HEIGHT = 240;
public static final int IMAGE_QUALITY = 30; // scale of 1 - 100
public static final int FRAME_RATE = 20;

/*properties related to audio*/
public static final int AUDIO_FREQUENCY = 11025;
public static final int AUDIO_CHANNEL = AudioFormat.CHANNEL_CONFIGURATION_MONO;
public static final int AUDIO_ENCODING = AudioFormat.ENCODING_PCM_16BIT;

TransportStore.java

package com.network.transport;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.util.Log;

/**
 * @author Vidhya
 * the primary controller class for the TransportStore. This class is responsible for providing a
 * store for the client to deposit data, the transport to send the data to the server, Maintain
 * the * bandwidth share for different data types and ensure the communication with minimum
 * latency
 */

public class TransportStore implements IConstants, Runnable {
    /* variable to hold the DataStore */
    private final OutgoingStore storeSend = OutgoingStore.getInstance();
    private final IncomingStore storeRecv = IncomingStore.getInstance();
    /* variable to store the transport */
    private Transport transport = null;
    /* local instance */
    private static TransportStore instance = null, recvInstance = null;
    /* variable to Data Packet */
    private final DataPacket dPacketSend = new DataPacket();
    private final DataPacket dPacketRecv = new DataPacket();
    /* variable to Data Packet */
    private final NetworkPacket nPacketSend = new NetworkPacket();
    private final NetworkPacket nPacketRecv = new NetworkPacket();
    /* variable to hold processed Image */
    private final ProcessedData processedDataRecv = new ProcessedData();
    /* array to hold bytes to read from each type */
    private int[] dataBytesToSendContainer = new int[IConstants.DATA_TYPES_SUPPORTED_COUNT];
    int[] dataBytesAvailableToSend = new int[IConstants.DATA_TYPES_SUPPORTED_COUNT];
    /* instance of the ByteBufferOutputStream */
    private ByteBufferOutputStream byteBuffer = null;
    /* read pointers for command types */
    private int cBytesToRead = 0;
    private int bytesAvailable = 0;
    private long dataLength = 0;
    private boolean sliceData = false;
    private boolean processThreadStatus = false;
    private boolean testMode = false;
    private boolean recvThreadStatus = false;
    private boolean EOF = false;
    private static boolean isVideoDataInt = false;
    private long currentTime = 0;
    private long imagePerSecond = 0;
}
private int storeCounter = 0, processedImageCounter = 0;
private boolean serverMode = false;
private String Receive_Thread_Lock = "LOCK";

/* constructor to accept server and port. this will be used for sending */
private TransportStore(String server, int port) {
    transport = Transport.getInstance(server, port);
    byteBuffer = new ByteBufferOutputStream();
}

/* Default constructor. This can only be used in receive mode */
private TransportStore() {
    serverMode = true;
}

/* Default constructor. This can only be used in receive mode */
public TransportStore(boolean sMode) {
    serverMode = sMode;
}

/* the method will return the outputstream to compress the image. This is special
   requirement for the Bitmap class in android, which requires a outputStream to
   compress the image. */
public ByteBufferOutputStream getOutputStream() {
    return byteBuffer;
}

/* method to get instance of the class */
public static TransportStore getInstance(String serverIP, int port) {
    if (instance == null) {
        instance = new TransportStore(serverIP, port);
    }
    return instance;
}

/* method to get instance of the class */
public static TransportStore getInstance(String serverIP, int port, boolean testMode) {
    if (instance == null) {
        instance = new TransportStore(serverIP, port);
    }
    instance.testMode = testMode;
    return instance;
}

/* method to get instance of the class */
public static TransportStore getInstance(boolean testMode) {
if(recvInstance == null) {
    recvInstance = new TransportStore();
}
recvInstance.testMode = testMode;
return recvInstance;

/*method for a client to deposit data*/
public void storeDatatosend(int mediaType, byte[] data) {
    if(!serverMode) {
        synchronized(storeSend) {
            storeSend.addDatatoStore(mediaType, data);
            if(!processThreadStatus) {
                processThreadStatus = true;
                sliceData = true;
                new Thread(this).start();
            } else {
                storeCounter++;
            }
            storeSend.notifyAll();
        }
    }
}

/*Convenience method to store int[] in the store*/
public void storeDatatoSend(int mediaType, int[] data) {
    /*The flag is to enable special processing for image*/
    if(!serverMode) {
        isVideoDataInt = true;
        storeSend.addDatatoStore(mediaType, data);
        if(!processThreadStatus) {
            processThreadStatus = true;
            sliceData = true;
            new Thread(this).start();
        }
    }
}

/*method to create network packets. This method will be called from inside a thread*/
@SuppressWarnings("static-access")
public void createNetworkPackets() {
    int counter = 1;
    imagepersecond = currenttime = System.currentTimeMillis();
    while(sliceData) {
        /*size of a network packet. will shrink when data is filled*/
bytesAvailable = IConstants.MAX_NETWORK_PACKET_SIZE;

/*The cumulative length of data available across all datatypes.
* Length of 0 signifies no data to send .*/
dataLength = getDataAvailableToSend();
if(dataLength > 0) {
    while(sliceData == true && (bytesAvailable > 0 &&
    dataLength > 0)) {
        /*use the following loop to get the data to be
        send*/
        for(int i = 0; i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++) {
            calculateDatatoSend(bytesAvailable, i + 1);
        }
        /*get the data from store and construct the network
        packet*/
        writeDatatoPacket();
        /*counters are reset to make it available for next network
        packet*/
        resetReadCounter();
        /*Send the network packet across the network*/
        sendNetworkPacket();
        /*EOF signifies the packet sent has a termination bit */
        if(EOF) {
            counter++;
            log("log", " TransportStore -- > Time taken for
            one image " + (System.currentTimeMillis() - currenttime));
            currenttime = System.currentTimeMillis();
            EOF = false;
            if(counter == 11) {
                log("info", " TransportStore -- > Time
taken for 11 image " + (System.currentTimeMillis() - imagepersecond));
                imagepersecond = System.currentTimeMillis();
                counter = 1;
            }
            try {
                /*Allow other theards to run*/
                Thread.currentThread().sleep(5);
            } catch (InterruptedException e) {
                log("error", "Exception in thread.." +
e.getMessage());
            }
        }
    }
} else {
try {
    /*Wait on the store till data is available*/
    synchronized(storeSend) {
        storeSend.wait();
    }
} catch (InterruptedException e) {
    log("error","Exception in thread..") +
    e.getMessage();
}

log("log", "Waiting for data....");

/*Signal to stop process and disconnect from server*/
transport.stopTransport();
/*the processing is done*/
processThreadStatus = false;

/*This is a utility method to reset the read counters everytime a network packet is
sent*/
private void resetReadCounter() {
    for(int i=0;i<dataBytestoSendcontainer.length;i++) {
        dataBytestoSendcontainer[i] = 0;
    }
}

/*method to specify if the video data is stored as int[]. If the data is stored as
int[], then the data will be converted to Bitmap and compressed to jpeg data
before sending to server. The only difference is calling this method will make the
dataFunnel to compress image against the main thread.*/
public static boolean isSpecialtreatmentForImage() {
    return isVideoDataInt;
}

/*method to determine the amout of bytes that can be sent in this
networkPacket*/
private void calculateDatatoSend(int storageavailable,int position) {
    if(dataBytesAvailabletoSend[position - 1] > 0) {
        if(dataBytesAvailabletoSend[position - 1] >
            DATA_TYPE_SIZE_LIMIT[position - 1]) {
            cbytestoRead = DATA_TYPE_SIZE_LIMIT[position - 1];
        } else {
cbyteoRead = dataBytesAvailabletoSend[position - 1];

nPacketSend.setDataTerminatedFlag(DATA_TYPE_KEY_ID[position - 1]);
EOF = true;
*
/*/assign the minimum of values*/
cbyteoRead = (cbyteoRead < storageavailable) 

cbyteoRead : storageavailable;

dataBytestoSendcontainer[position - 1] += cbyteoRead;
dataBytesAvailabletoSend[position - 1] -= cbyteoRead;
dataLength -= cbyteoRead;
bytesAvailable -= cbyteoRead;

nPacketSend.setDataTypeAvailableFlag(DATA_TYPE_KEY_ID[position - 1]);
}
}

/*get data from store to construct the networlPacket*/
private long getDataAvailabletoSend(){
    int dataSize = (int)dataLength;
    for(int i=0;i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++)
    {
        /*if the available data is empty then load data from the store*/
        if(dataBytesAvailabletoSend[i] == 0){
            dataBytesAvailabletoSend[i] =
            dPacketSend.getDataAvailabletoSend(i +1);
        }
        dataSize = dataSize + dataBytesAvailabletoSend[i];
    }
    return dataSize;
}

/*Method to return processed data from the incoming store*/
public byte[] getProcessedData(int mediaType){
    byte[] data = null;
    while(recvThreadStatus){
        synchronized(Receive_Thread_Lock){
            data = (byte[])storeRecv.getData(mediaType);
            if(data == null){
                TransportStore.log("log","DF -- data is null");
                try {
                    TransportStore.log("log","DF -- data is null..
waiting for data.");
                } catch (InterruptedException e) {

log("error","Error getting processed data.." + e.getMessage());
}

else{
    TransportStore.log("info",processedImageCounter + " -- DF -- getting processed image..." + mediaType + " size of data.. " + data.length);
    processedImageCounter++;
    return data;
}
}
}

return data;


/*Method to return processed data from the incoming store without reference to dataType*/
public DataContainer getProcessedData(){
    DataContainer data = null;
    while(recvThreadStatus){
        synchronized(Receive_Thread_Lock){
            data = (DataContainer)storeRecv.getData();
            if(data == null){
                TransportStore.log("log","DF -- data is null");
                try {
                    TransportStore.log("log","DF -- data is null..
waiting for data..");
                    Receive_Thread_Lock.wait();
                } catch (InterruptedException e) {
                    log("error","Error getting processed data.." + e.getMessage());
                }
            } else{
                TransportStore.log("log",processedImageCounter + " -- DF -- getting processed image...
" + data.getDataType() + " size of data.. " + data.getDataSize());
                processedImageCounter++;
                return data;
            }
        }
    }
    return data;
}

/*construct the packet with data */
private void writeDatatoPacket(){
    for(int i=0;i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++
    if(dataBytestoSendcontainer[i] > 0){}
log("log","Data requested for packet is .." +
dataBytestoSendcontainer[i]);
if(testMode){
    nPacketSend.write(dPacketSend.getDataPacket(i + 1, dataBytestoSendcontainer[i]),true);
} else {
    nPacketSend.write(dPacketSend.getDataPacket(i + 1, dataBytestoSendcontainer[i]));
}
}
 /*transport the network packet to the server*/
private void sendNetworkPacket(){
    byte[] networkpacket = nPacketSend.getNetworkPacket();
    transport.send(networkpacket);
}
 /*Stop processing images*/
public void stopSendProcessing(){
    processThreadStatus = false;
    sliceData = false;
    synchronized(storeSend){
        storeSend.notifyAll();
    }
}
 /*Stop Receive processing images*/
public void stopRecvProcessing(){
    TransportStore.log("info","DF -- stopping receive processing...");
    recvThreadStatus = false;
    synchronized(storeRecv){
        storeRecv.notifyAll();
    }
}
 /*The thread which starts processing. If you are trying to understand
the program start from here and follow the method calls.*/

@Override
public void run() {
    if(!serverMode){
        /*start the thread to create network packets*/
        createNetworkPackets();
    } else {
        /*Start the thread to process the packets and get the media*/
        processNetworkPacket();
    }
}
```java
/*utility method to print logs*/
public static void log(String name, String value) {
    if (!name.equalsIgnoreCase("log")) {
        Log.i("test", " ---- " + value);
        // System.out.println(" ---- " + value);
    }
}

/*method to deconstruct a network packet and get the media data back. The data is picked from the incoming store and the processed data is stored back in the incoming store.*/
public void processNetworkPacket() {
    byte[] data = null;
    long curTime = 0;
    int clientId = 0;

    /*127 is hardcoded. for now thats the maximum clients that can be supported*/
    /* The number of packets received per client*/
    int packetCounter[] = new int[Byte.MAX_VALUE];
    /* The time stamp for the client when the packet is received*/
    long timeCounter[] = new long[Byte.MAX_VALUE];

    while (recvThreadStatus) {
        curTime = System.currentTimeMillis();
        if ((data = storeRecv.getDataFromReceiveQueue()) != null) {
            /* get the packet from the receive queue*/
            nPacketRecv.readPacket(data);
            /* get the client Id from the packet*/
            clientId = nPacketRecv.getClientIDforPacket();
            /* process the data to get the media back*/
            for (int i = 0; i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++) {
                /* stage the data in the datapacket*/
                if (nPacketRecv.isDataAvailable(DATA_TYPE_KEY_ID[i]) == DATA_SWITCH_ON) {
                    processedDataRecv.stageDataPacket(nPacketRecv.getDataPacket(), i + 1, clientId);
                } else { /* if data is complete, move it to the store*/
                    if (nPacketRecv.isDataTerminated(DATA_TYPE_KEY_ID[i]) == DATA_SWITCH_ON) {
                        break;
                    } else {
                        break;
                    }
                }
            }
        }
    }
}
```

```java
byte[] imgData = processedDataRecv.getProcessedData(DATA_TYPE_KEY_ID[i], clientId);
storeRecv.addDatatoStore(DATA_TYPE_KEY_ID[i], imgData, clientId);

/*increment the value to the client
packetCounter[clientId]++;
TransportStore.log("log","$\text{DF -- Image Processed.. client is }" + clientId + " the size of data is }" + imgData.length + " Time taken is .." + (System.currentTimeMillis() - curTime) + "Time is ...");
System.currentTimeMillis();

/*if 10 packets are received for the client*/
if(packetCounter[clientId] == IConstants.DEFAULT_FRAME_RATE){
calculateFrameRate(clientId,timeCounter[clientId]);
packetCounter[clientId] = 0;
timeCounter[clientId] = System.currentTimeMillis();
}

/*Wait on the store till data is available*/
synchronized(Receive_Thread_Lock){
TransportStore.log("log","\text{DF -- I will wait for images}";
Receive_Thread_Lock.notifyAll();
}

try {
    Thread.sleep(3);
} catch (InterruptedException e)
    log("error","Interrupted

/*Wait on the store till data is available*/
synchronized(storeRecv){
```

TransportStore.log("log","DF -- process thread I will wait for images");
storeRecv.wait();
} catch (InterruptedException e) {
log("error","Exception in thread.."+
e.getMessage());
}
}
}
/*method to store the data in the receiving queue. The incoming store will be used*/
public void storeDataReceived(byte[] networkPacket){
if(serverMode){
synchronized(storeRecv){
/*add data to the receive queue*/
storeRecv.addDataToReceiveQueue(networkPacket);
if(!recvThreadStatus){
recvThreadStatus = true;
new Thread(this).start();
TransportStore.log("log","DF -- signalled to start reverse processing..");
}
storeRecv.notifyAll();
TransportStore.log("log","DF -- data inserted people notified..");
}
}

/*Method to find the rate at which video frames are received per client*/
private void calculateFrameRate(int clientId, long timeStamp){
/*if 10 packets are received for the client*/
long receiveRate = (System.currentTimeMillis() - timeStamp);
int frameRate = (int)Math.floor(10000/receiveRate);

}

TransportStore.java
package com.network.transport;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.util.Log;

/**
* @author Vidhya
*
* the primary controller class for the TransportStore. This class is responsible for providing a
* store for the client to deposit data, the transport to send the data to the server, Maintain the
* bandwidth share for different data types and ensure the communication with minimum latency
*
*/

public class TransportStore implements IConstants, Runnable {
    /* variable to hold the DataStore */
    private final OutgoingStore storeSend = OutgoingStore.getInstance();
    private final IncomingStore storeRecv = IncomingStore.getInstance();
    /* variable to store the transport */
    private Transport transport = null;
    /* local instance */
    private static TransportStore instance = null, recvInstance = null;
    /* variable to Data Packet */
    private final DataPacket dPacketSend = new DataPacket();
    private final DataPacket dPacketRecv = new DataPacket();
    /* variable to Data Packet */
    private final NetworkPacket nPacketSend = new NetworkPacket();
    private final NetworkPacket nPacketRecv = new NetworkPacket();
    /* variable to hold processed Image */
    private final ProcessedData processedDataRecv = new ProcessedData();
    /* array to hold bytes to read from each type */
    private int[] dataBytestoSendcontainer = new int[IConstants.DATA_TYPES_SUPPORTED_COUNT];
    private int[] dataBytesAvailabletoSend = new int[IConstants.DATA_TYPES_SUPPORTED_COUNT];
    /* instance of the ByteBufferOutputStream */
    private ByteBufferOutputStream byteBuffer = null;
    /* read pointers for command types */
    private int cbytestoRead = 0;
    private int bytesAvailable = 0;
    private long dataLength = 0;
    private boolean sliceData = false;
    private boolean processThreadStatus = false,
    testMode = false,
    recvThreadStatus = false;
    private boolean EOF = false;
    private static boolean isVideoDataInt = false;
}
private long currenttime = 0, imagepersecond = 0;
private int storeCounter = 0, processedImageCounter = 0;
private boolean serverMode = false;
private String Receive_Thread_Lock = "LOCK";

/* constructor to accept server and port. This will be used for sending */
private TransportStore(String server, int port) {
    transport = Transport.getInstance(server, port);
    byteBuffer = new ByteBufferOutputStream();
}

/* Default constructor. This can only be used in receive mode */
private TransportStore() {
    serverMode = true;
}

/* Default constructor. This can only be used in receive mode */
public TransportStore(boolean sMode) {
    serverMode = sMode;
}

/* the method will return the outputStream to compress the image. This is a special
Requirement for the Bitmap class in Android, which requires an OutputStream to
compress the image. */
public ByteBufferOutputStream getOutputStream() {
    return byteBuffer;
}

/* method to get instance of the class */
public static TransportStore getInstance(String serverIP, int port) {
    if (instance == null) {
        instance = new TransportStore(serverIP, port);
    }
    return instance;
}

/* method to get instance of the class */
public static TransportStore getInstance(String serverIP, int port, boolean
testMode) {
    if (instance == null) {
        instance = new TransportStore(serverIP, port);
    }
    instance.testMode = testMode;
    return instance;
}
/* method to get instance of the class */
public static TransportStore getInstance(boolean testMode) {
    if (recvInstance == null) {
        recvInstance = new TransportStore();
    }
    recvInstance.testMode = testMode;
    return recvInstance;
}

/*method for a client to deposit data*/
public void storeDatatosend(int mediaType, byte[] data) {
    if (!serverMode) {
        synchronized (storeSend) {
            storeSend.addDatatoStore(mediaType, data);
            if (!processThreadStatus) {
                processThreadStatus = true;
                sliceData = true;
                new Thread(this).start();
            }
        }
        storeCounter++;
    }
    storeSend.notifyAll();
}

/*Convenience method to store int[] in the store*/
public void storeDatatoSend(int mediaType, int[] data) {
    /*The flag is to enable special processing for image*/
    if (!serverMode) {
        isVideoDataInt = true;
        storeSend.addDatatoStore(mediaType, data);
        if (!processThreadStatus) {
            processThreadStatus = true;
            sliceData = true;
            new Thread(this).start();
        }
    }
}

/*method to create network packets. This method will be called from inside a thread*/
@SuppressWarnings("static-access")
public void createNetworkPackets() {
    int counter = 1;
    imagepersecond = currenttime = System.currentTimeMillis();
    while (sliceData) {
/*size of a network packet. will shrink when data is filled*/
bytesAvailable = IConstants.MAX_NETWORK_PACKET_SIZE;
/*The cumulative length of data available across all datatypes.
* Length of 0 signifies no data to send .*/
dataLength = getDataAvailabletoSend();
if(dataLength > 0) {
    while(sliceData == true && (bytesAvailable > 0 &&
dataLength > 0)) {
        /*use the following loop to get the data to be
send*/
        for(int i = 0; i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++) {
            calculateDatatoSend(bytesAvailable, i + 1);
        }
    }
    /*get the data from store and construct the network
packet*/
    writeDatatoPacket();
    /*counters are reset to make it available for next network
packet*/
    resetReadCounter();
    /*Send the network packet across the network*/
    sendNetworkPacket();
    /*EOF signifies the packet sent has a termination bit */
    if(EOF) {
        counter++;
        log("log", "TransportStore -- > Time taken for
one image " + (System.currentTimeMillis() - currenttime));
        currenttime = System.currentTimeMillis();
        EOF = false;
        if(counter == 11) {
            log("info", "TransportStore -- > Time
taken for 11 image " + (System.currentTimeMillis() - imagepersecond));
            imagepersecond = System.currentTimeMillis();
            counter = 1;
        }
    }
    try {
        /*Allow other threads to run*/
        Thread.currentThread().sleep(5);
    } catch (InterruptedException e) {
        log("error", "Exception in thread.." +
e.getMessage());
    }
}
else{
    try {
        /*Wait on the store till data is available*/
        synchronized(storeSend) {
            storeSend.wait();
        }
    } catch (InterruptedException e) {
        log("error","Exception in thread.." +
        e.getMessage());
    }
    log("log", "Waiting for data....");
}

/*Signal to stop process and disconnect from server*/
transport.stopTransport();
/*the processing is done*/
processThreadStatus = false;

/*This is a utility method to reset the read counters everytime a network packet is sent*/
private void resetReadCounter() {
    for(int i = 0;i < dataBytestoSendcontainer.length;i++) {
        dataBytestoSendcontainer[i] = 0;
    }
}

/*method to specify if the video data is stored as int[]. If the data is stored as int[], then the data will be converted to Bitmap and compressed to jpeg data before sending to server. The only difference is calling this method will make the dataFunnel to compress image against the main thread.*/
public static boolean isSpecialtreatmentForImage() {
    return isVideoDataInt;
}

/*method to determine the amount of bytes that can be sent in this networkPacket*/
private void calculateDatatoSend(int storageavailable, int position) {
    if(dataBytesAvailabletoSend[position - 1] > 0) {
        if(dataBytesAvailabletoSend[position - 1] > DATA_TYPE_SIZE_LIMIT[position - 1]) {
            cbytestoRead = DATA_TYPE_SIZE_LIMIT[position - 1];
        }
    }
}
else {
    cbytestoRead = dataBytesAvailabletoSend[position - 1];

    nPacketSend.setDataTerminatedFlag(DATA_TYPE_KEY_ID[position - 1]);
    EOF = true;
}

/*assign the minimum of values*/
cbytestoRead = (cbytestoRead < storageavailable) ?
    cbytestoRead : storageavailable;
    dataBytestoSendcontainer[position - 1] += cbytestoRead;
    dataBytesAvailabletoSend[position - 1] -= cbytestoRead;
    dataLength -= cbytestoRead;
    bytesAvailable -= cbytestoRead;

    nPacketSend.setDataTypeAvailableFlag(DATA_TYPE_KEY_ID[position - 1]);
}

/*get data from store to construct the networlPacket*/
private long getDataAvailabletoSend(){
    int dataSize = (int)dataLength;
    for(int i=0;i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++){
        /*if the available data is empty then load data from the store*/
        if(dataBytesAvailabletoSend[i] == 0){
            dataBytesAvailabletoSend[i] = 
            dPacketSend.getDataAvailabletoSend(i +1);
            dataSize = dataSize + dataBytesAvailabletoSend[i];
        }
    }
    return dataSize;
}

/*Method to return processed data from the incoming store*/
public byte[] getProcessedData(int mediaType){
    byte[] data = null;
    while(recvThreadStatus){
        synchronized(Receive_Thread_Lock){
            data = (byte[])storeRecv.getData(mediaType);
            if(data == null){
                TransportStore.log("log","DF -- data is null");
                try {
                    TransportStore.log("log","DF -- data is null..
                    waiting for data..");
                    Receive_Thread_Lock.wait();
                } catch (InterruptedException e) { 

log("error","Error getting processed data.. " +
e.getMessage());
}

return data;
}

public DataContainer getProcessedData()
{
    DataContainer data = null;
    while(recvThreadStatus)
    {
        synchronized(Receive_Thread_Lock){
            data = (DataContainer)storeRecv.getData();
            if(data == null){
                TransportStore.log("log","DF -- data is null");
                try {
                    Receive_Thread_Lock.wait();
                } catch (InterruptedException e) {
                    log("error","Error getting processed data.. " +
e.getMessage());
                }
            } else{
                TransportStore.log("log",processedImageCounter + " --> DF -- getting processed image..." + mediaType + " size of data.. " + data.length);
                processedImageCounter++;
                return data;
            }
        }
    }
    return data;

    /*Method to return processed data from the incoming store without reference to
dataType*/
    public DataContainer getProcessedData()
    {
        DataContainer data = null;
        while(recvThreadStatus)
        {
            synchronized(Receive_Thread_Lock){
                data = (DataContainer)storeRecv.getData();
                if(data == null){
                    TransportStore.log("log","DF -- data is null");
                    try {
                        Receive_Thread_Lock.wait();
                    } catch (InterruptedException e) {
                        log("error","Error getting processed data.. " +
e.getMessage());
                    }
                } else{
                    TransportStore.log("log",processedImageCounter + " --> DF -- getting processed image..." + data.getDataType() + " size of data.. " +
data.getDataSize());
                    processedImageCounter++;
                    return data;
                }
            }
        }
        return data;
    }

    /*construct the packet with data */
    private void writeDatatoPacket()
    {
        for(int i=0;i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++)
        {
            if(dataBytestoSendcontainer[i] > 0){
log("log","Data requested for packet is .." + 
    dataBytestoSendcontainer[i]);
if(testMode){
    nPacketSend.write(dPacketSend.getDataPacket(i 
    + 1, dataBytestoSendcontainer[i]),true);
} else {
    nPacketSend.write(dPacketSend.getDataPacket(i 
    + 1, dataBytestoSendcontainer[i]));
}

/*transport the network packet to the server*/
private void sendNetworkPacket(){
    byte[] networkpacket = nPacketSend.getNetworkPacket();
    transport.send(networkpacket);
}

/*Stop processing images*/
public void stopSendProcessing(){
    processThreadStatus = false;
    sliceData = false;
    synchronized(storeSend){
        storeSend.notifyAll();
    }
}

/*Stop Receive processing images*/
public void stopRecvProcessing(){
    TransportStore.log("info","DF -- stopping receive processing...");
    recvThreadStatus = false;
    synchronized(storeRecv){
        storeRecv.notifyAll();
    }
}

/*The thread which starts processing. If you are trying to understand
the program start from here and follow the method calls.*/

@Override
public void run() {
    if(!serverMode){
    /*start the thread to create network packets*/
    createNetworkPackets();
    } else {
    /*Start the thread to process the packets and get the media*/
    processNetworkPacket();
}
utility method to print logs*/
public static void log(String name, String value) {
    if (!name.equalsIgnoreCase("log")) {
        Log.i("test", " ---- " + value);
        //System.out.println(" ---- " + value);
    }
}

/*method to deconstruct a network packet and get the media data back. The data is picked from the
* incoming store and the processed data is stored back in the incoming store.*/
public void processNetworkPacket() {
    byte[] data = null;
    long curTime = 0;
    int clientId = 0;

    /*127 is hardcoded. for now thats the maximum clients that can be supported*/
    /*The number of packets received per client*/
    int packetCounter[] = new int[Byte.MAX_VALUE];
    /*The time stamp for the client when the packet is received*/
    long timeCounter[] = new long[Byte.MAX_VALUE];

    while (recvThreadStatus) {
        curTime = System.currentTimeMillis();
        if ((data = storeRecv.getDataFromReceiveQueue()) != null) {
            /*get the packet from the receive queue*/
            nPacketRecv.readPacket(data);
            /*get the client Id from the packet*/
            clientId = nPacketRecv.getClientIDforPacket();
            /*process the data to get the media back*/
            for (int i = 0; i < IConstants.DATA_TYPES_SUPPORTED_COUNT; i++) {
                /*stage the data in the datapacket*/
                if (nPacketRecv.isDataAvailable(DATA_TYPE_KEY_ID[i]) ==
                    DATA_SWITCH_ON) {
                    processedDataRecv.stageDataPacket(nPacketRecv.getDataPacket(), i + 1, clientId);
                } /*if data is complete , move it to the store*/
        }
    }
}
if(nPacketRecv.isDataTerminated(DATA_TYPE_KEY_ID[i]) == 
DATA_SWITCH_ON){
    byte[] imgData = 
processedDataRecv.getProcessedData(DATA_TYPE_KEY_ID[i],clientId);

    storeRecv.addDatatoStore(DATA_TYPE_KEY_ID[i], imgData,clientId);

    /*incriment the value to the client
     *counter*/
    packetCounter[clientId]++;

    TransportStore.log("log","$STITTTTTTTTTT
DF -- Image Processed.. client is " + clientId + " the size of data is.." + imgData.length +" Time taken is .." + (System.currentTimeMillis() - curTime) + "Time is ..."+
System.currentTimeMillis());

    /*if 10 packets are received for the
client*/
    if(packetCounter[clientId] ==
IConstants.DEFAULT_FRAME_RATE){
        calculateFrameRate(clientId,timeCounter[clientId]);
        packetCounter[clientId] = 0;
        timeCounter[clientId] =
System.currentTimeMillis();
    }

    /*Wait on the store till data is
available*/
    synchronized(Receive_Thread_Lock){
        TransportStore.log("log","DF --
I will wait for images");

        Receive_Thread_Lock.notifyAll();

        try {
            Thread.sleep(3);
            } catch (InterruptedException e)
{
            log("error","Interrupted
while sleeping in processnetworkPacket..");

            }
        }
    }
} else {
try {
    /*Wait on the store till data is available*/
    synchronized(storeRecv) {
        TransportStore.log("log","DF -- process thread I will wait for images");
        storeRecv.wait();
    }
} catch (InterruptedException e) {
    log("error","Exception in thread.." + e.getMessage());
}

/*method to store the data in the receiving queue. The incoming store will be used*/
public void storeDataReceived(byte[] networkPacket) {
    if (serverMode) {
        synchronized(storeRecv) {
            /*add data to the receive queue*/
            storeRecv.addDatatoReceiveQueue(networkPacket);
            if (!recvThreadStatus) {
                recvThreadStatus = true;
                new Thread(this).start();
                TransportStore.log("log","DF -- signalled to start reverse processing..");
            }
        }
        storeRecv.notifyAll();
        TransportStore.log("log","DF -- data inserted people notified..");
    }
}

/*Method to find the rate at which video frames are received per client*/
private void calculateFrameRate(int clientId, long timeStamp) {
    /*if 10 packets are received for the client*/
    long receiveRate = (System.currentTimeMillis() - timeStamp) / 10;
    int frameRate = (int) Math.floor(10000 / receiveRate);
}

Datapacket.java

package com.network.transport;
import java.io.BufferedOutputStream;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import android.graphics.Bitmap;
import android.graphics.Bitmap.CompressFormat;
import android.graphics.Bitmap.Config;
import android.os.Environment;
import android.util.Log;

/**
 * @author Vidhya
 *
 * This class represents a dataPacket. A dataPacket represents data of one type. For example
 * a video datapacket will contain bytes which represent a video. The data contained in a packet
 * may be partial. Multiple datapackets will be combined to for meaningful information. All datapackets
 * are proceeded by a length header which is of 2 bytes (1 short).
 * The dataPacket is represented by a byteBufff container.
 */

public class DataPacket implements IDataPacket {
    /*byteBuffer which represents a dataPacket*/
    private ByteBuffer dataPacket = null;
    /*Array of byteBuffers to store the data from the store*/
    private ByteBuffer[] dataAvailable = new ByteBuffer[Constants.DATA_TYPES_SUPPORTED_COUNT];
    private ByteBuffer[] dataReceived = new ByteBuffer[Constants.DATA_TYPES_SUPPORTED_COUNT];
    /*The header which contains the length of bytes in this packet*/
    private short dataLength = 0;
    /*variable to hold the DataStore*/
    private final OutgoingStore storeSend = OutgoingStore.getInstance();
    private final IncomingStore storeRecv = IncomingStore.getInstance();
    /*local instance variable*/
    private static DataPacket instance = null;
    /*instance of byteBufferoutputStream*/
    private ByteBufferOutputStream byteBuffer = new ByteBufferOutputStream();

    protected DataPacket(){
        dataPacket = ByteBuffer.allocate(Constants.MAX_NETWORK_PACKET_SIZE);
    }

    /*singleton class to get the instance of the class*/
    public static DataPacket getInstance(){
        if (instance == null) {
            instance = new DataPacket();
        }
        return instance;
    }
}
if(instance == null){
    instance = new DataPacket();
}
return instance;
}

@Override
public short getDataLength() {
    dataLength = (short)dataPacket.capacity();
    return ((short)(dataLength));
}

@Override
public void setDataLength(short dataLength) {
}

/*@method to write data to the packet*/
public void write(byte[] data){
    dataLength = (short)data.length;
    /*the header id added*/
    dataPacket.putShort(dataLength);
    /*the data is added.*/
    dataPacket.put(data,0,data.length);
}

/*@method to get the data packet*/
public byte[] getDataPacket(int dataType,int length){
    byte[] dst = null;
    /*if we need the full buffer, which will be a entire data*/
    if(length == -1){
        int size = dataAvailable[dataType - 1].capacity();
        dst = new byte[size];
        dataAvailable[dataType - 1].get(dst, 0, size);
        return dst;
    }
    /*process partial data based on length*/
    int available = 0;
    if(dataAvailable[dataType - 1] != null){
        available = dataAvailable[dataType - 1].remaining();
    }
    if(length > available){
        length = available;
    }
    dst = new byte[length];
    if(length > 0){
        dataAvailable[dataType - 1].get(dst, 0, length);
TransportStore.log("log", "Returning data for .. for id " + (dataType) + 
is " + length + " remaining data is .." + dataAvailable[dataType - 1].remaining());
return dst;

@Override
public int getDataAvailabletoSend(int dataType) {
    int dataAvailabletoSend = 0;
    /*load data if required*/
    loadDataFromStore(dataType);
    if(dataType > 0 && dataAvailable[dataType - 1] != null) {
        dataAvailabletoSend = dataAvailable[dataType - 1].remaining();
        TransportStore.log("log", "Data available to send .. for id " +
dataType + " is " + dataAvailabletoSend);
    }
    return dataAvailabletoSend;
}

/*Load the data from the store for each type*/
private void loadDataFromStore(int mediaType) {
    byte[] data = null;
    if((IConstants.DATA_TYPE_VIDEO == mediaType) &&
TransportStore.isSpecialtreatmentForImage()) {
        data = getCompressedJPEGusingStream((int[]) storeSend.getData(mediaType));
    } else {
        data = (byte[]) storeSend.getData(mediaType);
    }
    if(dataAvailable[mediaType - 1] == null && data != null) {
        dataAvailable[mediaType - 1] = ByteBuffer.wrap(data);
        TransportStore.log("log", "loading data to buffer ..
remaining data is " + dataAvailable[mediaType - 1].remaining());
    } else if(dataAvailable[mediaType - 1] != null &&
dataAvailable[mediaType - 1].remaining() == 0 && data != null) {
        dataAvailable[mediaType - 1].clear();
        dataAvailable[mediaType - 1] = ByteBuffer.wrap(data);
        TransportStore.log("log", "loading data to buffer
afterreaching limit .. remaining data is" + dataAvailable[mediaType - 1].remaining());
    }
}

/*Method to use the byteBufferOutputStream for data staging*/
private byte[] getCompressedJPEGusingStream(int[] unprocessedData) {
    if(unprocessedData == null) {
        return null;
    }
Bitmap bitmap = Bitmap.createBitmap(unprocessedData,
IConstants.IMAGE_CROP_WIDTH,IConstants.IMAGE_CROP_HEIGHT,
Config.ARGB_8888);
bitmap.compress(CompressFormat.JPEG,
IConstants.IMAGE_QUALITY, byteBuffer);
writetoFile(byteBuffer.getArray());
return byteBuffer.getArray();

/*function to test the image created. Will only work with Android device*/
private void writetoFile(byte[] data)
{
try {
    File root = Environment.getExternalStorageDirectory();
    if (root.canWrite()){
        File gpxfile = new File(root, "image.jpg");
        FileOutputStream gpxwriter = new
FileOutputStream(gpxfile);
        BufferedOutputStream out = new
BufferedOutputStream(gpxwriter);
        out.write(data);
        out.flush();
        out.close();
    }
} catch (IOException e) {
    Log.e("test", "Could not write file " + e.getMessage());
}
}

NetworkPacket.java

package com.network.transport;

import java.nio.ByteBuffer;

/**
 * @author Vidhya
 *
 * This class represents a network Packet. It implements the INetworkPacket interface. It
 * provides
 * operations to create and read a networkPacket. The major container in this class is a
 * bytebuffer
 * which stores the content to be sent across the network as a packet
 */
public class NetworkPacket implements INetworkPacket {
/*The ByteBuffer which acts as a container for data*/
private ByteBuffer networkPacket = null;
/*array of bytes which signify the data header bits*/
private StringBuffer header = new StringBuffer("00000000");
/*array of bytes which signify the data termination bits*/
private StringBuffer dataTerminator = new StringBuffer("00000000");
/*local instance variable*/
private static NetworkPacket instance = null;
/*convenience variable to check the packet structure*/
private String pStructure = "";
/*variable to store the header status*/
private boolean headerCreated = false;
/*variable to notify the length of the network packet. default is to 2 to signify the
length bytes*/
private int packetLength = 4;
/*Short variable(2 bytes) to signify the first two bytes which hold the length
value*/
private short nLengthBytes = 0;
/*boolean to set if the transmission is a test*/
private boolean testMode = false;
/*identifier for the client*/
private int clientIdentifier = -1;

/*private constructor to instantiate the buffer*/
protected NetworkPacket() {
    /*The 12 bytes are for the header and end bits and length header*/
    networkPacket = ByteBuffer.allocate(IConstants.MAX_NETWORK_PACKET_SIZE + 12);
}

/*singleton class to get the instance of the class*/
public static NetworkPacket getInstance() {
    if (instance == null) {
        instance = new NetworkPacket();
    }
    return instance;
}

@Override
public char isDataAvailable(int datatypeConstant) {

    return this.header.charAt((Byte.SIZE - datatypeConstant));
}

@Override
public char isDataTerminated(int datatypeConstant) {

    return this.dataTerminator.charAt((Byte.SIZE - datatypeConstant));
}
```java
@Override
public void setDataTerminatedFlag(int dataType) {
    this.dataTerminator.setCharAt((Byte.SIZE - dataType), IConstants.DATA_SWITCH_ON);
}

@Override
public void setDataTypeAvailableFlag(int dataType) {
    this.header.setCharAt((Byte.SIZE - dataType), IConstants.DATA_SWITCH_ON);
}

public void write(byte[] dataPacket) {
    packetLength = packetLength + (dataPacket.length + 2);
    if (!headerCreated) {
        headerCreated = true;
        /*The header is added*/
        networkPacket.putShort(nLengthBytes);
        byte bHeader = createHeader();
        networkPacket.put(bHeader);
        pStructure = pStructure + nLengthBytes + ".... " + bHeader + "-->" + header.toString();
    }
    byte bTerminator = createDataTerminationByte();
    networkPacket.put(bTerminator);
    pStructure = pStructure + bTerminator + "-->" + dataTerminator.toString();
    if (networkPacket != null && networkPacket.remaining() > 0) {
        networkPacket.putShort((short)(dataPacket.length));
        pStructure = pStructure + dataPacket.length + "data ....";
        //TransportStore.log("Network packet", "after writing length .. length of data packet " + dataPacket.length + " space remaining in network Packet " + networkPacket.remaining());
        networkPacket.put(dataPacket);
        //TransportStore.log("Network packet", "After writing data.. remaining space " + networkPacket.remaining());
    }
    //method to write data to network packet. This method takes a datapacket
    It writes the length header before the data is added */
```
It writes the length header before the data is added. The overloaded version is used to pass the image without header and other attributes. */

```java
public void write(byte[] dataPacket, boolean test) {
    //TransportStore.log("Network packet", "about to create packet");
    this.testMode = test;
    if(!testMode){
        write(dataPacket);
        return;
    }
    packetLength = dataPacket.length;
    if(!headerCreated){
        headerCreated = true;
        /*The header is added*/
        //networkPacket.putShort(nLengthBytes);
        byte bHeader = createHeader();
        //networkPacket.put(bHeader);
        pStructure = pStructure + " " + bHeader + "---->" + header.toString();
        /*Data termination is added*/
        byte bTerminator = createDataTerminationByte();
        //networkPacket.put(bTerminator);
        pStructure = pStructure + " " + bTerminator + "---->" + dataTerminator.toString();
    }
    if(networkPacket != null && networkPacket.remaining() > 0){
        //networkPacket.putShort((short)(dataPacket.length));
        pStructure = pStructure + " " + (dataPacket.length);
        pStructure = pStructure + " " + "data ...."
        //TransportStore.log("Network packet", "after writing length .. length of data packet " + dataPacket.length + " space remaining in network Packet " + networkPacket.remaining());
        networkPacket.put(dataPacket);
        //TransportStore.log("Network packet", "After writing data.. remaining space " + networkPacket.remaining());
    }
}
```

/*method to get the byte array which should should be sent to the network*/

```java
public byte[] getNetworkPacket() {
    byte[] dst = new byte[packetLength];
    nLengthBytes = (short)(packetLength - 2);
    networkPacket.putShort(0, nLengthBytes);
    byte[] trimmedArray = networkPacket.array();
    if(packetLength < networkPacket.capacity()){
```
System.arraycopy(trimmedArray, 0, dst, 0, packetLength);

} else {
    dst = trimmedArray;
}

pStructure = " " + nLengthBytes + ".... " + pStructure;
TransportStore.log("log", "Packet structure " + pStructure);

networkPacket.clear();

packetLength = 4;
headerCreated = false;
/*reset the header*/
header.replace(0, header.length(), "00000000");
/*reset the terminator*/
dataTerminator.replace(0, header.length(), "00000000");
pStructure = "";

return dst;

/*method to create header*/

private byte createHeader() {
    return Byte.parseByte(header.toString(), 2);
}

/*method to create data termination header*/

private byte createDataTerminationByte() {
    return Byte.parseByte(dataTerminator.toString(), 2);
}

/*method to create a header string from byte value*/

private void createHeaderFromByte(byte byteHeader) {
    /*reset the header*/
    header.replace(0, header.length(), "00000000");

    if (byteHeader > 0) {
        String binaryHeader = Integer.toString(byteHeader);
        TransportStore.log("log", "the header byte string is.. " + binaryHeader);

        int index = 0;
        for (int i = Byte.SIZE - binaryHeader.length(); i < header.length(); i++) {
            header.setCharAt(i, (binaryHeader.charAt(index)));
            index++;
        }
    }
}

/*method to create a termination byte string from byte value*/

private void createTerminatorFromByte(byte terminator) {
    /*reset the terminator*/
dataTerminator.replace(0, dataTerminator.length(),"00000000");

    if(terminator >0) {
        String binaryTerm = Integer.toBinaryString(terminator);
        TransportStore.log("log","the terminator byte string is.. " + binaryTerm);
        int index=0;
        for(int i=(Byte.SIZE - binaryTerm.length());i<dataTerminator.length();i++) {
            dataTerminator.setCharAt(i, (binaryTerm.charAt(index)));
            index++;
        }
    }

    /*method to construct a networkPacket from a byteArray*/
    public void readPacket(byte[] nPacket) {
        networkPacket = ByteBuffer.wrap(nPacket);
        /*get the client identifier from the packet*/
        clientIdentifier = networkPacket.get();
        /*get the header from the packet*/
        createHeaderFromByte(networkPacket.get());
        pStructure = pStructure + header.toString();
        /*get the termination byte from the packet*/
        createTerminatorFromByte(networkPacket.get());
        pStructure = pStructure + " " + dataTerminator.toString();
        pStructure = pStructure + " " + networkPacket.remaining() + ".. current position.." + networkPacket.position() + " .....data";
        TransportStore.log("log", "NetworkPacket --- received data...structure...
" + pStructure);
        pStructure = "";
    }

    /*method to get a dataPacket from the networkPacket*/
    public byte[] getDataPacket() {
        /*get the length of packet*/
        short dPacketLength = networkPacket.getShort();
        byte[] dPacketData = new byte[dPacketLength];
        TransportStore.log("log", "current pos.." + networkPacket.position() + "..remaining.." + networkPacket.remaining());
        networkPacket.get(dPacketData, 0, dPacketLength);
        return dPacketData;
    }

    /*method to get the clientId for the networkPacket*/
    public int get CIDforPacket() {
        return clientIdentifier;
    }
package com.network.transport;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.net.SocketAddress;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.channels.DatagramChannel;
import java.util.Date;

/**
* @author Vidhya
*
* The following class is used to transport data. This abstracts the details of the actual
* network connection and the mechanism. The current implementation this class is a TCP socket.
* This will be enhanced later to support UDP or HTTP connections
*/
public class Transport implements IConstants, Runnable {

    /*static instance of the class*/
    private static Transport instance = null;
    /*Socket to connect to server*/
    private Socket serverSocket = null;
    /*variable to stop connecting to the server*/
    private boolean sendData = false;
    /*The outputStream of the Socket*/
    private DataOutputStream out = null;
    /*The inputstream to read data from socket*/
    private DataInputStream in = null;
    /*variable to point to the counter for read.*/
    private int rCounter = 0;

    }
private final OutgoingStore outStore = OutgoingStore.getInstance();
private final IncomingStore inStore = IncomingStore.getInstance();
private long sleepTime = 20;
private String serverIP = null;
private int serverPort = 0;
private DatagramChannel channel = null;
private SocketAddress server = null;
private int server_type = 2; //1 - TCP, 2 UDP

private Transport(String ip, int port) {
    serverIP = ip;
    serverPort = port;
}

public static Transport getInstance(String serverIP, int port) {
    if (instance == null) {
        instance = new Transport(serverIP, port);
    }
    return instance;
}

public void send(byte[] networkPacket) {
    outStore.addDatatoSend(networkPacket);
    if (!sendData) {
        startTransport();
    }
    sendData(outStore.getDatatoSend());
    sendDataUDP(outStore.getDatatoSend());
}

/* the run method of the thread which will send the data to the server*/
public void run() {
    //for later use
}

/*non thread implementation to send the data to server*/
private void sendData(byte[] networkPacket) {
    byte[] data = networkPacket;
    if(data != null){
        try {
            out.write(data);
            out.flush();
            TransportStore.log("log", "Transport --- Sending data to server .. size is "+ data.length);
        } catch (IOException e) {
            TransportStore.log("error", e.getMessage());
        }
    }
}

/*non thread implementation to receive data from server*/
private void receiveData() {
    byte[] data = null;
    short length = 0;
    /*read the first 2 bytes to get the length of packet*/
    try {
        length = in.readShort();
        in.read(data, 0, length);
        inStore.addDatatoReceiveQueue(data);
    } catch (IOException e) {
        TransportStore.log("error", e.getMessage());
    }
}

/*private method to connect to server*/
private void connect() {
    try {
        if(server_type == 1) {
            serverSocket = new Socket(InetAddress.getByName(this.serverIP), this.serverPort);
            out = new DataOutputStream(new BufferedOutputStream(serverSocket.getOutputStream(),4096));
            in = new DataInputStream(new BufferedInputStream(serverSocket.getInputStream(),4096));
            sendData = true;
            TransportStore.log("info", "About to connect to server.");
        } else if(server_type == 2) {
            //connect to UDP server
            channel = DatagramChannel.open();
            SocketAddress address = new InetSocketAddress(0);
            Socket( InetAddress.getByName(this.serverIP), this.serverPort);
            out = new DataOutputStream(new BufferedOutputStream(serverSocket.getOutputStream(),4096));
            in = new DataInputStream(new BufferedInputStream(serverSocket.getInputStream(),4096));
            sendData = true;
        }
    } else if(server_type == 2) {
        //connect to UDP server
        channel = DatagramChannel.open();
        SocketAddress address = new InetSocketAddress(0);
DatagramSocket socket = channel.socket();
socket.setSoTimeout(5000);
socket.bind(address);
server = new InetSocketAddress(this.serverIP,
this.serverPort);
}
TransportStore.log("info", "connected to server..");

} catch (UnknownHostException e) {
    TransportStore.log("error", e.getMessage());
e.printStackTrace();
} catch (IOException e) {
    TransportStore.log("error", e.getMessage());
e.printStackTrace();
}

} /*disconnect from server*/
private void disconnect() {
    try {
        if (server_type == 1) {
            out.close();
in.close();
socket.close();
} else if (server_type == 2) {
            channel.close();

        }
        TransportStore.log("info", "Disconnected from
server..");
    } catch (IOException e) {
        TransportStore.log("error", e.getMessage());
    }
}

/*Request to stop sending packets*/
public void stopTransport() {
    TransportStore.log("info", "signal to disconnect..");
    if (sendData) {
        sendData = false;
        disconnect();
    }
}

/*Request to start sending packets*/
public void startTransport() {

connect();
sendData = true;

/*send the data as a UDPPacket*/
private void sendDataUDP(byte[] nPacket) {
    try {
        ByteBuffer buffer = ByteBuffer.wrap(nPacket);
        // time protocol always uses big-endian order
        TransportStore.log("test", "Sending packet to server of size " + nPacket.length + ".");
        channel.send(buffer, server);
    } catch (Exception ex) {
        System.err.println(ex);
        ex.printStackTrace();
    }
}

/*Convenience method to see what bytes are sent to server*/
private void writetoFile(byte[] networkpacket) {
    try {
        File targetFile = new File("c:\temp\rnd\images\transport_network.jpg");
        FileOutputStream fos = new FileOutputStream(targetFile,true);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        bos.write(networkpacket,0,networkpacket.length);
        bos.flush();
        fos.close();
        bos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

ProcessedData.java

package com.network.transport;
import java.nio.ByteBuffer;
import java.util.HashMap;

/***
This class acts as a wrapper for processed Data. It is primarily used when network packets are reconstructed to get the processed image. It provides a staging area where partially constructed data packets are stored. It has the capability to support multiple clients with a single instance.

```java
public class ProcessedData {
    private ByteBuffer[] dataReceived = null;
    private HashMap<Integer, ByteBuffer[]> clientMediaContainer = new HashMap<Integer, ByteBuffer[]>();

    protected ProcessedData() {
    }

    public void stageDataPacket(byte[] partialMediaData, int mediaType, int clientIndex) {
        if(isClientInMap(clientIndex)) {
            dataReceived = clientMediaContainer.get(new Integer(clientIndex));
        } else {
            dataReceived = new ByteBuffer[IConstants.DATA_TYPES_SUPPORTED_COUNT];
        }

        if(dataReceived[mediaType - 1] == null) {
            dataReceived[mediaType - 1] = ByteBuffer.allocate(IConstants.MAX_MEDIA_DATA_SIZE);
        }

        if(dataReceived[mediaType - 1].remaining() >= partialMediaData.length) {
            dataReceived[mediaType - 1].put(partialMediaData);
        } else {
            // this is not a place to be. The media is larger than 8K. not a good sign
            TransportStore.log("log", "DataPacket --> increasing size of incoming buffer..I might hit memory issue");
            int oldCapacity = dataReceived[mediaType - 1].capacity();
        }
    }
}
```
int newDataSize = partialMediaData.length;
ByteBuffer newBuffer = ByteBuffer.allocate(oldCapacity +
newDataSize);

/*first add the byte buffer*/
newBuffer.put(dataReceived[mediaType - 1].array());
/*add the new data*/
newBuffer.put(partialMediaData);
/*replace the old buffer with the new one*/
dataReceived[mediaType - 1] = newBuffer;
}
/*Add the modified array to the map*/
clientMediaContainer.put(new Integer(clientIndex), dataReceived);
}

/*load the media data to the store*/
public byte[] getProcessedData(int mediaType, int clientIndex){
    dataReceived = clientMediaContainer.get(new Integer(clientIndex));
    byte[] mediaData = null;
    if(dataReceived != null){
        /*transfer the data from the staging buffers to the store*/
        int mediaLength = dataReceived[mediaType - 1].position();
        /*place the read pointer at the beginning of the buffer*/
        dataReceived[mediaType - 1].position(0);
        mediaData = new byte[mediaLength];
        /*Transfer the contents to the new array*/
        dataReceived[mediaType - 1].get(mediaData, 0, mediaLength);
        /*Make the buffer ready for new operation*/
        dataReceived[mediaType - 1].clear();
    }
    return mediaData;
}

/*
 * method to find if the clientId is present in the map
 */
private boolean isClientinMap(int clientId){
    return clientMediaContainer.containsKey(new Integer(clientId));
}
BIBLIOGRAPHY

[1] Introduction to Mobile technology and communications, [Online]

Available: http://qikapi.pbworks.com/


[7] Bill Evjen, Scott Hanselman, Devin Rader, “Professional ASP.NET 3.5 in C# and VB.”


Available:

http://build.xuggle.com/job/xuggle_java_xuggler_red5_jdk6_i386_ubuntu/javadoc/java/api/index.html


Available:

http://www.inb.uni-luebeck.de/~boehme/using_libavcodec.html