2011

Live Video Streaming from Android-Enabled Devices to Web Browsers

Justin M. Bailey
University of South Florida, jbailey5@mail.usf.edu

Live Video Streaming from Android-Enabled Devices to Web Browsers

by

Justin Bailey

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Adriana Iamnitchi, Ph.D.
Miguel Labrador, Ph.D.
Robert Lawrence, M.F.A.

Date of Approval:
October 19, 2011

Keywords: video transcoding, Android application, h263, rtsp, rtp

Table of Contents

List of Figures iii

Abstract iv

Chapter 1: Introduction 1
1.1. Contributions 2
1.2. Thesis Roadmap 2

Chapter 2: Architecture 5
2.1. Existing Components 6
2.1.1. H.263 7
2.1.2. Real-Time Transport Protocol 7
2.1.3. Real Time Streaming Protocol 9
2.1.4. Encoding and Decoding 10
2.1.5. Android Platform 11
2.2. Contributed Components 12
2.2.1. Video Streaming 12
2.2.2. Web Interface and User Management 12

Chapter 3: Video Streaming 14
3.1. Server Side 14
3.1.1. Android Device Communications Server 14
3.1.2. Real Time Streaming Protocol Server 16
3.1.3. Real-Time Transfer Protocol Server 16
3.2. Android Device Side 17
3.2.1. Conversion H.263 to RTP H.263 17
3.2.2. Communication 18

Chapter 4: Web Interface and User Management 19
4.1. Conversion to Flash Video 19
4.2. Flash Player 20
4.3. User Accounts and MySQL Database 21

Chapter 5: Related Work 23
5.1. Free Commercial Web Video Streaming Services 23
5.2. Other Commercial Solutions 24
5.3. Streaming Protocols 25

Chapter 6: Experimental Evaluation 27
6.1. Experimental Setup 27
6.2. Experimental Results 29
6.2.1. Encoder and Server Tests 30
6.2.1.1. Picture Quality 31
6.2.1.2. CPU and Memory Usage 32
6.2.1.3. Bandwidth 34
6.2.1.4. Scalability 34
6.2.2. Android Device 35
6.2.2.1. Bandwidth 36
6.2.2.2. Battery and CPU Usage 36
6.2.2.3. Delay 37

Chapter 7: Conclusions and Future Work 38
7.1. Future Directions 38
7.2. Final Thoughts 40

References 41

List of Figures

Figure 1: Architecture Overview 6

Figure 2: RTP Header 9

Figure 3: Picture Quality Reference 32

Figure 4: CPU Consumption by the Encoders and the Server with a Pentium D at 3.4 Ghz. 33

Figure 5: Memory Consumption by the Encoders and the Server. 33

Figure 6: Bandwidth from Server to Web Browser 34

Figure 7: CPU Usage by Server with 5 Users 35

Figure 8: Memory Usage by Server with 5 Users 35

Figure 9: RTP Bandwidth from Android Device to Server. 36

Abstract

The wide-spread adoption of camera-embedded mobile devices along with the ubiquitous connection via WiFi or cellular networks enables people to visually report live events. Current solutions limit the configurability of such services by allowing video streaming only to fixed servers. In addition, the business models of the companies that provide such (free) services insert visual ads in the streamed videos, leading to unnecessary resource consumption.

This thesis proposes an architecture of a real-time video streaming service from an Android mobile device to a server of the user's choice. The real-time video can then be viewed from a web browser. The project builds on open-source code and open protocols to implement a set of software components that successfully stream live video.

Experimental evaluations show practical resource consumption and a good quality of the streamed video. Furthermore, the architecture is scalable and can support large number of simultaneous streams with additional increase in hardware resources.

Chapter 1: Introduction

The internet combined with smart phones and its peripherals open the doors to limitless mobile possibilities. One of these possibilities is explored and exploited by capturing video on a mobile device, in real-time, and transferring to a web page, viewable by the entire world.

Currently, streaming live video from a mobile phone is limited to commercial products such as Bambuser and Qik, while an open source solution has not yet surfaced. Open source lets developers and users view the source code free of charge and create their own custom modifications.

This project develops an open source solution capable of transferring the live video with little overhead on the phone and/or server. Users will have the ability to broadcast news and events live using only an Android-enabled mobile devices and an internet connection via the cellular network or WiFi. Developers will have access to suggest changes to the source code, paving the roads for new innovative ideas based on the technology. Personal users and enterprises will have complete control over where the video is transferred over the internet, whereas with existing services videos are transferred to a third party that can access the video for viewing or analysis.

1.1. Contributions

The contribution of this project is in the design and implementation of an open-source video streaming application for Android devices. The components that enable this application are the following:

1. An Android application capable of capturing and streaming video to a server.
2. A modified version of the class responsible for Real-time Transfer Protocol (RTP) packet and datagram socket imported from SipDroid.
3. An implementation of the Real-time Streaming Protocol (RTSP) server that was based on the RFC 2326 specifications in order to transfer the Real-time Transfer Protocol to viewing clients.
4. An Android Device Communications Server was created to send and receive instructions between the web page and the Android device.
5. A web interface was designed for flexibility; SQL database statements were implemented for organization and ease of data access between interfaces.
6. An end-to-end performance evaluation of the resulting system.

1.2. Thesis Roadmap

The organization of this thesis is as follows. Chapter 2 presents the overview of the architecture employed for implementing video streaming on the Android device. The architecture includes components already accepted by the industry, such as Android, H.263, RTSP, RTP, SQL, and Flash Video.

Chapter 3 presents integration of components to accomplish the live video feed. The servers and the communication required to adequately stream the video is detailed. The servers consist of Android device communication, RTSP client communication with RTP transfer and conversion from H.263 to RTP H.263.

Chapter 4 explains the web interface and shows the requirements for displaying the live video to users on the web page. These requirements include conversion to flash video, a flash player and user account management via a MySQL database.

Chapter 5 discusses related works currently available in the field of video streaming. The fields include current Android video streaming services such as Bambuser, Qik, Ustream or other architectures that provide video streaming over the web and upcoming protocols along with the support in mobile devices.

Chapter 6 presents an experimental evaluation of the system and device. The server side tests featured encoder and server tests while monitoring application memory and processing performance as well as bandwidth analysis. The Android device tests examined the applications memory, processing and battery performance with bandwidth analysis while actively streaming video.

Finally, Chapter 7 concludes this thesis with a discussion of directions to consider in future development. The changes highlighted include enhanced video quality, heightened communication security, delay decreases along with increased longevity of the hard drive.

Chapter 2: Architecture

Figure 1 presents the architecture overview. The two remote, Internet-connected platforms are the Android device (on the left) and the server where the video is received (on the right). The Android device logs on the server using a custom protocol over a TCP connection established on an Android device port of the server.

Upon successful authentication, the Android device turns the camera on and begins transferring the video over UDP. The video is sent as RTP packets to the server. Users with an RTSP capable video player or RTSP client applications wishing to view or decode the stream connect to the RTSP server and after the stream information such as encoding type and connection information is requested from the server, the client sends a play command. When the server receives this request, it will begin forwarding the RTP packets to the viewer. The user's RTSP client displays the video; in our case, the RTSP client decodes the video and encodes it in an embeddable web video object. The encoding is saved to a Flash Video file on the server's hard drive or in memory.

The current user's information is recorded in an SQL database. When a user visits the web page, the SQL database is queried to find current live streams. To display the live streams from the Android device for the web user, the

Flash Video file is seeked to the end of file with a custom script on the server, new video data discovered by monitoring the Flash Video file size is transferred over HTTP where it is received by the Shockwave player on the web page.

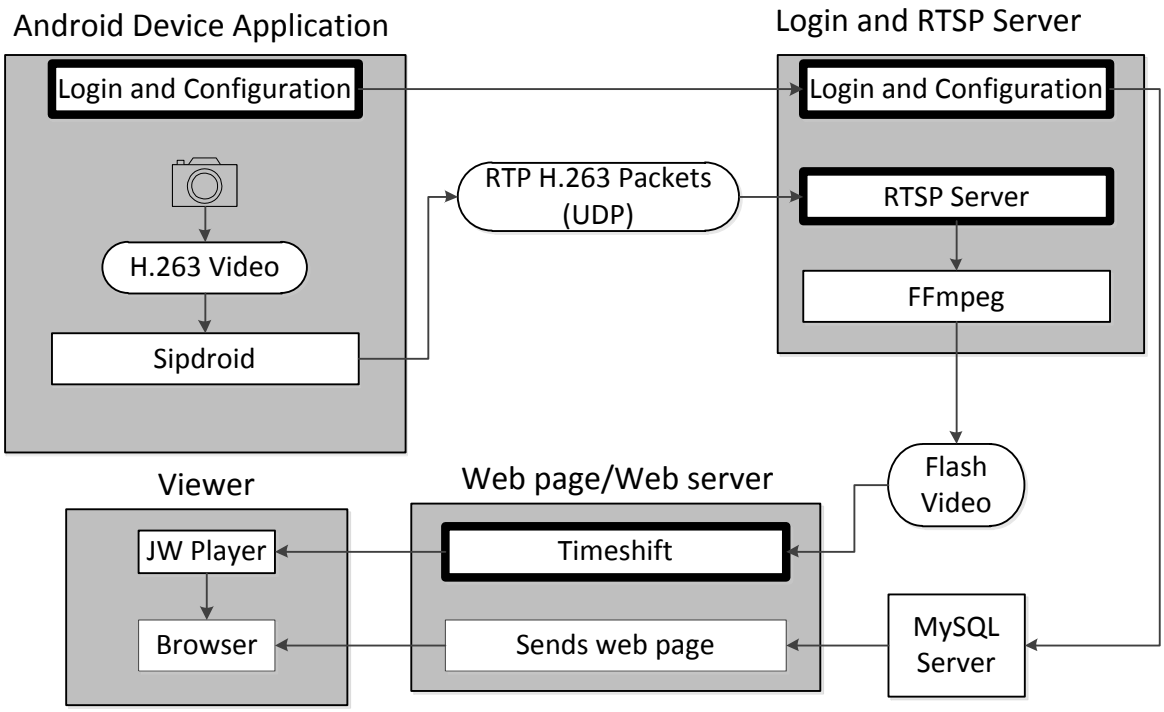

Figure 1: Architecture Overview

2.1. Existing Components

Several existing video standards and open source applications were used to process and stream the live video. The most important such components are H.263, Real-time Transport Protocol, Real Time Streaming Protocol, ffmpeg, and the Android Operating System, all presented below.

2.1.1. H.263

H.263 is a video encoding and decoding standard that was originally developed in 1996 by ITU-T Video Coding Experts Group for low bandwidth videoconferencing with adjustments made for higher quality streams in 1998 and 2000. H.263 was chosen from the multitude of similar video encoding/decoding standards because it is adopted by the Google Android Operating System, and thus it is supported by the majority of current android devices on the market. Along with compatibility, H.263 is a low bandwidth stream, making it possible to transfer over the 3rd generation (3G) cellular network.

2.1.2. Real-Time Transport Protocol

Real-time Transport Protocol, or RTP, is a packet format for transmitting audio and video encoded with various encoder types over the internet or local networks. It was originally developed in 1996 as RFC 1889 by the Internet Engineering Task Force[1] (IETF). In 2003, RFC 1889 was replaced by RFC 3550. Many software and hardware applications support RTP, such as Windows Media Player, QuickTime, VLC, mplayer, and ffmpeg. The applications may call upon other decoder software installed in the system to play the video contained within the RTP.

[1] IETF is a well established organization formed in 1986 to develop Internet standards. The organization has various meetings and relies on volunteers to assist in the production of standards.

RTP is transmitted over UDP thus it does not require TCP acknowledgements, or TCP ACK packets. The lack of ACK packets is one reason UDP was chosen as the type of packet chosen to send to the server. For example, if a device or any user sending video were to enter into an area of low cellular coverage this could result in not having the bandwidth required for all the packets to make it to the server in time for the viewer to see the events. This would result in a choppy video but video still transmits. In a worst case scenario where the user completely loses cellular coverage the playback would be halted on the viewer side, until the device reached an area where it can successfully transmit the packets then the viewer would immediately have the video redisplayed.

UDP RTP may present a problem with certain Network Address Translations (NAT). NAT is commonly used in facilities with more computers than external IP addresses (e.g., a family with one broadband modem and 3 computers that need to access the internet simultaneously). An internal IP address is assigned and converted to an external IP at the router when a computer communicates to another external IP. This works great for outgoing connections but when a computer behind a NAT wishes to become a server and listen on a port, port forwarding by either manual entry or uPnP needs to take place. The RFC 2326 provides a fallback mode to RTP over TCP in the event port forwarding does not occur. It should not be necessary to fallback to TCP mode as the server will likely be on the same computer or network as the decoding program.

Bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	Padding	Extension	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	SSRC identifiers						
96	CSRC						
96+32xCC	Profile extension header					Extension header length	
128+32xCC	Extension header						
	H.263 Payload Header						H.263 Compressed data stream
	H.263 Compressed data stream						

Figure 2: RTP Header

2.1.3. Real Time Streaming Protocol

The Real Time Streaming Protocol, or RTSP, is used to control the playback of an RTP stream. An RTSP Server allows a viewer to receive the RTP packets sent from the device. A viewer (e.g. VLC or ffmpeg) cannot directly open an RTP stream, therefore an RTSP server is required to provide additional stream information and control the starting and stopping of the streaming packets. The RTSP server is based on RFC 2326 published in 1998. It incorporates the following commands: options, describe, announce, setup, play, pause and teardown. The Session Description Protocol, or SDP, based on RFC 4566 is used in conjunction with the describe command.

An added bonus of using RTSP is that it is supported by the latest versions of the Safari and Konquer browsers. This is important because the HTML 5 final standard may support a <video> tag which will enable a video to be

embedded without the need for a Flash player. However, the operating system or browser will typically need to have the decoder installed for the type of video being transmitted. Development is still in its infancy at this time and successful playback of RTSP streams is buggy and likely to be improved as HTML5 is finalized. H.263 and H.264 are currently the most common streams for videos using RTP.

2.1.4. Encoding and Decoding

The encoders and decoders have the role of transforming the live video stream from the Real-time Transfer Protocol encoded video into the Flash Video for the web page. There are many types of encoding and decoding for video playback. Each typically has many advantages and disadvantages when compared to other types. The differences consist of various properties such as file size, encoding time, decoding time, processor usage, memory footprint, video quality and many more. Each video being transferred from the Android device requires processing, therefore an efficient encoder and decoder is desired for optimal scalability.

Several open source options were tested in this project, including ffmpeg, VLC, openRTSP combined with ffmpeg, and mplayer/mencoder. FFmpeg was chosen as the default transcoder (decode and re-encode) as it has the best picture quality and low CPU and memory usage per video stream. VLC was tested against ffmpeg on a variety of metrics. Mplayer and openRTSP

produced a frame speeds faster than the video was recorded at resulting in a stutter video.

2.1.5. Android Platform

Android is a recently developed operating system designed for mobile devices. It was developed by Google and uses a Linux based kernel, Java compatible libraries along with the just-in-time compiler for development in the Java programming language. It supports many hardware components. Common hardware consists of cameras, a WiFi communications chip, cellular commutations chip, Bluetooth sender and receiver, and a color touch screen. The Android Application Program Interface (API) contains many functions and classes to control the cellular devices. This functionality is all available in a single device with at least a day worth battery life.

For this project H.263 was used in development on the Android device. The initial Android API supports recording in H.263 with Android 3.0 introducing support for H.264. Android ships with a built-in RTP receiver with support for H.263 and H.264 decoding to display video play audio. Android 3.1 introduces RTP encoding support for transmitting audio over a network using the IETF standards. With the RTP encoding integration audio may be transmitted by using the operating system streaming class. Resolutions for the encoders are limited to the recording and playback capabilities of the camera, the processor speed, and the graphics card of the device.

2.2. Contributed Components

Several components were interlinked in order to make the live video streaming possible. The project interlinked all the components into a user friendly, easy to use application. The components consisted of video streaming from the Android device along with simple access to the live video stream on a web page.

2.2.1. Video Streaming

Video streaming from the Android device entails a combination of many elements. The custom RTSP server built to interact with the Android devices, their video streams and applications (e.g, ffmpeg or VLC) capable of playing live video is the largest contribution of the project (over 2000 lines of code). Chapter 3 provides details on the video streaming design.

2.2.2. Web Interface and User Management

Chapter 4: Web Interface and User Management provides more in-depth details on the web integration. The web page is designed to be easily incorporated into a predesigned page such as Word Press. It does this by keeping the layout of the page minimal then embedding this layout into a frame on an ordinary web page or transferred via AJAX into an element to create dynamic web pages that update in the background after a page has already loaded.

An SQL database is used to allow for ease of access on the web page and an organized storage structure for user and stream information. SQL allows for fast and efficient searching of large amounts of structured data on the server. The database can be used to maintain synchronization between the server and the web page. The MySQL server may be accessed by many clients simultaneously for queries, searches, or inserting data and updating data.

Chapter 3: Video Streaming

The video streaming is the main component of the project. Many individual projects and specifications were used in combination with a custom server and device application to create a live stream. The different components contributed are discussed throughout this chapter.

3.1. Server Side

The server opens many different listening ports that accept incoming connections. The protocols used for the ports are TCP and UDP. The server spawns two listeners, one for clients wishing to view the incoming video, or the viewer, and one for clients wishing to send the video, the phone. Each new TCP or UDP port required an additional thread. A different java class was developed for each thread type which may be spawned. A concurrent hashmap is used to synchronize the users viewing the video and the phones streaming the videos among the threads.

3.1.1. Android Device Communications Server

For the Android device sending video the TCP port chosen to send protocol and negotiation commands to is 10084. The port may be changed in web server configuration as this setting is retrieved from the associated web address. The default port chosen is not associated with any current service making it easy to bind to without interfering with other services. It also has

the additional advantage of being above 1024 thus does not require administrative permissions to bind to the port.

After negotiations have taken place on the TCP ports, the UDP ports are activated, if not already active to receive the RTP transmissions. The server then spawns a viewer or decoder process that connects to the Real Time Streaming Protocol server referred to in the next subsection. This process may be VLC, ffmpeg, or mencoder or any other decoder and/or encoder with RTSP input and a savable output. The command line used to call the decoder so that it may easily be replaced for analysis and conversion to any encoding type with a different program or different command arguments. The command line process is a server side command which transcodes the RTSP live stream from the Android device into a file of the Flash Video (FLV) format. The FLV file is saved on the server so that it may be read by the web program and be sent to viewers when they view the video. The decoder command line process may be modified to duplicate a video thereby saving the stream as a FLV file as well as other formats for future playback and download. VLC, mplayer and ffmpeg support a large range of audio and video decoders and encoders. Future upgrades to newer encoders and decoders will require very little work aside from converting from a streaming protocol to the web protocol such as Flash, or H.264 for HTML5 or newer versions of Flash.

3.1.2. Real Time Streaming Protocol Server

The RTSP server is a custom RTSP server built based on the specifications to provide the stream information and forward RTP packets to the correct destination. The TCP port 554 is recommended as this is the default port for a RTSP Server according to the RFC 2326. Some users without administrative privileges may be unable to bind to this port so the ports may be changed in the configuration. The web configuration will need to be set to the same port. RTSP clients may view a users live video by connecting at rtsp://<server address>/<username> (eg. rtsp://host.com/george)

The server attempts to limit the consumption of UDP ports by only binding to as many as necessary. Therefore most users will be provided with the default UDP port the server chose are start up to receive RTP packets. A user may receive a different port if someone else is logged in from the same IP as them using a NAT. This is a possibility if the number of users streaming is high as wireless networks use NAT or if many users are using WiFi at the same location to stream. Each time a user has joined or left the stream it is updated in the database so that a update list of users is available to the web interface.

3.1.3. Real-Time Transfer Protocol Server

The RTP server receives the RTP packets from the Android device. If there are not any viewers the packets are simply ignored. When a viewer connects

to the RTSP server and requests to play a stream, the RTP server modifies the RTP packets and forwards them to the viewing users or transcoder.

3.2. Android Device Side

The Android device will be the creator of the video via the camera and responsible for transmitting the video over the internet to the server. It also maintains communication with the server through a custom protocol to ensure proper identification and video streaming.

3.2.1. Conversion H.263 to RTP H.263

The video is transmitted through the RTP. As cellular networks are not extremely reliable and signal may fluctuate, the RTP Conversion takes place on the Android device. This requires more processing power and battery life than doing it on the server but will provide a continuous stream if the connection is temporarily lost or does not have the required bandwidth to transmit the full video by skipping packets at the server that never arrive and creating an image from the packets received at the server.

In order to produce the RTP packets, conversions from the camera devices needs to occur. The phone sends the H.263 provided by the Android API to a local TCP port on the phone; the phone receives the packets from this local port then converts the packets to RTP H.263 UDP packets by wrapping sequential frames with a RTP header as shown in figure 2. The raw video is split into packets around 1500 bytes. The data is sent in blocks with the

marker set in the header at the beginning of each block. The RTP packets are then forwarded to the server on the corresponding video UDP port received earlier.

The RTP header contains pieces of information so that the frame may be decoded on the server and viewing client end to produce a viewable video. This conversion to RTP is required because the H.263 video codec does not send information about the video such as length until the closing of a recording and it stores this information footer of the file. Thus, RTP packets contain the time length of the packet, or timestamp along with a sequence number to remedy this problem. Other information that is contained in the RTP packet consists of a source identifier, a marker, a version and the payload type, in our case H.263.

3.2.2. Communication

Most of the communication to the platform occurs over the TCP port connected to the Android Device Communications Server. Communication commands include information such as ports used for streaming the video, user log in information, session identification, play and termination commands. Sign up and forgot password dialogs take advantage of the functions already implemented on the server side web page by using the java built in HTTP client to fetch web pages using GET by including required information in the GET URI.

Chapter 4: Web Interface and User Management

The web interface is the first impression when viewing a stream. Therefore it should be minimal, only displaying necessary objects on the page while being easy to navigate and compatible with the majority of web browsers in the market.

The program used for the HTTP server was Apache. Apache is the dominant HTTP server of the market. It is available for Windows, Mac OSX, Linux, BSD and Solaris. Other HTTP servers should be fully capable of embedding the stream and/or sending the Flash file but may have difficulty with the PHP development or mod_rewrite modules.

The web page has been tested on several HTTP client browsers. It is compatible with the latest versions Safari, Internet Explorer, Firefox, Mozilla, Chrome on Windows; Safari, Firefox, and Opera on Mac OSX and Firefox, Chrome, and Opera on Linux. The browsers should have the latest versions of Adobe Flash browser plug-in although the video should be compatible with most previous versions.

4.1. Conversion to Flash Video

Flash Video is a file format that can be used to display videos onto a web page through a Shockwave Player and the Flash browser plug-in. Flash

Video is developed by Adobe. Flash has the plug-ins and installers for most operating systems and browsers. The only incompatibility in a standard operating system is with Apple's iOS used on devices such as iPhone, iPod and iPad. As the current market leader with a web browser plug-in capable of video playback, Flash was chosen for the file conversion in order to display the video live on the web page.

The Flash video file extension is FLV. Videos need to be embedded in the shockwave object beforehand as shockwave/Flash is packaged with built in Flash player. The "FLV file encodes synchronized audio and video streams. The audio and video data within FLV files are encoded in the same way as audio and video within SWF files." (Adobe Systems Incorporated, 2010)[2] JWPlayer has created an open source player capable of reading the FLV files and playing them live without needing to embed the FLV object into the SWF file.

4.2. Flash Player

JWPlayer is an open-source FLV player by LongTail Video. It provides a Shockwave Flash File which is used to transfer the FLV file provided by the video transcoding output. Each video stream is a single instance of JW Player. The embedded coding is automatically included by the viewer page.

[2] Adobe FLV File Format Specification, August 2010, <http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf>

Since the FLV file is outputted by the encoder, the entire video recording will be saved to the file, if the video were to be played directly through the HTTP server this will result in the video played being the beginning of the recording not actually being the live video. To combat this problem, a stream.php was created using the GET URI to fetch the correct file, the FLV file is seeked until the end of the file. To begin streaming the live video, The FLV header is sent along with the new data after the viewer requested the file should be sent. The UNIX stat command is repeatedly ran to determine the new file size, if the file size is different, the new data is then read and forwarded to the requesting client. The reason for using UNIX stat is the filesize() function in PHP does not update after the initial file reading. Since the PHP scripts must have the file extension .php. JWPlayer is unable to play the .php even though the output was the streaming data. Apache HTTP mod_rewrite module and its built in regular expression functionality is used to create a file shortcut with the filename convention being username.flv.

4.3. User Accounts and MySQL Database

The database server chosen is MySQL due to being mainstream, open-source, its extensive documentation and high portability. Binary packages are available for Windows, Linux, Mac OSX, BSD and Solaris. Many functions rely on the MySQL database. Such as when a user connects to the Android Device Communications Server the information provided is verified with the SQL database by SELECT'ing from a MySQL table WHERE the username and password correspond to the data provided. When no rows are returned the

device server responds 401 Unauthorized. If a row is returned from the SQL statement then the server responds 200 OK along with a session id and the 2 ports the device should send Real-time Transport Protocol. The information inputted into the SQL database must be validated before entry while watching for SQL injections. This leaves the responsibility of providing error messages to the user up to the web page. To avoid code duplication the web page receives sign up and forgot password information from the android device interface

If a password is forgotten the password forgotten link may be clicked. An email will be sent to the email in the database corresponding to the username; the email will contain a link to reset the password. Password reset confirmation codes must be used within two days or they will expire.

Chapter 5: Related Work

There are many software technologies related to the live video streaming field. None of them provide a complete solution for video streaming from a mobile device to an open platform viewable through a web page. Most of the software available is commercial products only available as a binary. The software does provide benefits for other classes of applications, such as streaming from a web camera device.

5.1. Free Commercial Web Video Streaming Services

Bambuser is one commercial product that provides live streaming from Android devices as well as several other mobile phones and devices. Videos streamed using Bambuser are sent to the Bambuser server and watermarked. There is no control over what the company does with the video such as analysis and profiling. Video quality is comparable to that obtained by this project with the project setting at low resolution. The projects high resolution does not display motion as well as the low quality but it gives a crisper and more detailed image. The web page uses the same method of Flash integration.

Qik is another commercial product very similar to Bambuser. When testing Qik with our hardware, the live video was unavailable. Videos could only be viewed on the web page at the end of the recording the video.

Ustream is allows broadcasting of live video. It centers on the idea of "lifecasting" and live video streaming of events online. A "viewer" or video client is required to be installed in order to view the live video. When testing the recording a not found error was displayed on the web page while attempting to view the URL given by the Android application.

All solutions publish the video to a web page which is viewable by any web browser via the Flash plug-in. Each supports a wide variety of devices and mobile operating systems including Android. None of the solutions provide a server which can be installed or a client that may have the receiving video server address modified.

5.2. Other Commercial Solutions

Adobe Flash Media Live Encoder enables live streaming from a device to an Adobe Flash Media Streaming Server. Adobe Media Streaming Servers is a commercial server that reproduces the streaming video from files while using features from Flash Media Live Encoder to produce live broadcasts for multiple users using the Real-Time Message Protocol (RTMP). RTMP is Adobe's streaming protocol detailed in the next section. Prices range from $995 to $4500.

Wowza Media Server is another commercial server which offers the ability to convert RTSP streams to RTMP streams. It does not offer the ability to do so on the fly and thus new videos cannot be added after signing up or logging

in. Prices begin at $995 for single license or $65 per month per server for service providers.

Red5 is an open source Flash server written in Java with Real-Time Message Protocol (RTMP) support. Red5 developers have made plans to develop real-time conversion from RTSP to RTMP. It could be used to bypass the conversion to Flash video and directly send the stream to the web browser without needing to write the video file to memory and/or the hard drive. Ffmpeg can transfer to an RTMP server which would give the ability to display videos with Red5. Red5 was not tested and it is not known whether it will be quicker than saving the FLV to the hard drive.

Erlyvideo, an open-source project with commercial availability, has already added RTSP to RTMP support. It was not extensively researched due to only supporting predefined videos and requiring the application to be restarted after adding a video. Its source may offer advantages in the conversion from RTSP to RTMP or be adapted for on-the-fly stream selection conversion functionality into the program.

5.3. Streaming Protocols

Real-Time Messaging Protocol (RTMP) was developed by Adobe originally as a proprietary protocol and is now an open specification for transmission of audio and video between Adobe Flash products. It has been criticized in the past for failing to provide documentation on key elements. RTMP is limited

to the decoders for which Adobe provides support. Development based on the specification may hinder future development of the project in regards to non-web video streaming with new decoders and encoder developed by the more established ITU-T VCEG. RTSP and RTP provide support for multiple types of audio and video encoders and decoders, whereas RTMP can only be used with a limited number of decoders supported by the protocol.

Chapter 6: Experimental Evaluation

The main objectives of the performance evaluation efforts were to understand the impact load on the server as well as the device. On the server side, different encoders and decoders were evaluated to determine which would perform the best in a multi-user, multi-video stream environment. Performance is measured by the processing power required to convert a stream, the image quality compared to the original, playback of the stream, the bandwidth required to transfer the video stream to a user viewing the video. The main server was evaluated along with the encoders in the tests.

On the Android device, battery, network usage and processor usage are measured to determine the resources needed by the mobile device. The resources measured are scarce resources in a mobile environment and must be conserved as much as possible.

6.1. Experimental Setup

The Android device used during testing was the Google Nexus One with the Android Operating System version 2.3.4. The Nexus one has a 1 Ghz Snapdragon processor, 512 MB of RAM, and a 5 megapixel camera with IEEE802.11g for wireless communication. A 1500mAH battery at 4084mV was used in gathering of the battery results.

The operating system of the computer hosting the main server was the Linux distribution Debian 6.0. The hardware consisted of a Pentium D dual-core processor running at 3.4 Ghz with 2GB of RAM. The tests were performed in a local area network environment over wireless IEEE802.11g.

Picture quality for high resolution is set to 352 pixels width by 288 pixel height. The resolution may theoretically be increased, but the hardware limitations of the Android device did not allow for higher resolution in these experiments. Low resolution is set to 176x144 pixels. The frames per second are set at 30 frames per second in both resolutions. However, during testing, the Nexus One device was only capable of recording at 27 frames per second. The camera hardware did not support any other values. Videos recorded at 30 frames per second will result in larger video, thus using more bandwidth to send the extra 3 frames per second.

In order to maintain a consistent, reproducible experiment, the YouTube sensation Matt in *Where the Hell is Matt (2006)*[3] was streamed for each test except the scalability evaluation (Chapter 6.2.1.4.). The video combines multiple color scenes with dancing in each scene; the video was chosen to simulate different visual environments. The video was set to full screen on a 19" LCD and recorded with the Nexus One. The stress tests consisted of various indoor environments and each with varying degrees of movement.

[3] *Where the Hell is Matt* is available on YouTube at http://www.youtube.com/watch?v=bNF_P281Uu4

6.2. Experimental Results

The commands used to perform the transcoding tests are

```
`cvlc rtsp://host.com/user --
sout='#transcode{vcodec=FLV1,scale=<N>,qmax=1000,qmin=1000}:stand
ard{access=file,mux=ffmpeg{mux=flv},dst="file.flv"}`
```

<N> is the scaling ratio where 1 is set for no scaling, 2 is set for image scaling of x2.

```
`ffmpeg -i  rtsp://host.com/user -y -f flvfile.flv`.
```

Picture quality was determined with visual screen shots and using the peak signal-to-noise ratio (PSNR). ImageMagick, an open-source imaging program, was used to compute the value between the original and the transcoded images.

Figure 5 and Figure 6 show the server with the video transcoding process and their processor loads and memory usages. A single user sent a stream of the recorded the YouTube video. The Unix `` `ps auxw` `` was logged in 1 second intervals to obtain CPU usage and memory usage graph values, presented in Figure 4: CPU Consumption by the Encoders and the Server with a Pentium D at 3.4 Ghz.

To measure the communication bandwidth from the Android device to the server, packets were captured using `tcpdump` on the server, then analyzed and filtered with Wireshark. Wireshark was unable to graph a total number

of bytes per second with multiple streams. A custom script was used to add the total number of bytes for each second in order to produce a clean graph.

In order to determine the bandwidth used while transferring the video to the web user the total video file size was used. The Unix `` `stat -c%s<file>` `` command was logged in 1 second intervals in order to obtain bandwidth values. The difference in the file size was taken from the last second to find the number of bytes transferred per second.

6.2.1. Encoder and Server Tests

Several encoders were tested. The encoders tested were Mencoder (a mplayer based encoder), VLC, openRTSP with ffmpeg, and ffmpeg. All are open source and natively run on Linux. They have also been ported to Windows, and Mac OS X operating systems. Mencoder consumed resources similar to that of VLC scaled to x2 with similar quality results. It did not display the correct frame rate, therefore it was excluded early on from testing. OpenRTSP combined with ffmpeg was the incorrect frame rate and frames were dropped. It was left out of testing due to its display issues. VLC and ffmpeg had no problems displaying the video and were extensively tested and reported on.

6.2.1.1. Picture Quality

Figure 3 shows the different picture qualities with various H.263 RTSP to Flash Video converters. The image was taken from a video screen shot while the Android device streamed the still image from a tripod in a very well lit room. VLC performed the poorest in video quality unless video is scaled. As seen later, when the video is scaled, the processing power required to re-encode the video increase significantly. Ffmpeg visually appeared to be most similar to the original image.

A peak signal-to-noise ratio between the original image and the images produced through the transcoding process is computed. The ratio captures the amount of distortion by finding the amount of error added by compressing the image. The ratio is computed between the original image and the flash video to be displayed on the web page. The comparison between the original image and the image outputted by ffmpeg was infinite indicating an identical image. The original image was compared to VLC with no scale and at a scale of 2, the PSNR ratio obtained is 20.3914 dB and 20.2257, respectively.

Original

Ffmpeg

VLC

VLC scaled x 2

Figure 3: Picture Quality Reference

6.2.1.2. CPU and Memory Usage

CPU and memory usage of the various encoders and the main server were graphed under a single user to see the impact and processing power required to stream one video. The values increase linearly as the number of Android devices connected to the server increases. The performance and bandwidth results are dependent on the amount of movement in the video.

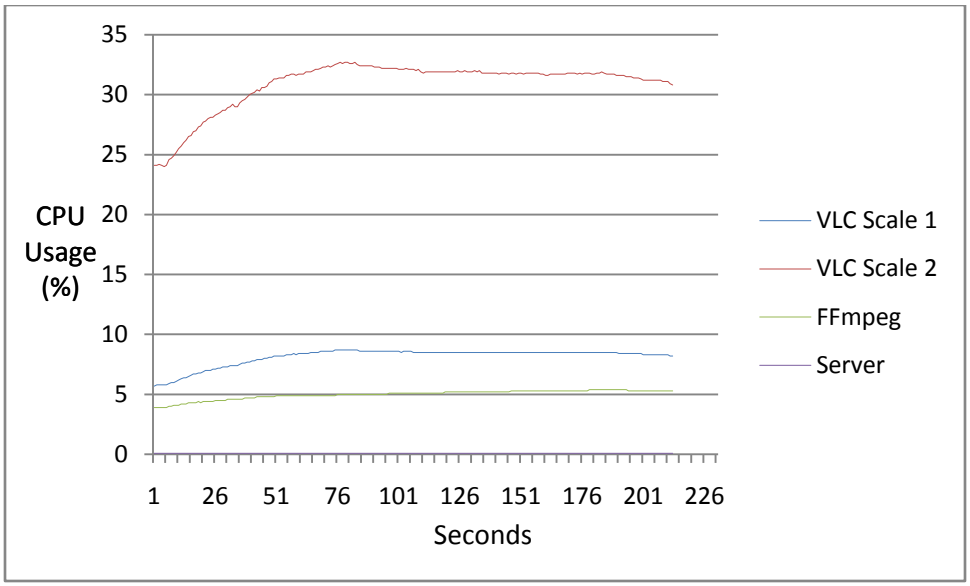

Figure 4: CPU Consumption by the Encoders and the Server with a Pentium D at 3.4 Ghz.

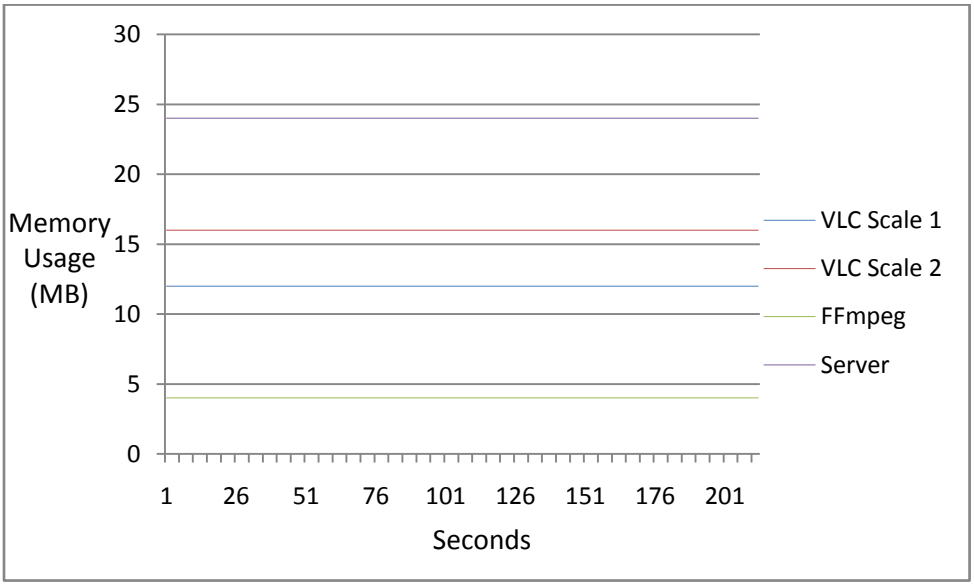

Figure 5: Memory Consumption by the Encoders and the Server.

6.2.1.3. Bandwidth

The bandwidth tests conducted in low quality show a similar bandwidth of 26KB/s (0.20 Mbps). This is what is expected as the Flash Video scheme is based on the H.263 codec so the video image itself is not altered.

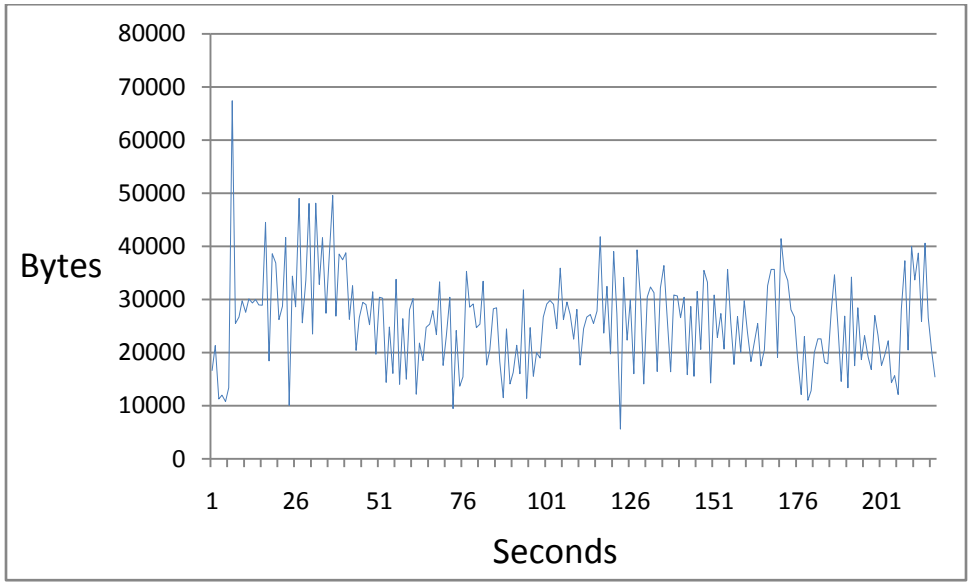

Figure 6: Bandwidth from Server to Web Browser.

6.2.1.4. Scalability

The server was tested with 5 users streaming simultaneously for approximately 4 minutes, the process scaled as expected. Each user consumed 5-6% CPU for the transcoding as shown in the single user results. The server did very little processing though it consumed more memory as it needs to keep track of each user's information streaming to and from. Since the video encoding is a CPU intensive process, multi-processor systems are recommended. In order for the system to handle a large streaming user base systems may need to be clustered together.

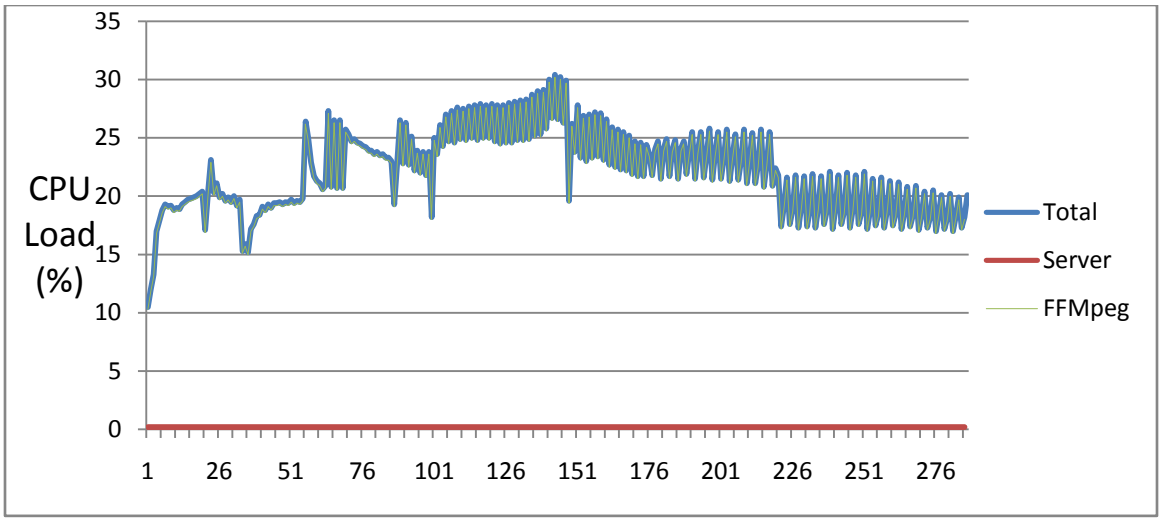

Figure 7: CPU Usage by Server with 5 Users

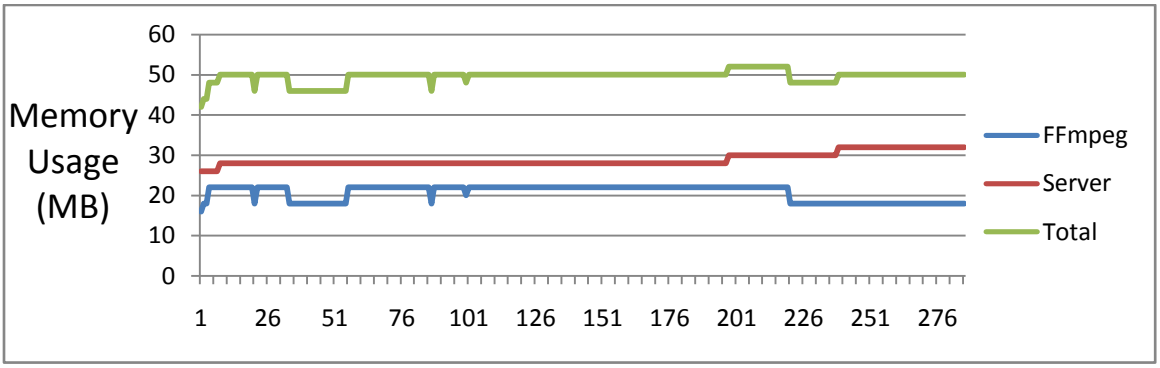

Figure 8: Memory Usage by Server with 5 Users

6.2.2. Android Device

The Android device uses the camera, screen, and network to send the live video stream. All these hardware components consume resources such as a user's cellular data plan or battery. The components are measured with various tests.

6.2.2.1. Bandwidth

Figure 7 presents the network bandwidth when sending RTP packets from the Android Device to the server. At full resolution the average bandwidth was 28KB/s (0.23 Mbps) with a maximum burst of 103KB/s (0.80 Mbps) while at half resolution the average bandwidth is 26KB/s (0.20 Mbps) with a maximum burst of 80KB/s (0.62 Mbps). No packet loss was observed in the VLC codec and media statistics dialog box viewed over many various recordings. Other measured field tests have shown more variability of bandwidth in the low quality stream typically resulting in lower bandwidth consumption.

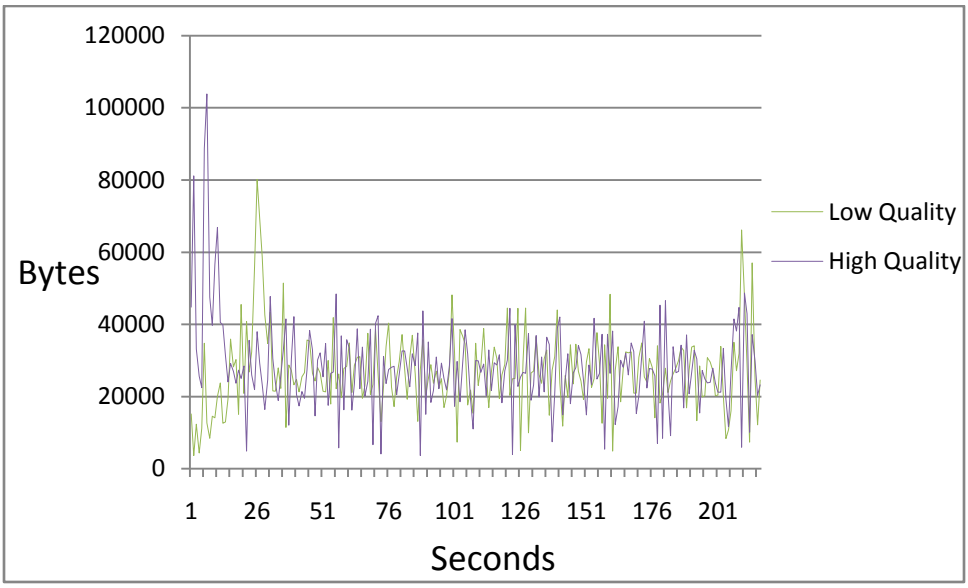

Figure 9: RTP Bandwidth from Android Device to Server.

6.2.2.2. Battery and CPU Usage

The device statistics could not be as accurately measured without rooting the Android phone. Battery Monitor and System Tuner were used after

streaming the YouTube video. The statistics obtained indicate a 48% overall battery usage caused by the screen to display the video during the recording. Only 26% of the battery consumption that occurred was by the program to process and send the video. The processor ran at 60% average during the recording in low resolution and 75% in high resolution.

6.2.2.3. Delay

Delay between recording and playback on the webpage varied based on video quality. At high quality, the delay was around 2 seconds while at low quality it was around 1.5 seconds. Viewing the stream directly through the RTSP server with RTP caching disabled and without converting to a Flash Video stream will result in a 2 second delay. The delay measured above was while using the Internet Explorer 8 browser. Safari and Chrome show similar results while Firefox and Opera experienced a longer delay of at least 4-5 seconds.

Chapter 7: Conclusions and Future Work

Many upgrades will take place as technology rapidly progresses in the cellular industry. The upgrades to networks and mobile devices will enable higher quality video transmissions with less delay. The project has been constructed for easy integration and modification to take full advantage of future technologies.

7.1. Future Directions

Several factors have already affected the future of video streaming. In Android 3.0, Google introduces H.264 AVC codec. H.264 is higher quality but consumes more uploading bandwidth as well as more phone and server power. H.264 has disclaimers in its licensing terms which limit what H.264 may be used for without requiring a licensing fee. The terms are available on the MPEGLA website. 3GPP Long Term Evolution, or LTE, is now being deployed in test cities by mobile service providers. LTE offers significant improvements in data transfer speed giving video streaming the ability to transfer high definition content from and to the phone.

Other projects may assist in supporting H.264 streaming in many different ways such as using the mod_264_streaming module to send new metadata of a live video. It would most likely need code modifications to support the live stream but these may be minor since streaming module already moves

header information. The header relocation is required for H.264 streaming. The only downfall is in its current form. It reads the metadata from the end of the file instead of producing a metadata on the fly by analyzing the frames. Moovrelocator relocates the metadata to the header of H.264 files. It could be modified as well to start from current position, infinitely stream and update. It is written in the PHP5 programming language. Security can be improved by authenticating completely over HTTPS via URL Connection instead of TCP socket.

If the encoder for transforming the RTSP to Flash Video's processing could be sped up enough by not requiring as many blocks of data in the buffer to efficiently process images. The delay could be reduced to phone processing time, which should become more negligible in the future as processors designed for cell phones increase in speed and efficiency.

Encoders and decoders are constantly changing and being upgraded as processing power and network bandwidth becomes more abundant. The server was designed with this in mind and offers easy adjustments to support these changes over the years. Long-term compatibility will be much higher by using well-established standards and along with adaptable code. Ffmpeg has been used by many open source projects and is continuously updated. Its performance outranked the other encoders in terms of video playback, picture quality and performance as seen in Figure 4, Figure 5 and Figure 6.

Audio was not tested in this project. The server has been developed with the capabilities of implementing various audio encoders and decoders for audio playback. RTP audio support is introduced in Android 3.0 and may be implemented into the program by sending the RTP packets to the server on the audio port indicated in the protocol.

Videos may be saved to Linux Ramdisk as they are temporary and will immediately be written, read and then transferred to any viewers. Using the Ramdisk will eliminate any bottleneck due to hard drive transfer speeds. Hard drives generally have trouble reading and writing multiple streams due to the constant movement of the hard drive reader head. Depending on the number of users recording video and the length of recording, a large amount of RAM may be required.

7.2. Final Thoughts

An Android mobile device combined with its camera and internet capabilities is used to stream real-time video to a web page. It accomplishes the streaming using various open source projects and open protocols. Using standardized and open protocols increases compatibility among clients and results in far more support than less popular or closed protocols. The open protocols used in this project were RTSP, RTP and H.263. Various open source encoders were tested and ffmpeg proved to be the best encoder for the task. The video streaming design uses minimal processing with little overhead while maintaining picture quality and frame rate.

References

[1] Session Description Protocol (SDP), December 2005, Network Working Group, Request for Comments: 4317, <http://www.ietf.org/rfc/rfc4317.txt>

[2] Real-time Protocol (RTP) H.264 Video, February 2005, Network Working Group, Requst for Comments: 3984, <http://www.rfc-editor.org/rfc/rfc3984.txt>

[3] Real-time Protocol (RTP) H.263 Video, September 1997, Network Working Group, Requst for Comments: 2190, <http://tools.ietf.org/rfc/rfc2190.txt>

[4] Real-time Streaming Protocol (RTSP), April 1998, Network Working Group, Requst for Comments: 2326, <http://tools.ietf.org/html/rfc2326>

[5] Real-time Transport Protocol, September 2011, Overview of system and packet structure, <http://en.wikipedia.org/wiki/Real-time_Transport_Protocol>

[6] Real-Time Streaming Protocol, September 2011, Overview of system and capabilities <http://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol>

[7] Adobe, September 2011, References to Flash products <http://www.adobe.com>

[8] Sipdroid Project, June 2011, RTP conversion <http://www.sipdroid.org>

[9] VideoLAN Client (VLC), September 2011, RTSP to FLV conversion <http://www.vlc.com>

[10] FFMpeg, September 2011, RTSP to FLV conversion <http://ffmpeg.org>

[11] JWPlayer, September 2011, embedded FLV player <http://longtailvideo.com>

[12] N. Vun and Y. H. Ooi. 2010. Implementation of an Android Phone Based Video Streamer. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing* (GREENCOM-CPSCOM '10). IEEE Computer Society, Washington, DC, USA, 912-915. DOI=10.1109/GreenCom-CPSCom.2010.76 <http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.76>

[13] Dapeng Wu; Hou, Y.T., Wenwu Zhu; Ya-Qin Zhang; and Peha, J.M; August 2002. Streaming Video over the Internet: approaches and directions In *IEEE Transactions on Circuits and Systems for Video Technology*, Volume 11 Issue 3, On pages 282-300.

[14] Lazaro, O.; Girma, D.; Dunlop, J.; 2004. H.263 video traffic modelling for low bit rate wireless communications In *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004*. Volume 3, On pages 2124-2128.

[15] Cote, G.; Erol, B.; Gallant, M.; Kossentini, F.; 2002. H. 263+: video coding at low bit rates In *IEEE Transactions on Circuits and Systems for Video Technology,* Volume 8 Issue 7, On pages 849-866.

[16] Michael Ransburg, Mario Jonke, and Hermann Hellwagner; An Evaluation of Mobile End Devices in Multimedia Streaming Scenarios. <http://www-itec.uni-klu.ac.at/publications/mmc/paper9355.pdf>

[17] Vun, N.; Ansary, M.; Implementation of an embedded H.264 live video streaming system In *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium* on , vol., no., pp.1-4, 7-10 June 2010 doi: 10.1109/ISCE.2010.5523699