Developing multimedia algorithms for the XO-Laptop (OLPC)

By:

Obada Sawalha

Supervised by:

Dr Yiannis Andreopoulos

26/03/2010

3rd Year Project

Final Report

Electronic Engineering Department
University College London (UCL)

Contents

1. Abstract

The increasing demand for real-time media over IP (with popular multimedia-rich services such as YouTube and BBC iPlayer) is currently straining the existing networking infrastructure. As a result, developing experimental testbeds that incorporate the latest developments in such technologies becomes important.

This project concentrates on low-delay video streaming over IP. This is achieved via the User Datagram Protocol (UDP) instead of the conventional Transmission Control Protocol (TCP). An encapsulation within a Real Time Protocol (RTP) session is pursued in order to allow for standard-compliant real-time stream transport. In terms of implementation, a real-time wireless video-transmission software testbed is proposed using the state-of-the-art MPEG/ITU AVC/H.264 video coding standard. The server hardware is deliberately chosen to be a very low-end multimedia-enabled platform (the "100$ laptop" of the OLPC foundation), while the client (receiver) can be any platform. The proposed client-server architecture uses the latest VLC video player to decode and display the video stream in real time.

The practical outcomes of the work are:

- The derived implementation consisting of the RTP/UDP framework programmed in JAVA (for both the client-server architecture). The derived software is provided as an open-source Creative-Commons project at http://www.ee.ucl.ac.uk/~iandreop/UNV.html.
- A standard-compliant AVC/H.264 codec incorporated within the proposed framework as a practical example. Measurements of traffic volume versus video quality are presented for several test cases of common video formats.
- The project report explaining how the entire framework can be tailored for encapsulation (and measurement) of other video coding standards.

2. Introduction

This project aims to address issues related to streaming video over packet switched IP networks, which have no bandwidth provisioning for media services.

The aim of the project is to offer an open alternative for robust video streaming [1] using the latest video coding and transmission technology. It is fair to assume that unpredictable resource provisioning and the bandwidth limitation of IP networks are here to stay as they are inherit in the architecture of such networks. Applications over such networks must therefore be able to cope within the uncertain conditions of such a network design.

The English proverb says 'A picture speaks a thousand words', and is often used to express the power of visual data. Similarly when looking at video streaming applications we come to realise that transmitting visual data is much more demanding than transmitting 'words'; the relation will almost definitely surpass the 1000:1 ratio.

Video data in its raw format is rather useless with respect to video streaming - even storage of short raw video is problematic - which is why various compression and encoding techniques are employed to make video data better suited to systems and networks where storage and transmission resources are limited.

Video streaming, unlike video download, can by definition begin playback of the video as it arrives [1]. Due to the nature of the Internet protocol (IP) both streaming and downloading rely on transmission of video data one packet at a time. Video streaming however utilises this to ensure that a given number of packets can be used to playback part of the streamed video as it arrives at the receiver side.

Video streaming therefore depends on packets arriving, and on time, to maintain playback - for which IP on its own is insufficient. The usual solution as employed by popular multimedia applications such as Youtube, iPlayer and Skype is the use of the Transmission Control Protocol (TCP) which employs functions such as acknowledgements and retransmissions to guarantee error-free packet arrival at the destination.

This however comes at the cost of speed since one must wait for the acknowledgement of correct reception of each TCP packet prior to the transmission of the next ones. This limits the quality factors of video that can be transmitted over the network.

This project explores the use of the User Datagram Protocol (UDP) with the Real Time Protocol (RTP) and the Real Time Streaming Protocol (RTSP) as an alternative configuration to overcome this problem.

Video compression is also analysed in order to obtain a system capable of operating with a raw video input, and to utilise the latest codec technology to ensure quality, reliability and easy widespread deployment.

The OLPC (XO-Laptop) [2] is used throughout this project as the server computer for reasons relating to its hardware configuration. It is a low-end platform system running the Linux operating system. This introduces significant challenges to the application because it has to be structured in a way such that it uses minimum hardware resources in order to operate correctly. The OLPC is also a multimedia enabled system with integrated camera and microphone, as well as WiFi capability.

These specifications render it very similar to a modern WiFi enabled mobile phone, running a Linux-based operating system, such as Google's Android [3], which widens the scope for further deployment and makes the project relevant to a large user community.

3. Networking

There are a huge number of protocols that define and control connections between hosts on a network. The 'Internet Protocol Suite' is a full set of these inter-networking protocols, and groups the protocols into layers known as the network(or internet) layer, the transport (or service) layer, and the application layer. [4]

A typical application will utilise stack connections, with several protocols used in a multi-layered network architecture where protocols in the higher layers utilize and expand on protocols in the lower layers.

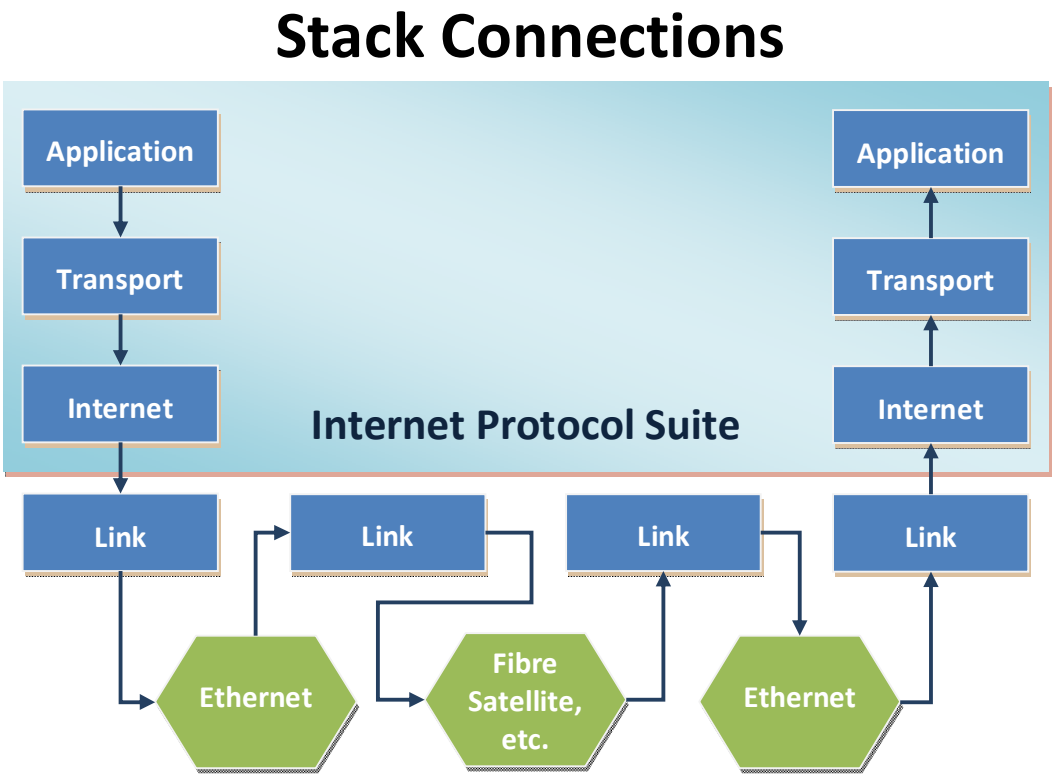

Figure 1 – A graphic representation of a Stack connection with IPS layers indicated [5]

Typically protocols that depend on others have a nested packet structure where each protocol adds its own header to the packet with the lower layer protocols headers wrapped around those from higher layers.[6]

The Internet Protocol (IP) is a network layer protocol which routes data in packet (datagram) form through the network. Its role is sending the packets from host to host, but is on its own not very reliable as packets with an IP header can disappear in the system or can arrive in the wrong order. IP includes several protocols. The

most important ones for our purposes in this work are described in the following sections.

3.1. TCP

Transmission Control Protocol (TCP) is a transport layer protocol for managing end-to-end connections and depends on IP, the Internet Protocol, to deliver the packets of data across the connection [7]. It is a full duplex protocol, which means that it supports two parallel data streams for incoming and outgoing data at a given host. [8]

TCP's distinguishing features are guaranteed delivery of packets, handling of timeouts and the handling of retransmission requests, all of which TCP achieves by implementing an acknowledgement system whereby the server waits for an ACK packet from the client to confirm the previous packet arrival before it sends the next one.[9]

TCP therefore overcomes the shortcomings of IP, creating a reliable connection that is the basis for many services including the internet's HTTP, however when looking at video streaming applications, the constant need to wait for acknowledgement heavily affects the speed of the connection.

3.2. UDP

The User Datagram Protocol (UDP) lies along with TCP in the transport layer of the Internet Protocols Suite. Where TCP guarantees that data packets will be delivered and in the correct order, UDP has few error recovery services, instead offering direct transport of packets without the need for opening a connection [10].

Due to UDP not requiring an acknowledgement that the packet has been received [9], it is much faster than TCP. However should a packet fail to arrive, it will not be resent.

Packets sent using IP may also arrive in the wrong order at the client, even if sent in the right order by the server, a predicament to which UDP has no solution.

The main advantage of UDP in comparison to TCP is its speed; this however comes at the cost of reliability, which renders applications running a UDP connection more complex as they must implement a framework for dealing with all packet errors.

3.3. RTP/RTSP

When implementing applications involving real-time media streaming, both UDP and TCP fall short of the requirements. On the one hand TCP is very reliable and can stream the packets to the client rather efficiently; however its speed limitations limit the quality aspects of the media stream, as one has to settle for less FPS (Frames per Second) than desired, as well as implications on other encoding configurations to limit the bit-rate.

On the other hand, UDP has fewer limitations on the speed, but its unreliability in packet delivery means that media frames may need to be dropped, which has a huge effect on the perceived quality of the video stream.

The Real-Time Protocol (RTP) provides the solution for this dilemma. RTP is designed to operate in the application layer, and as such builds upon protocols in the transport layer.

RTP is usually operated with UDP due to its speed [11]. Furthermore TCP's packet recovery functions not only slow down the connection, but a live or real time situation the resending of packets clogs up the connection, further increasing the time that packets require to arrive at the client and causing unnecessary pressure within the network.

RTP therefore effectively utilizes the strength of UDP and introduces an array of its own functions to raise the level of reliability, such as time-stamping and sequence-numbering the packets. [12]

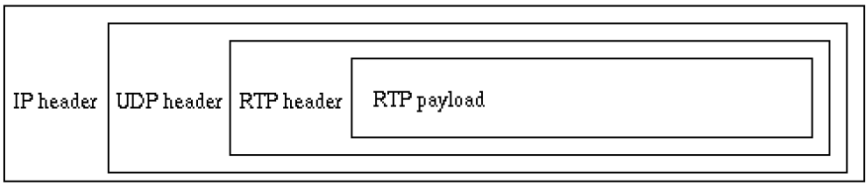

Figure 2 – RTP data in a UDP/IP packet, Adapted from "Multimedia Over IP: RSVP, RTP, RTCP, RTSP" [11]

The Real-Time Streaming Protocol (RTP) is another application level protocol which usually works alongside RTP [13].

RTSP is termed a 'control protocol' as it provides the control functions needed for real time streaming such as play, pause, setup, describe and other such services to control and update about the streaming session. It is essentially a 'VCR style' 'remote control' service [14], while another protocol, such as RTP, deals with the actual data delivery [15].

4. Video/Media Overview

4.1. Video Compression

4.1.1. What is video compression?

The aim of video compression, like all data compression, is to omit redundant data from the source signal so that the data transferred is less, meaning that with a fixed speed and bandwidth, more data that is relevant can be sent over a connection. This ultimately means that greater quality video can be sent over networks, be it streamed video or for storage and playback.

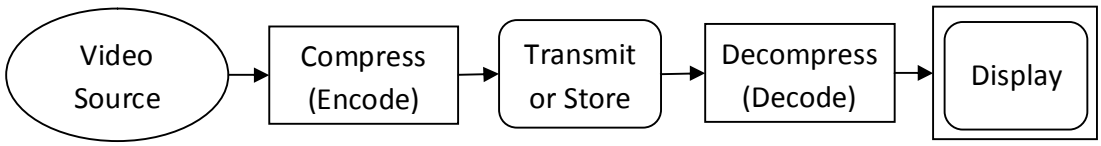

Figure 1: *Compression/Decompression system overview; adapted from "H.264 and MPEG-4 video compression: video coding for next-generation multimedia" [17]*

Because raw video data requires vast amounts of storage, it is especially important with video's to store or transmit them in as few bits of data as possible, while maintaining no substantial loss of visual information. The job of the encoder (compression) is to take a series of video frames and convert them into a bit-stream. [*16*] The decoder (decompression) reverses this process and converts a bit-stream into decoded video that can be sent to the display device of the receiver. The issue is therefore to achieve a balance between the *bit rate* (i.e. the number of bits contained in the bit-stream for one second of video data) and the *fidelity* (quality) of the output video, these two features being largely at odds with each other.

Codecs are analysed according to their *channel-throughput* (the overhead incurred by the transmission system due to the codec) and the *output distortion* [*17*]. This attempt at balancing is called *encoding efficiency* and has become an important measure for comparison between the different emerging and internationally standardised codecs.

The best possible compression techniques go beyond simple linear encoding and decoding [17]; an analysis of the data type and how the final data outcome will be used is essential for higher level encoding.

The book entitled 'Video Compression and Communications' [18] gives a simple illustration of this point; the authors demonstrate that encoding the letters 'e' and 'q' by

simply using the same number of bits can be seen as simple encoding, however with an understanding that the letter 'e' is statistically more frequent in occurrence within English language words, it would be far more efficient to encode it with less bits than 'q'.

This however is not the only factor governing the reduction of data size for transmission. There are many more in existence, and the techniques to encode/decode naturally fall within two distinct categories; Lossy and Lossless compression.

Codecs from both categories also vary immensely in their complexity, an important characteristic of a codec to compare; because complex codecs consume more system resources, which is especially important in this project as the XO Laptop is a low-end platform device.

4.1.1.1. Lossless Compression

Lossless compression techniques have the distinguishing feature of reconstructing, given the compressed data, the original video data perfectly with no loss of information. [19]

The example above regarding 'e' and 'q' encoding, although not video compression, can be described as lossless; because the data can be exactly reproduced by the decoder with no loss of information, despite 'e' being encoded with less bits.

Iain Richardson in his book 'H.264 and MPEG-4 video compression' cites that with current lossless image compression standards, using JPEG-LS as an example, the compression ratio achieved is between 3 to 4, i.e. the compressed bit-stream is 3 to 4 times smaller in size than the original video data [17].

4.1.1.2. Lossy Compression

Lossy compression techniques lend to the fact that certain visual data may be omitted without being noticed or given much attention by the human visual system. This enables such compression techniques to reach much higher compression ratios than lossless techniques.

The emphasis here is on the perception of the viewer and therefore techniques utilised rely on changes which even when noticed are deemed unimportant.

4.1.2. Compression Techniques

4.1.2.1. Quantisation

Quantisation is a very important compression technique that is used in many different applications not limited to video. Video quantisation allows obtaining of lower data rates, which are more easily transmitted through networks, by limiting the set of discrete values that represent the pixel values in every component of the colour space.

As a lossy compression technique, quantisation is able to achieve very high compression factors, which can be larger than 50 [20]. This is over 10 times the compression factors obtained by lossless techniques as discussed in previously.

Quantisation however causes a lot of distortion in the encoded video because it introduces a level of uncertainty proportional to the Quantisation Parameter (QP) value.

The QP represents the number of quantisation levels and therefore an arbitrary value will be represented by a quantised value where the percentage error between the two values increases with higher QP values.

4.1.2.2. Motion estimation and Spatial Correlation

Figure 3: *Temporal and Spatial correlation in two consecutive video frames; from"H.264 and MPEG-4 video compression: video coding for next-generation multimedia" [17]*

Temporal correlation is the similarity that occurs between consecutive video frames as can be seen from figure 3. Motion estimation is a lossless compression technique that lends to the fact that an algorithm can be created that utilises the data in past and future frames to minimise the data required to decode the current frame [*21*].

Spatial correlation is based on the fact that adjacent pixels in a single video frame have great similarity. This allows the prediction of the value of a complete region based on pixel area comparison with a single point; which significantly reduces the data that needs to be stored about all such areas. [17]

4.2. Video Standards

4.2.1. MPEG-2/4

The Moving Pictures Experts Group (MPEG) was established to put together standards for audio-video compression and transmission in 1988.

MPEG standards do not provide the actual codec; rather they describe the rules an encoder is required to follow to successfully encode a bit-stream that the MPEG decoder can play back. Therefore there are several implementations of MPEG standards that vary in their efficiency and also in the commercial/open licenses they fall under. The two popular MPEG standards are MPEG-2 and MPEG-4. [17]

MPEG-2 is still very popular today, many digital television signals (cable and satellite) use the MPEG-2 standard. It is also in wide use in DVD and Blu-ray media.

MPEG-4 is the later one and is in fact a group of many standards, two of which are; MPEG-4 Visual and MPEG-4 AVC. MPEG-4 AVC is discussed in the next section.

MPEG-4 visual has its focus on flexibility and non-conventional video streams, and as such is highly suited to applications in 3D and computer-generated visual data alongside the traditional rectangular video frames.

MPEG compliant codecs use inter-frame encoding, so other frame data is utilised with motion compensation (MC) to reduce the current frame size. Videos encoded by MPEG compliant codecs must be in YCbCr colour format, but can accept either progressive or interlaced video. MPEG compliant codecs can also be contained in a variety of multimedia wrappers.

4.2.2. H.264/AVC

H.264/AVC is the outcome of collaboration between MPEG and the Video Coding Experts Group (VCEG) and has its focus on coding efficiency, employing the latest techniques for video compression, and transmission efficiency, with built in functions to make its transmission more reliable and robust.

Like other MPEG standards, H.264/AVC uses both intra-frame and inter-frame encoding algorithms, and works in the YCbCr colour space.

It can also accept both progressive and interlaced videos, however as a new standard, wrappers that support H.264/AVC encoded video are limited.

H.264/AVC utilises a number of video compression techniques as can be seen in the encoder diagram in figure 5.

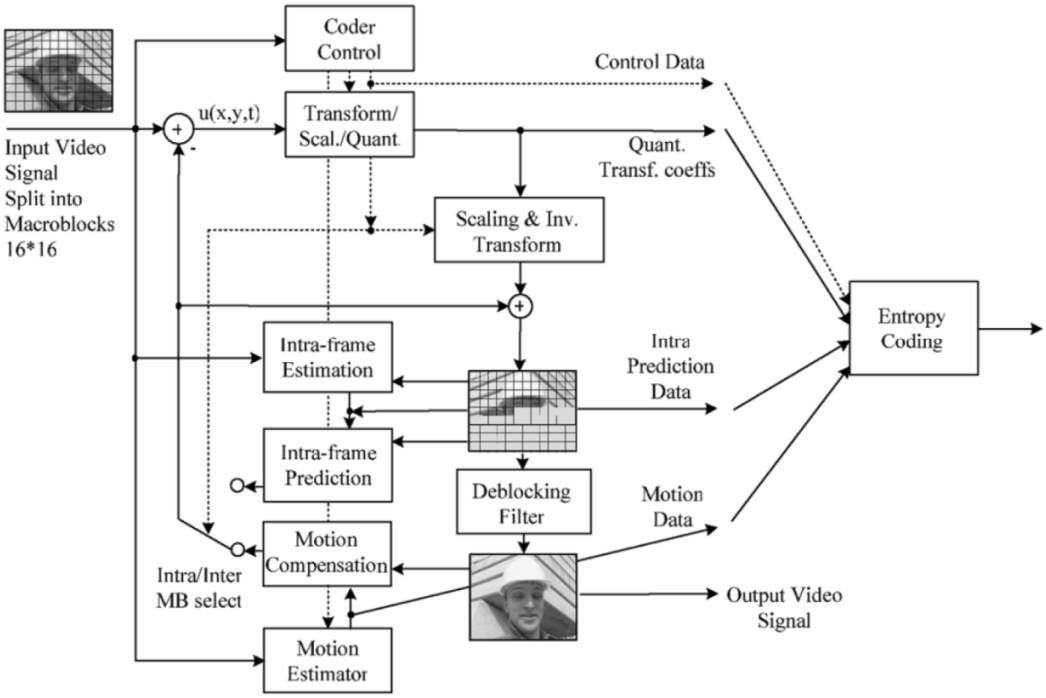

Figure 5: *H.264/AVC video encoder diagram; taken from "Video Compression - From concepts to the H.264/AVC standard, Proceedings of the IEEE"* [*21*]

The quantisation forms an essential part of the system, and is controlled by data such as the QP value or constant bit-rate value that are given as parameters to the encoder.

The system also includes intra-frame estimation and prediction that are based on spatial correlation redundancies discussed previously.

The motion estimation and motion compensation algorithms depend on inter-frame temporal redundancies that utilise other frame data to compress current frame data.

The output from some of the compression algorithms is utilised in others to maximise compression, and finally all outputs are fed into an Entropy Encoder which is very similar to the 'e' and 'q' encoder mentioned previously. The most common Entropy Encoding utilises the Huffman code method to give different weighting to segments according to the statistical redundancy of the data.

It is worth drawing attention to the fact that all the functions in the system, with the exception of Quantisation, are lossless algorithms.

4.2.3. Dirac

Dirac is an umbrella project by the BBC (British Broadcasting Corporation) which released in early 2008 two codec's by the names of; Dirac Pro and Dirac 2.1. Both are open-technology and royalty free, with Dirac Pro being submitted to the Society of Motion Picture and Television Engineers with an expectation for it to become an international standard [22].

Dirac Pro is an intra-frame codec; not utilising other frame data, with an aim at lossless or near lossless compression, while Dirac 2.1 is almost a mirror image of Dirac Pro with the significant difference of being an inter-frame codec, utilising motion estimation/motion compensation (ME/MC).

The codec's are highly flexible, accepting both RGB and YCbCr, progressive or interlaced video with no restrictions on frame size, resolution or rate, the target being streaming and broadcasting applications. Dirac can also be encapsulated in MPEG transport or OGG wrappers.

4.2.4. Theora

Theora I is a codec released in December 2006 by Xiph.org Foundation, featuring royalty free and open technology and has since become quite popular online especially with large websites that comply with the Wikimedia Commons like Wikipedia.org [22].

Theora employs inter-frame encoding, utilising other frames to reduce the data size. Videos encoded by Theora must be progressive and must be in YCbCr colour format. Like Dirac, Theora is a stream of data packets, and can easily be encapsulated in many transport container formats such as MPEG transport stream and OGG.

4.2.5. Motion JPEG

Motion JPEG or MJPEG is built upon still JPEG image compression with each individual frame compressed as a still JPEG image, as such MJPEG is an intra-frame codec where each frame output is also a JPEG image [23].

MJPEG requires very low computational requirements and is easily used as most systems have the JPEG codec included [24]. MJPEG has a low compression rate however it has high quality videos output which makes its use popular in Medical imaging applications.

4.3. Wrappers

Wrappers, otherwise called container formats, are used to store different data types in one location. Multimedia wrappers are used to interlace audio and video as well as other video data such as subtitles and chapter information in a single file or stream.

The vast majority of wrappers achieve this by interlacing the different data types to ensure synchronisation and efficiency when the video is played back. The first wrapper released in 1985 was the Interchange File Format (IFF) and most wrappers in existence today are based on IFF [25].

Important aspects to consider when choosing a wrapper are shown below [26].

- **Content:** Multimedia wrappers differ in what data types they support, while the essential video-audio types are accommodated, some wrappers do not support other types such as subtitles.
- **Codecs:** Most wrappers are limited in the codecs they support, the more recent codecs, such as H.264/AVC, are usually the ones with less support.
- **Compatibility:** If widespread deployment is an essential need, a wrapper with widespread compatibility is a requirement.
- **Commercial:** Wrappers are not all free for use; care must be taken to choose a free or even open-source wrapper.
- **Streaming:** Support for streaming is mandatory if a system is expected to play video in real-time.

4.3.1. OGG

Ogg is a wrapper maintained by the Xiph.Org Foundation, as an open-standard, free wrapper and has become very popular. It supports many data types and codecs, and is typically used with the Theora codec, also maintained by Xiph.Org. [26]

Ogg supports streaming, and is widely used for this purpose; however it does not have support for the H.264/AVC codec.

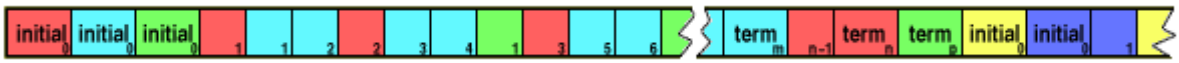

Figure 6: *OGG bit-stream showing interlacing of different data types; taken from "*Ogg logical and physical bit stream overview" *[27].*

4.3.2. AVI

The Audio Visual Interleave (AVI) format was derived from IFF in 1992. It is one of, if not the most, popular wrapper in existence today as it supported by Windows platforms by default as well as being very widely supported by most media players. AVI is however limited in its data type and codec support; subtitles and the H.264/AVC codec are not supported [27].

4.3.3. MKV

The Matroska (MKV) wrapper is an open-source, open-standard wrapper, and has seen a great increase in its popularity over the last few years. It is thought to be set to replace Ogg as it is much more flexible and supports a much wider codec base; including H.264/AVC [27].

It accommodates many data types and like Ogg supports streaming, essential for real-time video streaming applications.

More recently MKV is being supported by large electronics companies; the LG BD370 Blu-ray Player supports the MKV wrapper.

5. Design of Real-Time Video Streaming

5.1. Component Selection

5.1.1. Codec

Of the codecs explored the two most viable are the Ogg Theora codec and a H.264/AVC codec because they are both recent yet widely popular codecs.

Both also have a readily available open-source encoder that can be operated on the OLPC, Theora through the GStreamer library and H.264/AVC through the x264 encoder. Both are also flexible in terms of video interlacing and both operate in the YCbCr colour space.

Finally both are suited to video streaming applications and can be placed in wrappers allowing for audio and other functionalities in the future.

It therefore comes down to one of the most important features of codecs to differentiate between them, coding efficiency.

This is a rather complex feature to compare as there are many different tests that can produce contradicting results however I will refer to the paper entitled “Comparison of Open and Free Video Compression Systems: A Performance Evaluation” [22] which demonstrates the H.264 codec superiority and will use comparison images from the Streaming Learning Centre website [28] to demonstrate this, I will not delve into the details of the test, they can be obtained from the website aforementioned.

Figure 7: *Two comparisons between OGG Theora encoded videos (left) and H.264/AVC encoded videos (right)*

The two examples above demonstrate how at a given bitrate (800 kbps) the H.264/AVC codec is superior in the perceived quality both in terms of colour and the fact that is less blurry and has fewer artefacts.

5.1.2. Wrapper

Of the three wrappers explored, MKV was chosen primarily because it supports the H.264/AVC codec which was the codec selected for use in the system. In this respect MKV is also superior in its ability to potentially hold any codec.

MKV further supports streaming, and is becoming increasingly popular and is supported by the VLC player.

Finally MKV is a free open-standard wrapper which is also implemented in the x264 encoder used for the H264/AVC codec, which makes it more convenient.

5.1.3. Network

The protocol configuration chosen was the UDP/RTP protocols with a TCP/RTSP control session.

This configuration allows sending of important data such as session controls and video header information through the more reliable TCP connection, while the streaming itself is conducted through RTP to allow for better speed and sufficient reliability.

5.1.4. Client and Player

The client computer can be any computer irrespective of the hardware and the operating system it runs. The VLC player was therefore chosen to play-back the streamed video as it is a widely used open-source player with a big community.

Furthermore the VLC player supports the MKV wrapper and the H.264/AVC codec across different operating systems.

5.1.5. Server

The OLPC (One Laptop per Child) laptop, also known as the XO Laptop is a low-budget, low-power device [2] optimized for use in third world countries, where the scarce resources means that the life-time of the laptop battery is of more importance than its performance.

The specifications of the laptop (in the Appendix) emphasize how the OLPC laptop is a very low-end platform and will probably have a lot of difficulty in accommodating complex programs. Amongst other things the significantly low RAM and processor speed are a good indication of the general performance of the laptop.

It is worth noting that a recent HTC mobile phone running Android OS has CPU specifications greatly exceeding those of the OLPC.

The choice of the OLPC as a server is essentially for this reason; to implement a system where the server requirements are relatively low, which makes the final program much more useful because almost any application where the program will be used will have a more powerful server than the OLPC.

5.2. Encoder and decoder

The x264 encoder and decoder can be obtained from the x264 website [29] and there are links to its source-code.

x264 is cross-platform and is being used widely; Google Video uses the encoder and VLC has support for the decoder in playback.

The decoder with VLC can to a certain extent accommodate for packets lost in the network and do not terminate playback if a frame is dropped or corrupted.

5.3. Java Server and Client

The java server and client programs where not produced from scratch, rather a readily available assignment at the Computer Science and Electronic Engineering department of the University of Maryland, Baltimore County (UMBC), USA was obtained from the department website [30].

The assignment gives 4 java classes, a Server, a Client, an RTP handler and a Video Stream reader.

The Server and VideoStream classes needed no editing however the Client and the RTPpacket classes required some features to be added in order for them to become complete.

The completed program read data from an MJPEG file (also available on the website) at the server and sent this across the network one frame at a time. The client would then display this, one JPEG image at a time, to create a video output.

The advantages of the complete program were that it successfully created a UDP/RTP stream with a TCP/RTSP session, and played the video at the client successfully. The disadvantage however was in the fact that it was structured around the MJPEG codec, with no flexibility in this regard, and the video support was further limited as it only supported a proprietary MJPEG file format.

Furthermore the program did not run in real-time, as the entire video had to be read into memory before the transmission could commence, which for a video streaming application renders the program useless.

The program was enhanced over several versions, introducing several features, one of which its ability to potentially read any codec or wrapper with minimal editing of the program.

This feature was introduced in a separate class threadReadFile which is discussed in the video parser section.

Another feature of the final program was that it ran completely in real-time, both at the server and the encoder sides, this was implemented by running the video parser in a separate thread. This required further programming to ensure synchronisation of the parsed data between the several threads of the program as certain threads would remove data from memory while others would append to existing video data in memory, which in an unsynchronised scenario would cause significant memory errors and may affect the data itself.

Another feature was the use of the VLC video player at the client to play-back the received video in real-time. This was achieved by writing the received data to a local file and then launching the VLC, after a given number of packets is received, through a thread running its system command with the file path as an argument.

A different implementation for this was required in the different operating systems; the program therefore accommodates both Linux and Windows platform requirements by using a global variable to determine which of the two OS's is being run. This feature can be improved by use of pipes rather than writing to a file, and through a mechanism that automatically detects what OS is being run.

A further feature was allowing the program to run in 'live' mode. This included a function to extract the header of the video separately, so as to ensure it is sent to the client, the server would then drop frames, maintaining a given number of frames in memory, so as to limit delay when the stream is established and frames are sent.

The rationale is that if running live with a video source, the video frames obtained before the connection is established are problematic as they represent the past and will therefore introduce a delay. In a live scenario the user is only interested in sending video data obtained after the network connection is established.

This final feature however has not been tested as it requires an encoder that can run in live mode as well. Unfortunately the x264 encoder does not have this feature. The solution would be in finding an alternate encoder or editing the x264 encoder itself which is open-source and written in C++.

5.4. Video parser

The video parser was introduced as an alternative to the VideoStream class, in order to accommodate more video formats than merely the proprietary MJPEG format.

In the final version of the program the parser successfully identifies the header and most frames of a H.264 encoded video in the MKV wrapper. The parser itself is rather simple, as it looks for a given byte sequence that identifies the start and end of the packet with support for wildcards.

The challenge was in implementing this in real-time in a way that allows for complete flexibility so as to not limit the different video formats that can potentially be parsed. This was implemented by introducing a queue buffer that is updated with every byte read from the file and continually compared to the byte sequence. This enables the buffer to determine the start of a new packet when the buffer is identical to the byte sequence.

Furthermore the parser is able to ensure that packets do not exceed a certain size by splitting the data into more than one packet. This is important for the transmission as large packets cause problems for the network and are more likely to be dropped.

The byte sequence is unique to the codec or wrapper used and can also vary in length. For ultimate flexibility, the buffer size is only fixed to the current run of the program, it is not hardcoded, but dynamically set according to the size of the manually inserted byte sequence.

If a user wants to parse a different format all that is required is to alter the byte sequence to the one that correctly parses the required format in hexadecimal form, which can be obtained by researching the format byte stream. The byte sequence that correctly parses an unwrapped H.264/AVC video stream is 00,00,00,01 in hexadecimal form so if wanting to parse H.264/AVC unwrapped video, the variable that represents the byte sequence in the threadReadFile class would be set to the string “00000001”.

Furthermore if the parser is deemed too simple, its implementation in a separate class allows for it to be completely redone without affecting the way the rest of the program operates.

5.5. Final Overall system architecture

The final system architecture can be seen in the diagram below:

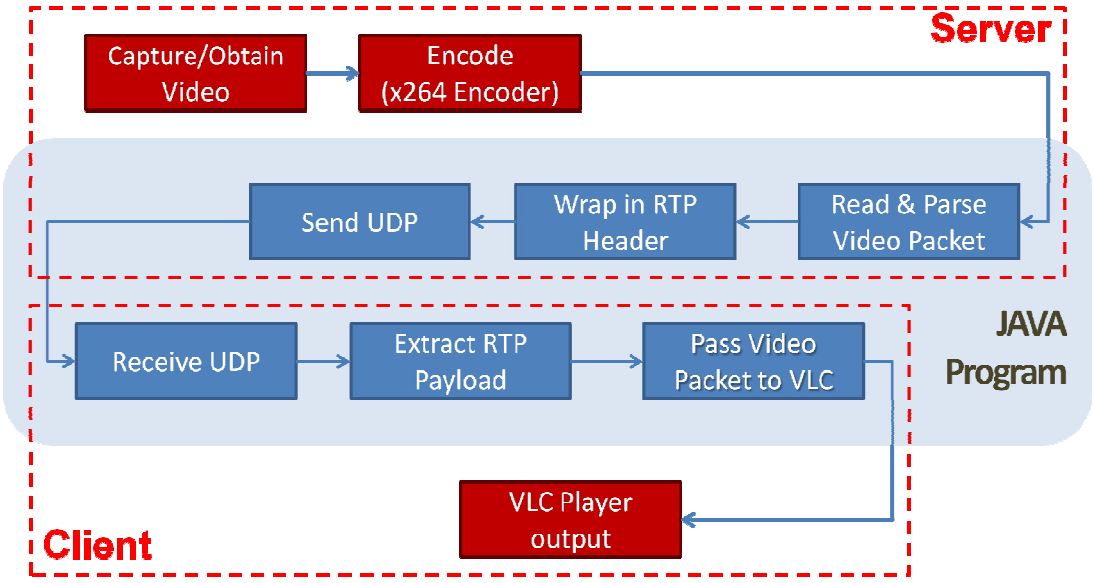

Figure 8: *The project system architecture spanning both the server and client.*

As can be seen in the diagram, the complete system includes more than just the java programs, as the encoder and video player (with the decoder) form an essential part of the system.

Live video streaming requires capturing video in real time, a rather simple operation with a variety of available options, but probably easier in Linux platforms than it is on Windows platforms.

Live video streaming also requires an encoder that can run live encoding, unfortunately x264 does not currently support this feature hence the live features of the Java program could not be tested.

Streaming pre-encoded videos in real-time however has been tested, and the results obtained are shown in the next section.

6. Experiment

6.1. Setup

The experiment hardware was setup as shown in figure 8 and figure 9, with the OLPC as the server and a HP laptop running Windows 7 OS as the client. They were connected wirelessly through a router.

The client computer runs wireshark [31] in the background, an open-source program that can be installed on both windows and linux platforms. Wireshark monitors the wireless card of the computer and logs data about the incoming and outgoing packets, with a filter feature to ensure only packets of interest are monitored.

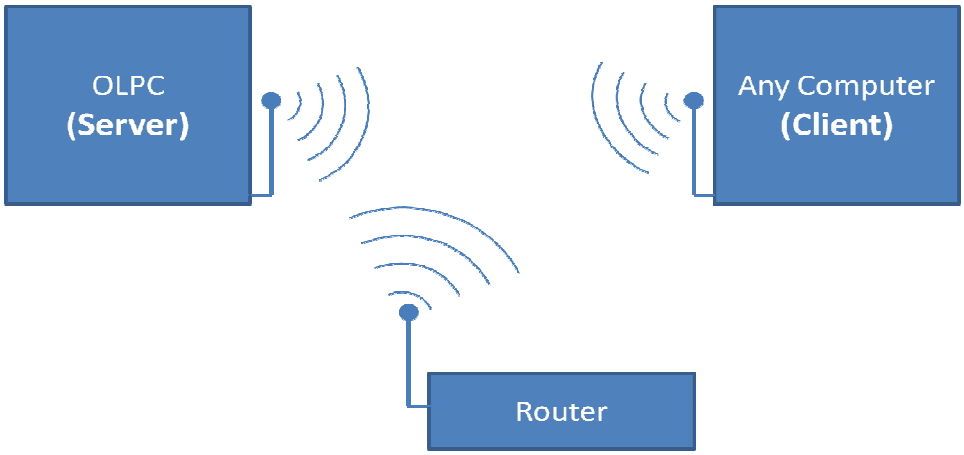

Figure 8: *A block diagram representing the system hardware setup*

The OLPC (server) runs the x264 encoder with the following line in the terminal:

x264 --crf 24 --psnr --demuxer yuv --muxer mkv -o foreman.mkv foreman.cif 352x288

This instructs x264 to work with a quantisation parameter (QP) of 24, to display the PSNR data of the encoded file, to work with a YUV input file, to wrap the decoded stream in an MKV wrapper and to set the output filename to “foreman.mkv”. It further takes the input video file path and the resolution of the input video (in this case a CIF video) as arguments.

This produces the MKV wrapped H.264/AVC encoded video file which will then be streamed over the network.

The java server is then run with the following command in the terminal:

java Server 1050

This instructs the java program to run the TCP/RTSP session on a connection at port number 1050, the UDP/RTP stream is not in need of a connection.

Figure 9: *A picture showing the real hardware setup with the OLPC as the server, a HP laptop as the client and a netgear router.*

The client computer then needs to run the java client with the following command in the terminal (called the 'command prompt' in windows):

java Client 192.168.0.2 1050 foreman.mkv

This instructs the java program to connect to port number 1050 on the network computer represented by the IP address, and to request the file 'foreman.mkv'. The program also supports use of the server name as an alternative to its IP address.

The java client opens a GUI with buttons to control the RTSP session, the relevant buttons are 'setup', 'play' and 'terminate'. The setup button is pressed, which sets up the RTSP session and sends important data between the two computers such as the

video stream header. The terminal on both computers is also updated to say that the session is ready.

The play button is then pressed which starts the packet transmission, the terminals are updated with the packet number, an extract of the beginning of the packet in hexadecimal form, and the size of the packet, straight after the packet is sent.

Once a certain number of packets arrive, the client program launches the VLC player which begins playback of the video.

Once the video finishes, the terminate button is pressed which shuts down the program on both the server and the client computers.

The Wireshark data capture can be stopped and the data saved to a file. The data can also be exported in the more popular CSV format.

The result graphs in the following section use the PSNR data produced by the x264 encoder at the server, the packet sizes output in the terminal by the java program and the time that packets arrive at the client as observed by wireshark.

The experiments were conducted on 3 different videos each with 2 different quantisation levels, for each one a VBR and CBR encoding was performed.

Each experiment was performed 5 times and an average taken in an attempt to prevent anomalies.

6.2. Results

6.2.1. Visual

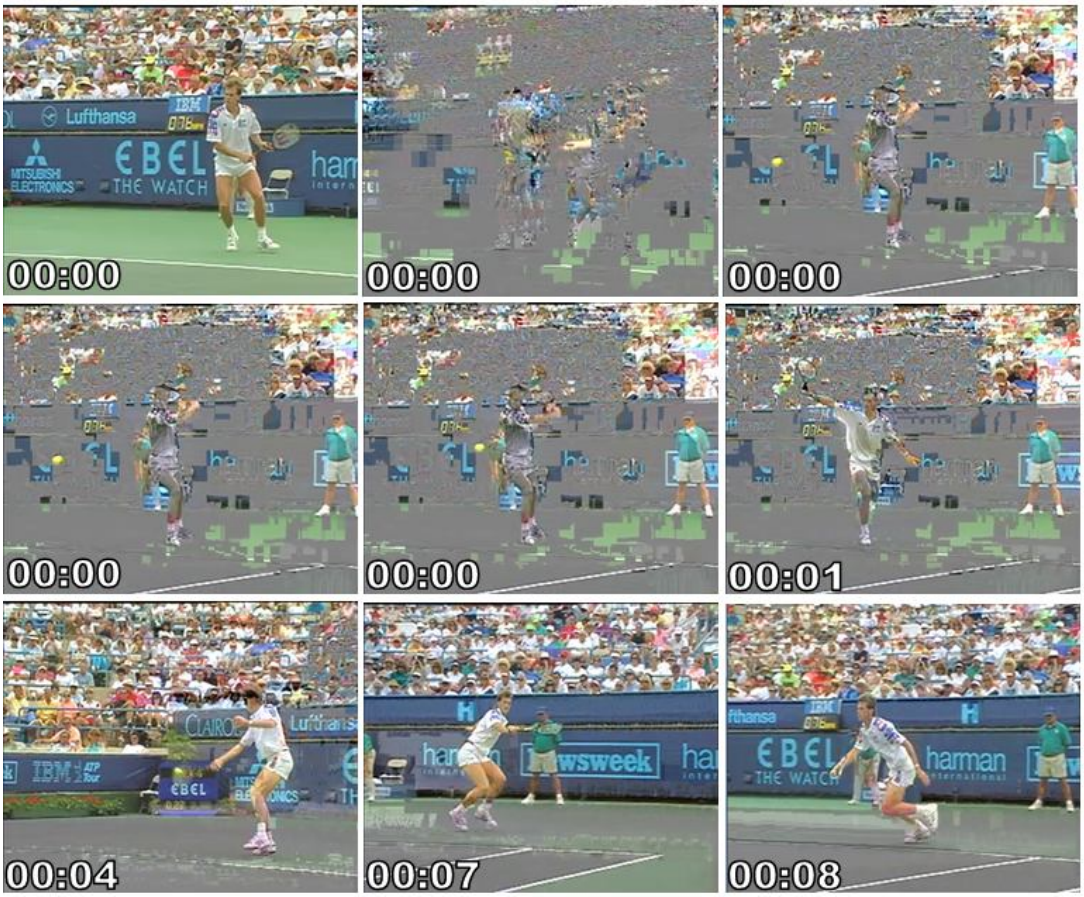

Figure 10: *Screen dumps of 9 frames exhibiting distortion caused by a dropped i-frame.*

The 9 frames in figure 10 show the playback when an i-frame is dropped. The VLC manages to continue playback, however due to the consecutive frames depending on the i-frame for data, the frames in question also come up distorted. This distortion remains in effect for a number of seconds.

It is important to note how the frames from 3 to 6 have clear groups of pixels where there is a lot of motion, which lends to the fact that the motion introduces data which the frames in question contain irrespective of the I-frame data.

Figure 11: *Distortion in the video at the client (right) as compared to the same frame in the encoded video at the server (left).*

The comparison in figure 11 shows the result when a b-frame is missing or corrupt. Unlike the case with the missing i-frame, the video data is displayed; however the motion vectors are being applied to the wrong reference frame which produces a frame with the pixel data severely misplaced.

Figure 12: *Artefacts in the video encoded with QP40 (right) as compared to the same frame in the original YUV video.*

The comparison in figure 12 shows the effect of quantisation on the encoded video. Artefacts are introduced around objects especially those that are moving at a high speed.

6.2.2. Quantative

The graphs below show the traffic volume for 3 videos each with low and high quantisation. The traffic is compared for Variable Bit Rate (VBR) and Constant Bit Rate (CBR) encodings of the video and the PSNR value for each is shown in the bottom left corner.

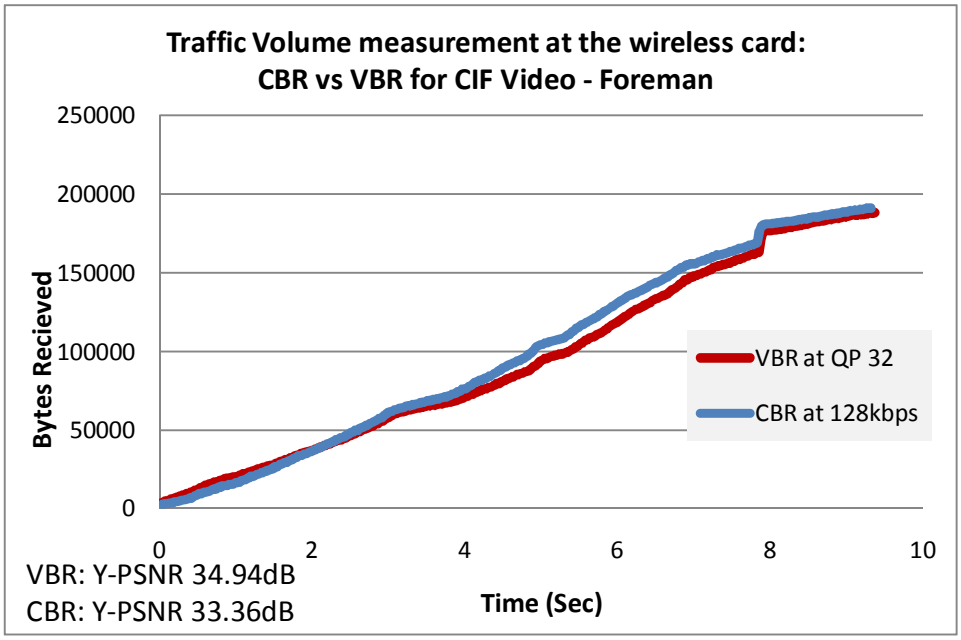

Figure 13: *Graph of traffic volume for video foreman at the client with high quantisation*

The graphs in figure 13 and figure 14 are for a video of a man speaking to the camera, towards the end of the video the camera is turned around to show a building. This motion can be seen in both graphs where a large amount of data is sent over the network in a short time.

The VBR and CBR videos both seem to be rather similar in their packet sizes and therefore have similar transmission times through the network.

The PSNR values indicate that at both quantisation levels the CBR encoded video is of lower quality.

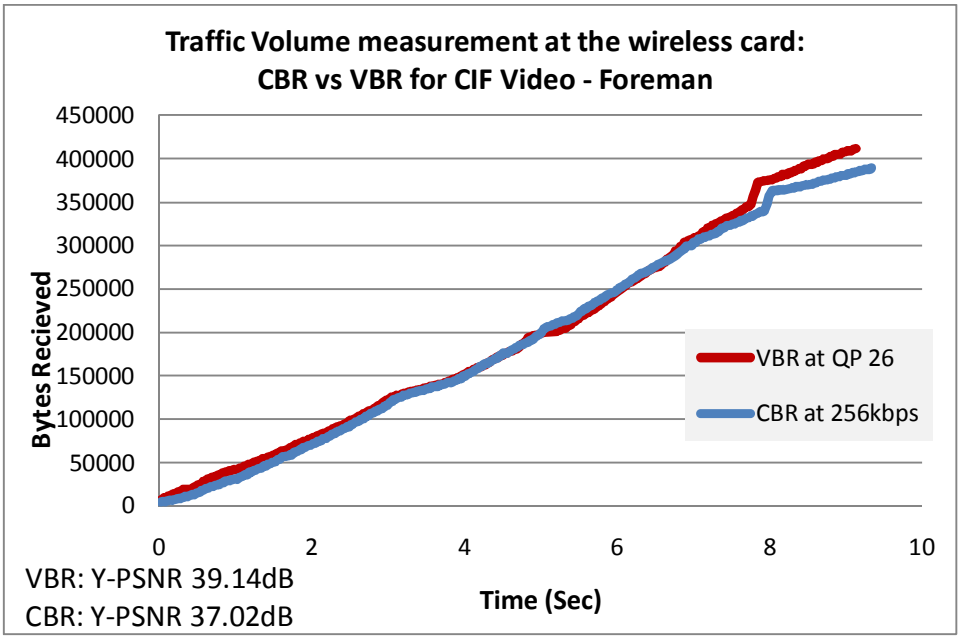

Figure 14: *Graph of traffic volume for video foreman at the client with low quantisation*

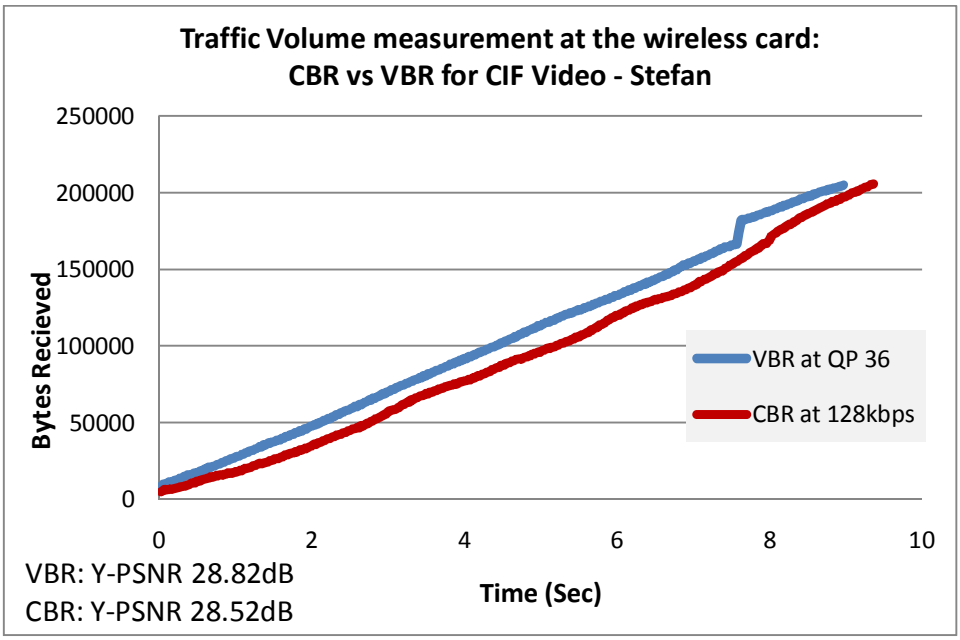

Figure 15: *Graph of traffic volume for video stefan at the client with high quantisation*

The graphs in figure 15 and figure 16 are for a video of a tennis player who constantly darts across the screen to hit the ball, the camera also moves constantly to keep the

player in focus. The rate of motion is rather linear which can be observed from the graphs.

The VBR and CBR videos seem to have slight differences in the packet sizes with the VBR videos having larger packet sizes. This also causes the VBR video transmission to end before that of the CBR video.

The PSNR values indicate that at both quantisation levels the CBR and VBR encoded videos are of similar quality.

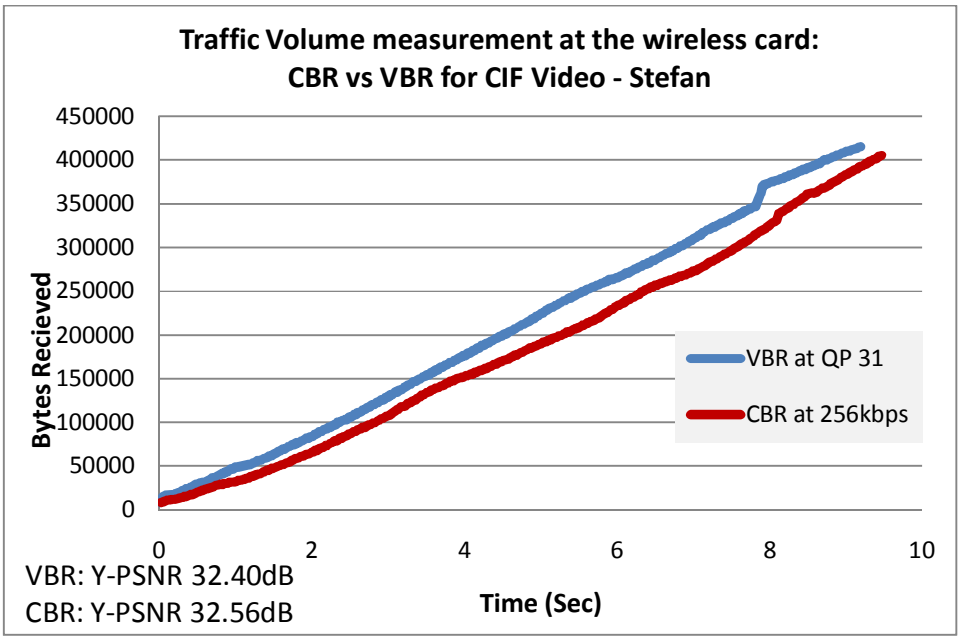

Figure 16: *Graph of traffic volume for video stefan at the client with low quantisation*

The graphs in figure 17 and figure 18 are for a video with many different moving objects in many different colours, the camera is also in motion. Towards the end of the video a new object with many different moving parts comes into the frame. This can be observed in the graph where a lot of data is sent rapidly over the network.

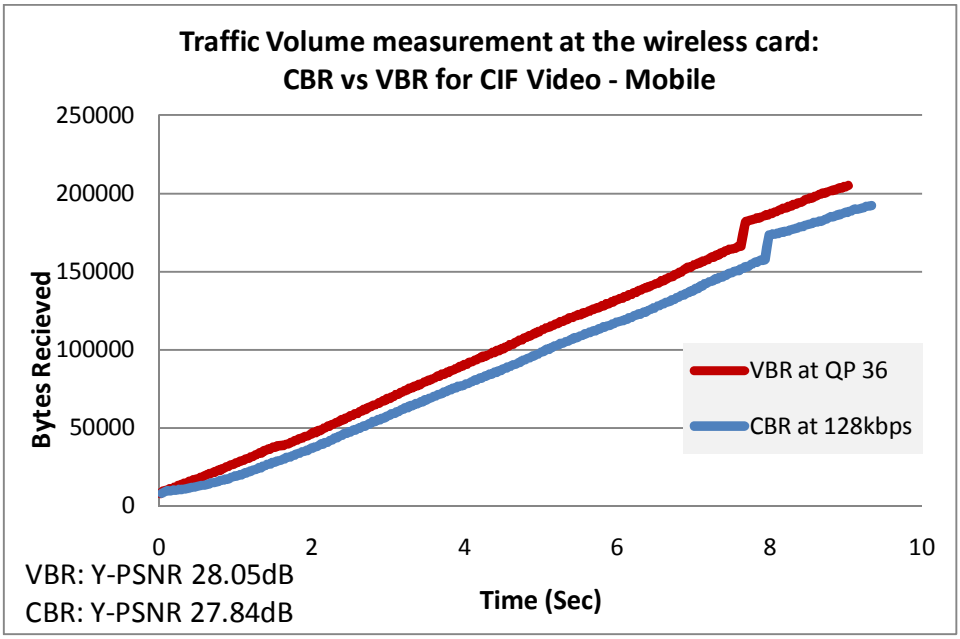

Figure 17: *Graph of traffic volume for video mobile at the client with high quantisation*

The VBR and CBR videos both seem to be rather similar in their packet sizes and therefore have similar transmission times through the network.

The PSNR values indicate that the encoded videos in CBR and VBR have a similar level of quality.

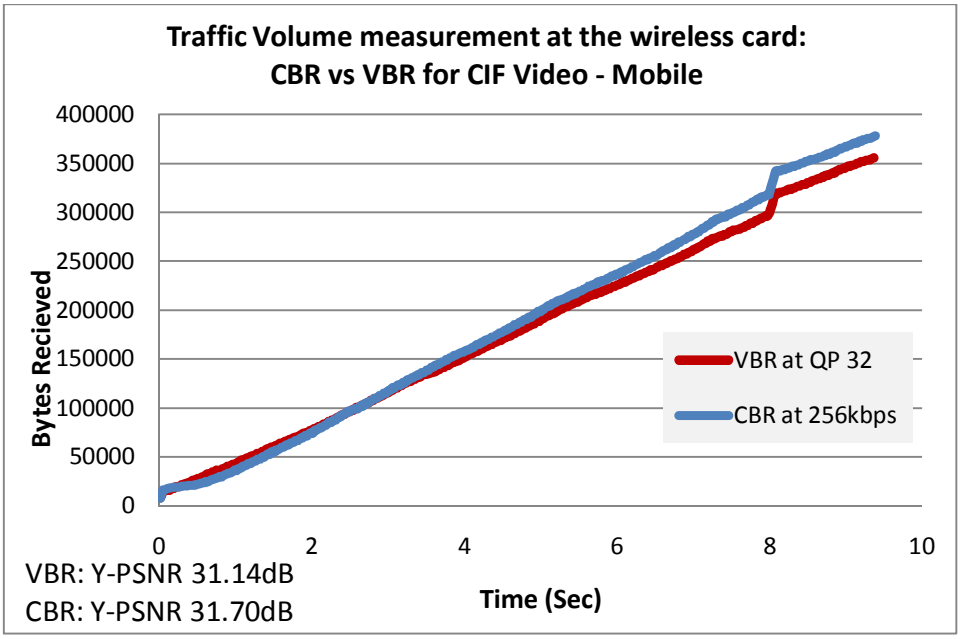

Figure 18: *Graph of traffic volume for video mobile at the client with low quantisation*

6.3. Current issues to be solved

The java parser can be improved to more accurately detect the video packets (frames) for a more robust RTP encapsulation. This will ensure that big packets will be less frequent as this can arise from the incorrect parsing of two or more video packets as one.

The server video capture and encoding can be refined to achieve live video streaming which the java program supports. This can be done by obtaining a different H.264/AVC encoder which supports live encoding, or by editing the x264 encoder which is open-source and written in C++.

The interfacing with VLC can be improved to obtain updates on the position of the seeker to send commands to pause and resume playback in a way that ensures there is always video data to be played back.

The server can also be refined to send RTP packets more frequently or less frequently according to the size of packets and signals from the client depending on the video player seeker position.

The server can also be programmed to support more than one simulations RTP/RTSP connection to allow a broadcasting service which becomes especially important when the live mode is used.

Audio can be added to the video stream, this is supported by the MKV wrapper and the video player.

7. Conclusion

A tool was proposed for RTP/RTSP video streaming with the H.264/AVC codec and the Matroska (MKV) wrapper, a configuration which utilises non-conventional components that are increasing in popularity and serve the objectives of the project.

A measurement framework was proposed using third-party open-source software and results obtained about RTP/RTSP sessions corresponding to different videos, quantisation levels and bit-rates.

The main objective of the project was to create a real-time video streaming application for the OLPC laptop that is able to run on the low-end platform while achieving high quality video playback.

The outcome is an open source implementation allowing for full customisation of streaming parameters of the x264 encoder as well as the RTP configurations, the program parameters (such as read buffer size and maximum packet size), and ending with the fully customisable interface of the VLC player.

Customisation for use with other codec and wrapper configurations is discussed and steps given to achieve this which allows the program to remain useful if other configurations are required.

Having successfully run on the OLPC there is scope to expand the program use to more relevant everyday devices such mobile phones running the linux-based operating systems.

The program has potential for many improvements in the way it runs. There is also scope for further development by including audio in the stream, to achieve a complete multimedia solution. It can further be developed by introducing the ability for the server to broadcast the video stream, and to have a packet and connection prioritisation framework.

The live functions of the program can be tested once a live encoder is obtained. This introduces many new challenges to the program that remains untested in this regard, but also possibilities for reliable interactive applications over networks such as the internet.

8. References

[1] John G. Apostolopoulos, Wai- tian Tan and Susie J. Wee, "Video Streaming: Concepts, Algorithms, and Systems", *Mobile and Media Systems Laboratory, HP Laboratories Palo Alto,* Chapter 4, 2002. Available: http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf

[2]http://laptop.org/en/vision/index.shtml last accessed 22/03/2010

[3]http://www.android.com/ last accessed 23/03/2010.

[4]Barry Leiner, Robert Cole, Jon Postel, David Mills, "The DARPA Internet Protocol Suite" *IEEE Communications Magazine*, Vol.23, No.3, March 1985.

[5] Adapted from http://en.wikipedia.org/wiki/File:IP_stack_connections.svg as last accessed 22/03/2010

[6] Joe Casad, Sams Teach Yourself TCP/IP in 24 Hours, USA, Edition 3, 2004, Pg 26

[7] http://www.ietf.org/rfc/rfc0793.txt , last accessed 22/03/2010

[8] Candace Leiden and Marshall Wilensky, "TCP/IP for Dummies", *Wiley Publishing Inc.*, Indiana, USA, 2009, Edition 6, Chapter 6

[9] Achyut S Godbole, "Data communications and networks", New Delhi, India, Edition 7, 2007, Pg 378

[10]http://www.faqs.org/rfcs/rfc768.html , last accessed 22/03/2010

[11] Chunlei Liu, "Multimedia Over IP: RSVP, RTP, RTCP, RTSP", *"Handbook of emerging communications technologies: the next decade", CRC Press, Inc.,* Boca Raton, 1999. Available: http://www.cse.wustl.edu/~jain/cis788-97/ftp/ip_multimedia.pdf last accessed 23/03/2010

[12] Abdul H. Sadka, "Compressed video communications", West Sussex, UK, 2002, Pg 191

[13] http://www.ietf.org/rfc/rfc2326.txt , last accessed 22/03/2010

[14] Keith Jack, "Video demystified: a handbook for the digital engineer", MA, USA, Edition 4, 2005, Pg 832

[15] Markus Hofmann, Leland R. Beaumont, "Content networking: architecture, protocols, and practice", San Fransesco, USA, 2005, Pg 87

[16] Teresa Liew Bao Yng, Byung-Gook Lee and Hoon Yoo, "Low Complexity, Lossless Frame Memory Compression Using Modified Hadamard Transform and Adaptive Golomb-Rice Coding", *IADIS International Conference Computer Graphics and Visualization 2008*, pp. 89, 2008.

[17] Iain E. G. Richardson, "H.264 and MPEG-4 video compression: video coding for next-generation multimedia", *John Wiley and Sons*, chapter 3.1, 2003.

[18] Lajos Hanzo, Peter Cherriman and Jurgen Streit, "Video Compression and Communications", *Wiley-IEEE*, chapter 1, 2007.

[19] S. Takamura, "Lossless Video Coding," Lecture of the Course EE398B on Image Communication at Stanford University, 04/2007. Available: http://www.stanford.edu/class/ee398b/handouts/lectures/LosslessVideoCoding.pdf

[20] Prof. Mihaela van der Schaar, "Multimedia networking and communications", ECE 289J, Lecture 1 powerpoint presentation. Available: http://www.ee.ucla.edu/faculty-van-der-schaar.htm

[21] Gary J. Sullivan and Thomas Wiegand, "Video Compression – From Concepts to the H.264/AVC Standard," Proceedings of the IEEE, Vol.93, No.1, January 2005.

[22] Till Halbach "Comparison of Open and Free Video Compression Systems: A Performance Evaluation", *Norwegian Computing Center*, 2009. Available: http://etill.net/projects/dirac_theora_evaluation/include/halbach-2009-dirac_theora-paper.pdf

[23] Zhenyu Yang, Yi Cui, Zahid Anwar, Robert Bocchino, Nadir Kiyanclar, Klara Nahrstedt and Roy H. Campbell, "Real-Time 3D Video Compression for Tele-Immersive Environments", *Department of Computer Science - University of Illinois at Urbana-Champaign,* 2006. Accessible: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.5574&rep=rep1&type=pdf

[24] D.T.Vo and T.Q.Nguyen: "Quality Enhancement for Motion JPEG using Temporal Redundancies," *IEEE Transactions on Circuits and Systems for Video Technology,* Vol. 18, No. 5, May 2008. Accessible: http://videoprocessing.ucsd.edu/publications/Year_2008/Vo_2008a.pdf

[25] Michael Streatfield, "Bringing it All Together: A Comparison of Two Modern Multimedia Container Formats", *Electronics and Computer Science, University of Southampton*, 2006.

[26]Tim De Pauw, "Development of a Streaming Video Platform for Educational Purposes", *Hogeschool Gent*, June 2008. Accessible: http://pwnt.be/thesis/files/thesis.pdf

[27] Taken from: "Ogg logical and physical bit stream overview" http://www.xiph.org/ogg/doc/oggstream.html last accessed on 23rd March 2010.

[28] http://www.streaminglearningcenter.com/articles/ogg-vs-h264---round-one.html as accessed 22/03/2010.

[29] http://x264.nl/ last accessed 22/03/2010.

[30] http://www.csee.umbc.edu/~pmundur/courses/CMSC691C/lab5-kurose-ross.html last accessed 22/03/2010.

[31] http://www.wireshark.org/ last accessed 22/03/2010.