
# Video Quality of IPTV over Residential Networks

Martin Conaghan

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Masters project proposal

23rd March 2012

# Contents

# List of Tables

# List of Figures

# 1 Introduction

IPTV systems are growing in number and popularity. These systems stream video to a number of IP-capable devices, typically dedicated set-top boxes, and are intended to replace the existing television broadcast networks which use satellite, cable or radio waves to broadcast many video streams. One of the key differences between IPTV and traditional TV is that IPTV typically unicasts or multicasts specific content to individual receivers rather than broadcasting all content to all receivers. Moreover, IPTV uses a two-way communication channel. Finally, IPTV is delivered over a packet-switched network that is unreliable and where the delivery time varies considerably.

One of the major factors in the success of IPTV is ensuring that the user experience matches or exceeds the user experience of traditional TV. The challenge in ensuring that user experience is good enough lies in the fact that IP networks are subject to packet loss and variable packet latency. Packet loss means that IPTV systems have to cope with the fact that some parts of the video are lost in transmission, and ideally conceal this from the user. Variable packet latency means that IPTV systems cannot rely on all packets being delivered within a fixed time bound. Some video streaming systems can accrue a large buffer of packets at the receiver before beginning playback so that a high level of transmission delay variance can be tolerated. Such systems suffer from high overall latency - the user must wait for the buffer to fill before video playback begins. In most IPTV systems the overall latency - the time between the video being sent and played - must be small, typically less than half a second. This is necessary to keep IPTV transmissions in synchrony with traditional television and to make channel changes quick. In such systems a packet with a high delay has the same effect as a lost packet.

There are mechanisms for coping with lost packets in data streams. One such mechanism is forward error correction (FEC), where the data that is transmitted is combined with extra information to allow reconstruction of lost packets in certain circumstances. There exist various competing FEC techniques, some of which are designed for different data stream types. While FEC is useful in that it can allow recovery of lost data, it requires extra information to be sent, and therefore either increases the total amount of data or decreases the amount of real data which can be sent. There are other ways to cope with packet loss in video streams. For example, some video codecs (e.g., H.264) allow duplicate information to be included in a stream, and allow network packets (e.g., RTP) to be constructed in such a way that individual packet loss has a minimal effect on the received video. It is therefore important to evaluate FEC schemes to ensure that the extra data being sent is more effective than either not using FEC or using an alternative strategy for countering packet loss.

With IPTV growing in popularity as a commercial service (for example BT Vision in the UK), it is in the interests of the IPTV providers to evaluate the performance of IPTV systems in terms of the video quality which users experience. This may be done in real-time, in an attempt to detect and fix problems before they cause customer complaints, or it may be done in a controlled environment in order to assess the expected performance of IPTV systems or to develop better standards and protocols. The performance of an IPTV

system can be measured based on the network characteristics - i.e. the number of packets dropped or arriving too late to be played, and the overall latency of the system. While this is very common and useful, the users of an IPTV system are aware only of the video quality, not the network performance. As such, there is a need to evaluate IPTV systems in terms of the degradation of video quality between the video which is transmitted and the video which is played to the user. This is difficult to do in real-time, because most video quality assessment (VQA) metrics involve comparing two videos, and in an IPTV system the two videos to be compared are never in the same place - one is on the IPTV provider's servers and the other is reconstructed at the users' devices.

This project will build on previous research [25, 26, 27], which has measured the ability of residential ADSL and cable networks to carry real-time video data. In particular it will investigate the effect FEC algorithms have on such real-time data; whether they can be used to repair lost packets and if so which mechanisms work the best and at what cost. Ellis et al. [27] investigate the effect of several commonly used FEC schemes using packet traces from real residential networks, considering the ability of these schemes to correct packet loss. While packet loss patterns are useful in evaluating FEC schemes, there is a need to evaluate the schemes in terms of the resulting video quality, because ultimately consumers of IPTV services are interested in the video quality they experience and are oblivious to the packet loss and delay variation that their IPTV device experiences.

This project proposal outlines an experiment which will investigate the extent to which video quality is degraded when transmitted via an IPTV service delivered over a residential network. The experiment will encode sample video using open source video encoding libraries, and simulate sending the data over a residential network. The video data will be decoded as if it were being received at an IPTV device and the quality of the received video will be compared to the transmitted video. The network simulation will be based on packet traces gathered from real residential networks [25, 26].

The structure of this report is as follows. Section 2 gives an overview of IPTV systems and a high level description of how they typically work. Section 3 states the problem which this project will address. Section 4 reviews relevant literature, including the research upon which this project builds, other research on IPTV performance and research on VQA metrics. Section 5 describes the proposed approach for the project, including the experiment set-up and the test cases which will be run. Finally, section 6 outlines the anticipated plan for completing the project.

# 2 Background

This section provides background information on IPTV, including common approaches and concepts.

## 2.1 Video Streaming

This sub-section provides a high level overview of the process of streaming a video in the context of IPTV, with the intention of introducing key concepts and terminology.

First, the IPTV supplier has a video to stream. The video may be live data coming straight from a video camera, or it may be a pre-encoded video stored in a standard container format such as MPEG-4[6]. Raw video is stored as a bit map of pixels, where each pixel is stored either as its constituent RGB values or as luma and chroma (brightness and colour difference) values. There are many codecs which can be used to encode raw video, and nearly all of them are lossy. Two of the most common are MPEG-2[7] and H.264[4]. Both of these codecs (and many others) store a video as a sequence of encoded frames, where each frame may be an intra (I), predicted (P) or bi-directional predicted (B) frame. Intra frames (which are also known as key frames) can be decoded on their own. P and B frames require the decoding of other frames. Encoded intra frames are usually much larger than P or B frames. The order of I, P and B frames is set as a group of pictures (GOP). An example of a GOP is IBBPBBPBBPBB, which means that a standalone intra frame is only transmitted once every 12 frames.

The IPTV supplier then breaks the encoded video stream into packets which can be sent across a network. If the H.264 video encoder is used to encode the video and RTP[11] is used for transport, then RFC 6184 [16] (previously RFC 3984 [13]) specifies that the video stream be broken into a number of Network Abstraction Layer Units (NALUs). Each NALU is a packet consisting of a single header byte (the first byte) and payload. This process will be discussed in more detail in section 4.6.

Next, the packets are encapsulated so that they can be sent over an IP network. There are two common approaches, the first is to put the data straight into UDP packets and the second is to put the data into RTP [11] packets, which are put into UDP packets. The structure of a packet created with the latter technique[1] is shown in figure 1. Using RTP has various advantages over standalone UDP, such as detecting packet loss and allowing out-of-order packets to be re-ordered at the receiver, but introduces a small overhead.

After this the IP packets are sent across an IP network, most probably the Internet. Some packets may be corrupted or dropped by routers. There may or may not be some re-transmission mechanism to recover lost packets, however waiting for retransmissions will increase the overall latency of the stream and typically IPTV systems cannot afford this (and therefore must survive without retransmissions). Packets may be delivered out of

[1]The diagram depicts RFC 6184 using a single NALU payload, see section 4.6 for more detail

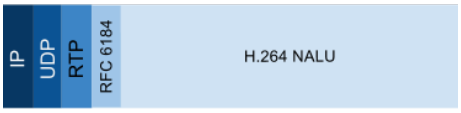


Figure 1: RFC6184 H.264 packet encapsulated using IP, UDP and RTP

order, and the receiver may or may not be able to correct this (using RTP allows this, for example).

Typically when streaming, approximately 1 second of video is sent every second. This does not necessarily mean every packet or frame is the same size; as previously mentioned intra frames take much more space than predicted frames. Depending how important a constant bit rate is, the encoder can vary the level of detail or compression. Moreover, the large intra frames can be split over many packets, if the technologies been used allow this (H.264 and RFC 6184 do allow this). Typically IPTV providers seek constant bit rates because they bring predictability. Streaming contrasts with a file download where the transmission rate is not fixed. Some video streaming services such as BBC iPlayer use TCP rather than UDP (and therefore can make no guarantees about bit rates), but these are not real-time streaming services; they are a cross between downloading and streaming (and are known as Internet Television rather than IPTV).

The IP packets are received at the consumer's device, for example a set-top box. Because of the inherent property of the Internet (and any packet-switched network) that transmission times are not constant, the device will almost certainly implement some form of playout/jitter buffer; the device will wait a short period after the first packet is received before starting to play the video. There is a trade-off with the size of this buffer: the larger it is the more transmission time variance that can be tolerated, but the longer it takes to change channel and the larger the storage requirements. Broadly there are two types of playout buffer; static buffers are a fixed size, while dynamic buffers change size in response to network behaviour. Playout buffers are discussed in more detail in section 4.5.

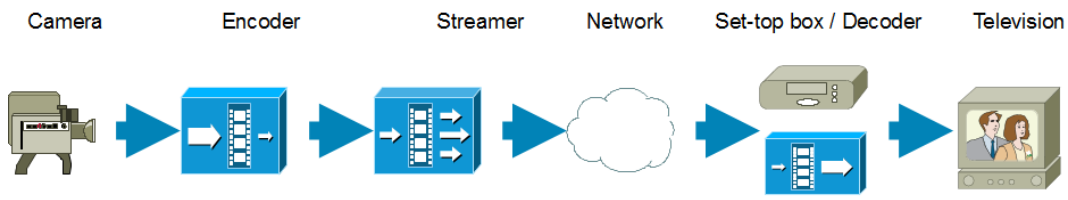


Figure 2: Overview of video streaming

Finally the consumer's device will decode the payload within the UDP/RTP/IP packets back into raw video, and display it on the playback device, which may be a television or a computer monitor. Due to the original (lossy) encoding of the video and possible lost/re-ordered/delayed packets, the video which is played will almost certainly have lost some of

the information in the original video. The challenge for IPTV providers is to minimise this loss of information, such that the user who is watching the video cannot detect it.

## 2.2 IPTV Motivation

IPTV offers several major advantages over existing television broadcast networks [37]. The communication is two-way, allowing the receiver to send feedback, and allowing interactive services to be deployed. Moreover, only the desired content (e.g., channel) need be sent at any one time, rather than always sending all the content. In addition, providers can manage their content more tightly: they can introduce more personalised billing models and can measure content popularity much more accurately. Finally, consumers can control access to the content they receive. For example parents can disable channels unsuitable for children

The main challenge for IPTV suppliers, if they are to attract and retain consumers of their services, is to ensure excellent QoS for the video and audio data which is sent [46], while maintaining suitably low latencies between the time video is send and viewed, for example to ensure that channel changes are smooth.

## 2.3 Classifying IPTV

IPTV services can be classified in several different ways, this section will explain the most common.

### 2.3.1 IPTV versus Internet Television

First it is important to differentiate IPTV from Internet television. IPTV services are more sophisticated, and typically use digital rights management (DRM) and content and billing management [23]. Internet television is used to describe services such as BBC iPlayer[2] or 4 On Demand[3] which stream television content on demand over the top of an existing network with little effort to guarantee QoS or monitor usage. An example of an IPTV service in the UK is BT Vision.

### 2.3.2 Dedicated Network versus OTT

IPTV can be delivered either over a dedicated network, or over-the-top (OTT) of an existing network. In practise OTT delivery is common; new wired metropolitan area networks (MANs) and wide area networks (WANs) are very expensive to lay, and existing networks

[2] `http://www.bbc.co.uk/iplayer/`
[3] `http://www.channel4.com/programmes/4od`

are much cheaper to use. The disadvantage of using an existing network is that the IPTV service has to contend with all of the other traffic being delivered. In a dedicated network suppliers can rely on guaranteed bandwidth and throughput and can more confidently agree to providing certain levels of QoS. Some Internet service providers (ISPs) such as BT in the UK use their own networks to deliver IPTV services, but continue to use the networks for other traffic too. There is a compromise between dedicated networks and OTT, where existing networks are used but traffic shaping or differentiated services (defined in RFC 2474 [10]) are used to prioritise video traffic in an attempt to guarantee low latency. In practise IPTV can be classified by whether or not the IPTV provider is the network provider or has a service level agreement (SLA) with the network provider to guarantee some level of QoS.

### 2.3.3 Live versus Pre-recorded Video

As with existing television networks, IPTV can transmit video which is pre-recorded or live (transmitted within seconds of being captured). Sending pre-recorded video gives the IPTV provider more time to put more effort into encoding the video. The extra time may allow the video to be encoded to a more constant bit rate for example allowing the large intra frames to be split over several packets and the encoding quality to be tuned more precisely. Moreover, a pre-recorded video could be encoded at various different compression rates, giving several streams with different sending bit rates. The provider could then switch to a lower bit rate in reaction to network congestion, or send different streams to different end points based on different levels of service or device resolutions. Strictly speaking live television could be simultaneously encoded at different rates, but this is more difficult.

### 2.3.4 Scheduled versus Non-scheduled

Within IPTV, content can be split into that which is scheduled to be broadcast at some pre-determined time (as with traditional television), or that which does not have a set broadcast time, such as video on demand (VOD). Scheduled video may be live or pre-recorded, where as non-scheduled is always pre-recorded. The reason this distinction is important is that IPTV providers face greater challenges when sending scheduled content, which must be played to users with only a small delay from the transmission time (typically less than half a second). To illustrate, consider that two people watching the same TV channel at the same time expect to see roughly (within a second or so) the same part of the video at the same time - if one of the users happens to be watching via IPTV is is not acceptable that the content be slowed down as a response to increased network transmission delay or packet loss. Consider what would happen if the user changed channel and then back again: they may miss a section of the video stream if they were previously many seconds behind the transmission time. While the IPTV provider can send video ahead of schedule (for pre-recorded, but not for live) and store it in a buffer at the playback device (if the network allows this), this is potentially wasteful because when the user changes channel the contents of the buffer will be discarded.

Conversely, sending pre-recorded, non-scheduled video is a little easier, as the supplier can send the video as fast as possible, in the extreme case sending the whole of the video stream before playback begins, thus ensuring a seamless user experience. Of course such services still suffer from network packet loss and jitter, but they have more flexibility in overcoming these issues. There are many services which provide non-scheduled pre-recorded video, such as BBC iPlayer or 4oD. These are Internet television services rather than IPTV, and typically these suppliers embed the video in HTTP(S) over TCP; because delay is not so important TCP retransmits which prevent data loss may be acceptable.

## 2.4 Security

There are several driving factors for providing some security in IPTV. Perhaps the main factor is that IPTV often carries video which has been copyrighted, and while the users may have the rights to watch the content they probably do not have the right to store (and re-distribute) it. In addition to the users of IPTV systems, IPTV streams could be intercepted by other nodes in the transmission network, potentially allowing unauthorised users to consume IPTV services. And even if other network nodes could only see and somehow not consume the video, this is potentially a breach of the user's privacy; users likely will not want others to know what they are watching.

Thus suppliers must attempt to prevent unauthorised users to consume the video, and prevent users from reconstructing video streams into digital copies rather than or as well as viewing the content. In practise this means encrypting the video stream (often referred to as scrambling in the context of television [28]), or using secure protocols such as SRTP [12]. Preventing users from being able to reconstruct the video, even with the use of encryption, is however very difficult however as users could still screen scape[4] their playback device - this cannot realistically be prevented.

Typically IPTV providers use established (video) encryption algorithms such as DVB's Common Scrambling Algorithm, some variation of Digital Encryption Standard (DES) or some flavour of the AES128 standard [28]. Having the overhead of encrypting and decrypting content adds to the challenge of keeping the overall latency as low as possible.

[4]Where the contents of a screen are recorded, for example by making a copy of the display buffer

# 3 Statement of Problem

The problem that this project will investigate is the extent to which the quality of video is degraded when transmitted over an IPTV system. Additionally, the project will investigate whether the use of FEC improves the quality of video transmitted over an IPTV system. Included in this will be a comparison of different FEC schemes to find which is the most effective. If there is time the project will compare FEC with other packet loss coping strategies, such as those built into the H.264 codec. Again, if there is time, the effect of other factors which may influence video quality in an IPTV system will be assessed.

The project will simulate an IPTV system which sends pre-recorded, scheduled, unencrypted video over the top of an existing residential IP network with no preferential treatment over other network traffic. The project will measure the difference between the quality of the video which is sent and the video which is played to the users of the system. The simulated network performance (packet loss patterns and packet delay) will be based on previously observed behaviour of real residential networks [25, 26]. The video will be pre-recorded as this is currently more common than live video in IPTV (and traditional TV) content. The video will be scheduled because there is a much greater challenge with scheduled content than with unscheduled, and because unscheduled content is typically sent reliably over TCP, and the network traces upon which this project is based are UDP. The video will be unencrypted because this makes very little difference to the video quality; the only effect is adding a small amount of time to decrypt the frames before decoding, which increases the overall latency (and may make a packet too late to be decoded and played). The simulated IPTV system will transmit video over the top of an existing network without any preferential treatment over other traffic because these are the characteristics of the network where the packet traces were captured.

There will be two major outputs of the project. The first will be a dissertation documenting the findings of the project, including video quality measurements for different packet traces and FEC schemes and recommendations for handling packet loss and variable packet delay in IPTV systems. The second will be software which, given a video file and a packet trace describing a network, will simulate the transmission of the video over the network, including encoding and decoding the video, and will give a numerical score (the output of one or more VQA metrics) depicting the difference in video quality between the video which is sent and the video which is played to the user.

# 4 Literature Survey

This section reviews existing literature. The section is split into several sub-sections. The first presents the research upon which this project builds, and the second discusses other research which evaluates the performance of IPTV systems. Sub-section 4.3 touches on previous work regarding the use of FEC in an IPTV context and sub-section 4.4 gives a very brief overview of the literature on video quality assessment techniques. The penultimate sub-section covers the main approaches to playout buffering and the final section discusses video codecs and standards.

## 4.1 Preceding Research

To date there are three published (or accepted to be published) papers which analyse the network traces which this project will use. The first [25] finds that packet loss levels are generally low, and re-ordered packets are extremely rare. Analysis of packet loss patterns shows that FEC is required to combat non-congestive packet loss but that simple FEC alone will not repair all packet loss. Specifically it is suggested that simple FEC will not handle congestive packet loss which was observed to occur in bursts (after router buffer overflows), which is a finding repeated in [26]. Overall the conclusion is that IPTV delivered over the top of existing residential networks is entirely feasible, especially if the IPTV traffic was given preferential treatment (which it was not in the network traces, but is common practice in IPTV systems). Subsequent research [26] analyses the packet delay finding that the distribution does not show tight bounds and therefore IPTV systems cannot choose the size of the playout buffer based on the worst case delay - this would be result in an overall latency which is too high to be acceptable for scheduled content. An experiment to apply commonly used FEC schemes to the packet traces finds that LDPC codes are the best FEC scheme for systems such as IPTV, though some lost packets cannot be recovered by any FEC schemes [27].

An important observation from this research is that using random packet loss is not realistic because it results in mostly isolated single packet losses, where as the packet traces show that packet loss is often bursty. One limitation of the research is the relatively small sample size (19 different residential hosts), though the service providers and access types were varied.

## 4.2 Directly Related Work

Song et al. [41] develop a tool named QScore to measure IPTV performance in real-time, using network performance to estimate video quality. The tool is designed to assist IPTV providers in spotting and fixing problems in the IPTV system, but as it is a real-time tool it cannot do full reference video quality evaluation. Klaue et al. [31] developed a framework named EvalVid which compares a sender packet trace and a receiver packet

trace to measure the network performance and evaluate the video quality via the PSNR (see section 4.4) algorithm. The framework is evaluated using packet traces gathered from sending video over a single wireless link. The system does not consider FEC, and is not used in conjunction with packet traces from a life-like IPTV set-up. Shihab et al. [39] investigate the number of IPTV connections which can be supported over various wired and wireless residential networks with varying buffer sizes by measuring packet loss. Mahimkar et al. [33] develop a sophisticated statistical analysis tool to identify problems in large IPTV systems using a range of data including device logs and customer complaints. Using WiFi to distribute IPTV data within a single household (or LAN) is common but can be problematic; it is found that a 2-hop network gives the best performance in terms of PSNR (see section 4.4) and network delay/jitter [32].

## 4.3 Forward Error Correction

Both [27] and [34] evaluate three FEC algorithms: 2D parity codes [21], Reed-Solomon erasure codes (RSE)[15] and LDPC-Staircase codes (LDPC)[14]. Matsuzono et al.[34] use simulated uniform packet loss ranging from 0 to 51%, where as Ellis et al.[27] use real packet traces which are found to exhibit bursty (rather than uniform) packet loss. Both find that LDPC gives the best performance, however it is suggested in [27] that the block size of 170 recommended in [34] is too large to allow packets to be repaired with a small enough delay for use in an Internet streaming application. Begen [23] compares Raptor codes [40] against 1D interleaved parity codes using packet loss simulations based on observed DSL noise models, finding that the best overall performance depends of the level of packet loss and loss burstiness.

To the best of my knowledge there do not exist any studies which measure the performance of FEC using resulting video quality, which is what the project will focus on.

## 4.4 Video Quality Assessment

There exist many different types of VQA metrics. There is not scope here to review VQA metrics in any deal, so a brief overview is given followed by a short review of some recent literature on VQA metrics.

VQA metrics can be objective or subjective, still-image or designed for video, traditional or perceptual, full-reference or no-reference. For an in depth classification see [24]. Objective metrics such as mean opinion score (MOS) involve a number of human testers watching videos and scoring the quality manually. While these are regarded as the most accurate, they are very expensive and time consuming. As such subjective techniques which automate the process are far more common. Subjective metrics are typically evaluated by how similar they are to objective metrics such as MOS. Full reference metrics rely on having both the original and copied videos so that a direct comparison can be performed where as no-reference metrics only have the copy of the video. Still-image metrics only compare two

individual images, so when used to evaluate video the approach is normally to compare each frame and calculate the mean value. One of the biggest disadvantages of this approach is that if frames get out of synchrony then a still-image metric will give a poorer score than an objective metric, because humans do not necessarily notice one video being one or two frames behind another, so long as there are no obvious distortions as a result. VQA metrics designed for video typically have a pre-processing stage where they line up the original and copied video. Perceptual based metrics such as [38] aim to simulate the human visual system (HVS) where as traditional metrics such as signal-to-noise ratio (SNR), peak-signal-to-noise ratio (PSNR) or mean squared error (MSE) treat the pictures as bitmaps and typically compare individual pixels.

A recent comparison of the performance of several common VQA metrics using the LIVE video database [29] is presented in [24]. The comparison finds that the best performing metrics are multiscale-structural similarity index (MS-SSIM), video quality metric (VQM) and motion-based video integrity evaluation (MOVIE). The Video Quality Expert Group (VQEG)[17] which is an organisation which assesses and standardises VQA metrics, found the VQuad-HD [18] metric to be the best metric for high definition (HD) video and standardised it as J.341 [9]. VQEG have a separate full reference standard for standard definition (SD) video - J.247 - which is based on the perceptual evaluation of video quality (PEVQ) algorithm [8]. For the purpose of this project, any of these VQA metrics will suffice. Factors such as cost and speed are important for the project; neither of the VQEG standards have freely available implementations and metrics such as MOVIE take a long time to execute. This project will start by using PSNR and MS-SSIM, this decision is explained in more detail in section 5.3.

## 4.5 Playout Buffering

Various approaches to implementing playout buffers exist. The choice of playout buffer can affect the quality of video in an IPTV system, so this section will review several such strategies.

Playout buffers can be static/fixed or dynamic. Static buffers simply store a fixed number of packets or video frames. The playback device takes a frame from the front of the buffer every time it needs one, and if a packet has not yet arrived when it is to be played then another frame will be substituted in, for example the preceding frame may be played again. More sophisticated techniques for substituting lost/late packets are possible, for example analysing the parts of the preceding frames which changed and predicting the subsequent frame. Thus any packet which experiences a queue delay longer than the playout buffer length (minus average queue delay) will be ignored; it will arrive too late to be played and will be substituted. Static playout buffers are very simple to implement and ensure a constant overall latency, but are inflexible and cannot react to changing network conditions.

Dynamic playout buffers can change the size of the buffer and can change the playback rate. They can be enlarged or shortened to store more or less packets and frames. Moreover, the playback strategy may be altered, and rather than always taking the next frame the

algorithm may choose to discard frames when the buffer is near full or play frames more than once if the buffer is near empty. Such systems are described in [47, 30]. When packets are taking longer than usual to arrive, and the buffer is therefore near empty, the algorithm slows down playback by playing some frames twice, for example making 10 frames last for 12 periods. When the slow packets finally arrive, and the buffer begins to fill, the algorithm speeds up playback by omitting some frames, for example using 10 frames in 8 periods. This system would outperform a static buffer if for example several subsequent packets were to arrive later than the size of the buffer. With a static buffer all the frames would be discarded and the preceding frame would be played many times causing visible distortion, which would last until a packet arrived with a small enough delay. With the dynamic buffer the video would first slow down a little, then speed up a little - which may or may not be noticeable. A similar system designed only for telephone audio is described in [35], however this shortens and elongates the silence present in voice conversations rather than changing the playback speed, and is therefore not directly applicable for a video playout buffer. The obvious disadvantage of dynamic playout buffers is that latency is not fixed, and without care may grow to be unacceptably high.

The overall latency which can be handled by IPTV systems is small. According to [39], IPTV systems can handle only around 100ms of latency. [47] recommends between 4 and 12 frames in a buffer with a target playout of 30 frames per second, with equates to around 130 - 400 ms. A more general recommendation for various types of audio and video playout buffer sizes is between 150 - 250ms [42]. According to [36] channel changes (which include but are not limited to the latency introduced by playout buffers) must be less than 500ms.

## 4.6 Video Codecs and Standards

Traditionally MPEG-2 is used for encoding video for television [45]. The H.264 standard (also known as MPEG-4 Part 10 or Advanced Video Coding (AVC) ) was introduced in 2003 with the target of doubling coding efficiency compared to previous video codecs. It was also designed with video streaming in mind; it is divided into a video coding layer (VCL) and network abstraction layer (NAL) which separates the concerns of encoding video and packaging it for transportation (or storage) [44]. H.264 has been observed to outperform MPEG-2 by a factor of 2.25 to 2.5 [45]. H.264 brings many technical improvements over older codecs such as MPEG-2, but the detail is outwith the scope of this document- see [44] for a thorough overview or the H.264 standard document [4] for the full detail.

RFC 6184 [16] outlines a mechanism for transmitting H.264 encoded data over RTP [11]. In short RFC 6184 allows 3 types of RTP packet payloads; single NALU, aggregated NALUs or fragmented NALUs. There exist different modes which disallow and allow NALUs from different frames to be aggregated. Aggregating NALUs from different frames is a potential mechanism for handling packet loss; because each frame's NALUs are spread across many packets the impact of a single lost packet on a single frame is reduced. Conversely, aggregating NALUs from different frames will increase overall latency compared to the other modes. When NALUs are fragmented, the loss of one RTP packet means that the entire

NALU cannot be decoded, therefore encoding data into smaller NALUs may be preferable to fragmenting large NALUs into many RTP packets.

# 5 Proposed Approach

The experiment for the project can be split into three stages:

1. Parse the existing network traces into an appropriate form
2. Simulate video being transmitted over the network
3. Compare the video which is received to that which was transmitted.

Stage 1 is relatively straight forward - it involves writing some scripts to parse some files. It is explained in a little more detail in section 5.6. Stage 2 is the most significant as it involves writing and testing the most code. The proposed approach for this work is outlined in section 5.7. Stage 3 involves utilising one of the VQA metrics described in section 4.4. Although there are many possible metrics which could be used, the experiment will primarily use those metrics which can be calculated without having to write out a full reference video, namely PSNR and MS-SSIM. This decision is explained in section 5.3. Because the video is evaluated as frames are generated by the simulator code, the VQA is performed by extending the simulator code, rather than having a separate process (which will be necessary if VQA metrics such as MOVIE are to be used). In order to speed up running the experiment some code will be written to allow the simulations to be run in parallel on many different computers simultaneously, this is described in section 5.8.

The remainder of this section is structured as follows. First some key decisions - the choice of video codec, libraries and VQA metric - are justified. Second the practical details of the experiment are discussed - which test cases will be run, what video will be used, how long the experiment should take. Finally the design of the code which will be written is presented.

## 5.1 Choice of Codec

H.264 will be the codec used for the core test cases. This is because it has a freely available open source implementation, it achieves a much better compression than codecs such as MPEG-2 (which is widely used in television), and it is widely used, especially for HD video. Other codecs may be experimented with, if time permits.

## 5.2 Choice of Video Libraries

The encoding and decoding of video will be achieved with the use of open source libraries. To encode video the libx264 library [20] will be used. To decode the video the libavcodec library (part of the FFmpeg library [2]) will be used. The FFmpeg command line tool will likely be used for various purposes, such as converting video files between YUV and H.264 in testing.

These open source libraries were chosen for a number of reasons. They are widely used, and therefore likely to be largely bug free. There are online communities providing examples and assistance. Moreover they are free, as in free speech and free beer; they cost nothing financially to use and the source code is available and can be changed for the purposes of the experiment if required.

There exist plenty of alternatives. There was not scope for an exhaustive comparison, though several options were considered. Neither the Microsoft implementation of H.264 [19] nor the Apple implementation [1] appear to offer the ability to control the size of encoded NALUs, which will be necessary to generate RTP packets of a fixed size (necessary because the packet traces used fixed packet sizes). In addition, these libraries are not platform independent, and in order to parallelise the experiment on a number of available Linux machines a Linux compatible library is incredibly useful. The GStreamer framework could have been used [3], but it is a very generic framework and so possibly would introduce extra complexity and hurt performance. Moreover the platform APIs would need to be studied, and the platform has dependencies which would make it less easy to install than an application written in C using libavcodec and libx264.

## 5.3 Choice of VQA Metric

The experiment will evaluate video quality using two video metrics: PSNR and MS-SSIM. As mentioned in section 5.5.2, other metrics may be used if time permits. This section will justify that choice.

PSNR is the standard used by much of the existing research, and therefore is a sensible choice as it eases comparison. However, PSNR is often criticised as a poor metric e.g., [43] because it ignores video distortions which humans notice most; other metrics have been shown to given better measures of video quality. Amongst the best performing metrics in [24] are J.341, MOVIE and MS-SSIM. J.341 cannot be used because there is not a freely available implementation, only an expensive proprietary implementation which only runs on Windows. MOVIE could be used, but the software is slow - taking several hours to compare two 500 frame videos. Moreover, it is not obvious if MOVIE would handle loading files which are bigger than the RAM of a machine. While the project could involve fixing this, it does not seem like a worthy way to spend the limited time. A second problem with MOVIE, and any other metric which considers temporal distortion is that the existing software implementations require two full video sequences (i.e., YUV files). Therefore using these VQA metrics would require either re-implementing the algorithms or writing the full received video sequence to disk for every network trace. Given that there are over 3000 network traces and each received video file would be tens of GBs, using these metrics would require significant hardware requirements (disk storage on each worker node doubles) and would cost time. It also adds complexity to the experiment - there would need to be one process to produce the received file and another process to compare the two files.

These problems can be avoided with static image metrics such as PSNR and MS-SSIM, because the received video frames only need to be held in memory until they have been

compared with the corresponding original frame, at which point they can be discarded. Moreover, while temporal distortion is a real problem with streamed video, the network traces being used in this experiment showed very low delay, and so temporal distortion is likely to be very limited. The playout buffer which is implemented can be chosen such that temporal distortion is minimised or possibly totally eradicated (if delay is low enough). Despite the fact that MS-SSIM is a static image quality metric which ignores temporal distortion, it performs nearly as well as MOVIE and J.341 in [24]. There are freely available implementations of the MS-SSIM and PSNR metrics in C which are designed to run quickly [5]. Thus for the purposes of this research MS-SSIM makes a sensible second choice for VQA after PSNR.

## 5.4 Sample Video

Sample videos of varying resolutions and frame rates are needed for the experiment. The required bit rates are determined by the packet traces, and are 1, 2, 4, 5, 6 and 8.5Mb/s. The formula provided in [22] suggests that the bit rate of encoded video can be estimated as:

```
bitrate = widthpixels x heightpixels x framepersecond x motionrank x 0.07
```

Where motion rank ranges from 1 to 4, with 1 being a video with little movement and 4 means lots of movement (e.g., an action film trailer). Using this formula, the bit rate for two standard HD resolutions are shown.

| | Resolution | Frame Rate | Motion Rank | Bit Rate (Mb/s) |
|---|---|---|---|---|
| A | 1280 x 720 | 25 | 1 | 1.5 |
| B | 1280 x 720 | 25 | 2 | 3 |
| C | 1280 x 720 | 25 | 4 | 6 |
| D | 1280 x 720 | 50 | 4 | 12 |
| E | 1920 x 1080 | 25 | 1 | 3.5 |
| F | 1920 x 1080 | 25 | 2 | 7 |
| G | 1920 x 1080 | 25 | 4 | 14 |

Table 1: H.264 bit rates

From this the required videos for the experiment are shown below in table 2 (assuming each pixel is stored as 3 bytes, which is standard for YUV 4:2:0). At first it may seem strange that a larger video is required for a 10 minute 4Mb/s trace than a 10 minute 6Mb/s trace, but this is because the former uses a video with a much lower motion rank and therefore the encoding of the video can achieve much higher compression of the data.

| Bit Rate (Mb/s) | Video Type | Longest Trace (minutes) | Video Size (GB) |
|---|---|---|---|
| 1 | A | 4 | 15.5 |
| 2 | A/B | 10 | 39 |
| 4 | E | 10 | 88 |
| 5 | C | 4 | 15.5 |
| 6 | C | 10 | 39 |
| 8.5 | F/G | 5 | 44 |

Table 2: Required video for experiment

## 5.5 Experiment Test Cases

This section will document that test cases which will be run as part of the experiment.

### 5.5.1 Core Test Cases

The core test cases for the project are based on the test cases in [27]. Copying the test cases allows for a comparison between the findings of that research and this project, and maximises the chance that some meaningful conclusions can be drawn from the experiment.

All of the network traces from [25, 26] will be used. In addition some fabricated traces will be used in order to allowed comparison. The fabricated traces will have: no loss, 1 / 5 / 10 / 15 / 20 / 25% random loss. The fabricated traces will have uniform latency; packets which take too long have the same effect as a lost packet and so varying packet latency would be the same as increasing packet loss in a less obvious way. The random packet traces will be generated only once, rather being generated randomly for every test case.

Each network trace will be run with several different FEC techniques (similar to [27] except there is no point in using LDPC with k = 500 or k = 1000 because the repair delay would be prohibitive). The FEC schemes used (to start) will no: No FEC, 2D k = 16, RSE k = 170, LDPC k= 170 and LDPC, k = (delay_bound / packet_spacing) -1.The last technique follows the recommendation from [27] that the block size be set such that the worst-case FEC delay does not exceed the delay bound.

For the core test cases, the following parts of the experiment will be constant: jitter buffer (simple static), RFC 6184 packetisation (non-interleaved), video codec used (H.264) and H.264 encoding parameters.

### 5.5.2 Additional Test Cases

It is anticipated that there will be time left after the core test cases have been run. If so further tests will be run, in the following order:

1. Try different types of video e.g., talking heads versus sports versus movie trailers.
2. Vary the H.264 encoding parameters: rather than use FEC add redundant frames into the stream, and/or alter the number of I-frames etc.
3. Vary the RFC 6184 packetisation mode - try interleaved mode
4. Vary the jitter buffer strategy - try a simple adaptive buffer, change the size of the adaptive buffer
5. Use more VQA metrics, such as MOVIE and/or VQM which consider temporal distortion
6. Use FEC on delayed packets as well as lost packets
7. Try a different codec e.g., MPEG-2

Varying the video type used is important because different video types will encode in different ways and therefore be affected by lost packets in different ways. It is common to evaluate image or video algorithms with many different images or videos, and doing so will lend credence to the results of the experiment. Varying the H.264 encoding parameters and/or varying the RFC 6184 packetisation mode are interesting because they are alternatives (or supplements) to using FEC, and so will allow a comparison between the different strategies for dealing with packet loss. The playout buffer is less interesting because the size can be set based on the a priori knowledge of the packet traces, but it may be interesting to compare static and dynamic buffers. Using other VQA metrics would probably give a slightly more accurate measurement of the video quality, and so might be important if the results are not clear. Using FEC on delayed packets would be interesting however it involves extra work (making more changes to openFEC) which may put it beyond the scope of the project. Finally, trying other codecs would likely yield different results and would be a more thorough evaluation of FEC techniques, however it would involve adding more decoders and encoders to the simulation code, which again may not be possible in the limited time available.

### 5.5.3 Anticipated Time Requirements

There will be 7 fabricated network traces, as well as around 3600 real network traces. Each of these will be simulated with 5 different FEC techniques. Therefore the core test cases will require around 18,000 simulations. Based on the assumption that the average trace is 4 minutes long[5] and the estimation that the simulator/quality assessment code will take around about the same amount of time as the network trace to run[6], it will take 50 days to run the core test cases on a single machine. There are around 100 potential worker nodes, meaning that the core test cases could in theory be completed in a single day. Running

[5]The trace lengths are 1, 4, 5 and 10 minutes

[6]Based on some prototype code which was written, executed and timed on some lab computers in the department

all of the tests in one day is unrealistic, however it should be possible to run one or more simulations in several weeks.

## 5.6 Parsing Trace Files

The parsing of trace files will be done via a small amount of scripting, probably with Python as that is what I am most comfortable with.

### 5.6.1 No FEC

The pre-existing binary output files which determine whether a packet was received can be combined with the pre-existing log files listing the arrival times of each packet.

### 5.6.2 FEC

In the case where openFEC has been run over the network traces, there exist trace files which indicate, for each packet, whether it was received, whether it was lost but recovered or whether it was lost and not recovered. If the packet was recovered then the delay incurred is given, as the number of subsequent packets which must be received before recovery. This number-of-packets delay can be translated into a time delay, by looking at the arrival time of the repair packet.

This means that FEC repairs are only ever attempted if a packet is lost. In a video streaming situation, where a suitably delayed packet may the have same effect as a lost packet, FEC could be used to recover a packet which was delayed (if the repair packet was not delayed). Given that [26] found very little re-ordering of packets, in this experiment it is very unlikely that this would make a difference.

## 5.7 Simulating Video Transmission

The process of simulating video transmission, including evaluating the decoded video can be broken down as shown in figure 3. Each numbered component is explained in more detail below.

### 5.7.1 Video Reader

This component simply reads an input YUV video file one frame at time, ensuring there are enough frames in the input buffer such that there is always encoded video available for the network simulator. A copy of each frame is placed in a separate buffer for the

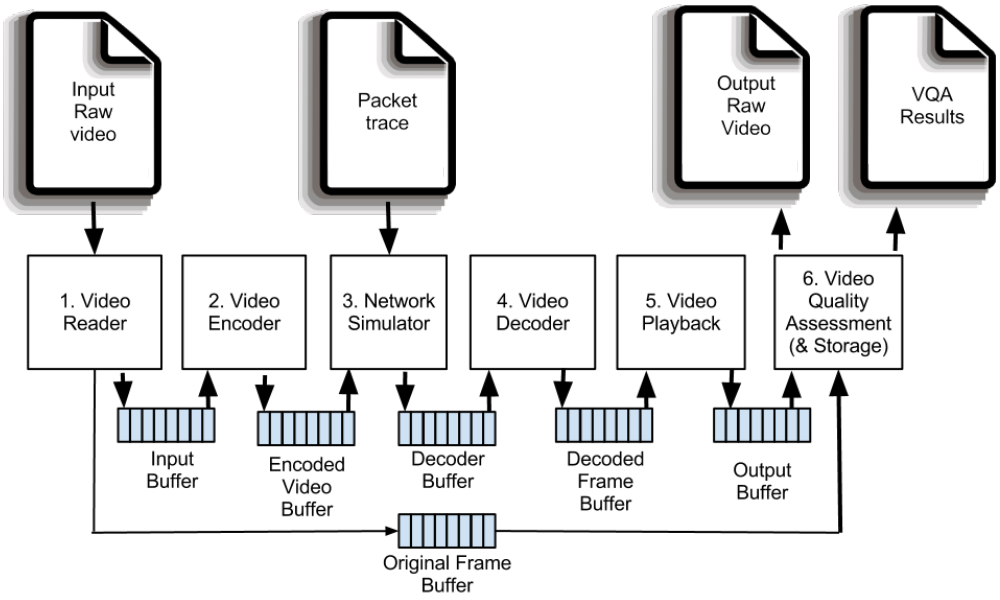


Figure 3: Process of simulating video transmission

VQA component. In theory one buffer could be shared by the video encoder and VQA component, but this introduces needless complexity and potentially a small time penalty when components block each others access to the buffer. The added memory requirement for an extra buffer is reasonably small.

### 5.7.2 Video Encoder

The video encoder encodes each single YUV frame into a number of H.264 NALUs using the open source libx264 library. The x264 library takes (optionally) as a parameter a maximum bitrate to generate, and this will be set to the bitrate associated with the packet trace. The NALUs will then be packetised as per RFC 6184 [16], as if they were to be transmitted as the payload of RTP packets. There is no need to actually create RTP packets, because the data will not really be sent over a network. Because the packet traces use fixed-size fixed-rate packets, the NALUs will be packetised into equal size packets as determined by the packet trace. For example, a packet trace with a bitrate of 1Mb/s will be split into 100 packets per second each of size 1316 bytes. RFC6184 is flexible in how NALUs are combined to form packets, so while generating packets of identical size will almost certainly not be possible, the packet sizes should be very similar.

Note that when FEC is being used the maximum bitrate given to the encoder will be reduced, to allow for the FEC packets. The packet size will not be changed. This means the decoder will have to achieve a slightly higher level of compression when FEC is to be used. To illustrate, for the 10Mb/s trace, the encoder will be asked to produce encoded content which does not exceed 10Mb/s when no FEC is used. When some FEC technique with an overhead of 25% is used (meaning that 80% of the traffic is data and 20% is FEC),

the encoder will be asked to produce a maximum of 8Mb/s.

### 5.7.3 Network Simulator

The network simulator will read the encoded RFC 6184 H.264 packets in FIFO order, and it will read the packet trace information chronologically. The simplified psuedo-code for the network simulator is as follows:

```
next_packet = encodedVideoBuffer.read()
next_packet_info = packet_trace_file.next_packet_info()
for (time = 0 ;time < END_TIME_MS; time++)
  //assume packets delivered in order
  if (time == next_packet_info.arrival_time)
    if (next_packet_info.was_delivered)
      decoderBuffer.write(next_packet)
      next_packet = encodedVideoBuffer.read()
      next_packet_info = packet_trace_file.next_packet_info()
```

The code will need to have a strategy for out of order packets, though in the packet traces this is rare. A simple approach would be to wait one or two times the average period between packets, then count the packet as lost and discard it when it arrives. If the video decoder can handle out of sequence RFC6184 packets, then a more sophisticated strategy may be possible.

### 5.7.4 Video Decoder

The video decoder continuously reads packets from the decoder buffer, splits them back out into NALUs and decodes them. It will use an external library (libavcodec) to do this. Note that some information will have been dropped by the network simulator, and the decoder will have to cope with this - libavcodec handles this without the calling code need not do anything - e.g., video decoder can be totally oblivious to lost NALUs. Whenever the decoder has decoded a full frame it writes it immediately to the decoded frame buffer.

### 5.7.5 Video Playback

The playback component simulates the role of the display device in a real IPTV set-up, for example the television or computer screen. It has a frame rate (the same as the frame rate of the input video), and renders a (new) frame every period, where the period is (1000 / frames per second) ms. For example at a frame rate of 50, a frame is displayed every 20ms. In the experiment, rather than displaying the frame, it will be written to the output buffer.

This component is responsible for implementing the playout buffer, which is common in video streaming applications. The experiment will start by using a simple static playout buffer, and if time permits, this may be compared to a simple adaptive buffer as described in [30]). Analysis of the packet traces, shown in table 3, shows that a buffer size of 50, 100, or 200ms would mean only a tiny fraction of packets would be lost.

| Overall Packet Delay (ms) | | Number of packets | Percentage of packets |
|---|---|---|---|
| Greater than | Less than | | |
| 0 | 1 | 42,902,633 | 18.49310 |
| 1 | 5 | 143,648,744 | 61.91953 |
| 5 | 10 | 29,796,262 | 12.84362 |
| 10 | 20 | 14,400,314 | 6.20723 |
| 20 | 50 | 1,242,304 | 0.53549 |
| 50 | 100 | 2,032 | 0.00088 |
| 100 | 200 | 296 | 0.00013 |
| 200 | 500 | 40 | 0.00002 |
| 500 | infinity | 13 | 0.00001 |

Table 3: Packet delay

### 5.7.6 Video Quality Assessment and Storage

The final component will primarily be used to measure the quality difference between sent and received YUV frames. If required (for example if a temporally-aware VQA metric is to be used) it will write out the received video to file.

The video quality assessment component will compare each received frame in the output buffer to the corresponding frame in the original frame buffer, for example using PSNR. There exist open source implementations of both PSNR and MS-SSIM in C which take two YUV frames and return a score e.g., [5]. The score will either be written to file for each frame, or held in memory to be averaged.

### 5.7.7 Implementation

The code for the network simulation will be written in C, given that it will be using the libx264 and libavcodec C libraries. Originally it was planned that each component would be a single thread, in order to allow multi-core processors to be utilised effectively. However, running prototype code has shown that the video encoding (thread) dominates the CPU usage to the extent that more than one instance of the code needs to be executed simultaneously to fully utilise multiple cores. This means that a single threaded approach is better, as it is simpler.

The code will be modular in that each component will be implemented as a pair of .c and .h files. This ensures separation of concerns, and should improve the maintainability and extendibility of the code. The buffers which components write frames to will be simple doubly linked lists, meaning that components can efficiently append (to the end) and remove (from the head) frames to/from the buffers.

The code will be designed to run within the amount of RAM available (4GB). It will not attempt to read the entire input video file (which exceeds the amount of RAM) at the start; it will read frames a few a time a time such that there are always enough frames available for the network simulator component. Assuming that each buffer holds a maximum of 100 frames, and the largest un-encoded frame is 3MB (single raw 1920 x 1080 YUV 4:2:0 frame), the largest encoded H.264 frame is 0.5MB then the total RAM requirements should be under 1GB.

## 5.8 Distributing the Simulations

Simulation will be run on many machines, to cut down the elapsed time. The simulator code will be stored on the school file server (in my home directory). This will allow a number of unused Linux lab computers - *the worker nodes* - on the school network to be used to run the simulations (as these nodes have my home directory on the school file server mounted locally). The raw video files for the video are too big to be stored on the school file server (see table 2 in section 5.4) so they will be stored on a dedicated Linux box which I have requested from support - *the server node*. The network traces may or may not fit in my home directory, if not they will be stored with the video files on the server node.

Some control code will be written (probably a Python script) to manage many worker nodes running the simulator code simultaneously. This control code will be stored in my home directory along side the simulator code, and so can be invoked on the worker nodes via SSH. Along with this control code will be some configuration which specifies which worker nodes should simulate which network traces against which sample videos.

The control code will copy the sample video to be used from the server node to the local disk of the worker node. This may take some time given the size of the files. One potential solution would be to encode the video on the server node, and write the encoded contents to a file which can easily to read by the network simulator - the encoded file would be much smaller than the raw video file. The problem with this however is that the reference YUV frame would not be available for VQA assessment.

Where the video file is too big for the local disk then the simulation will be run on the server node. This should only happen for the 10 minute trace at a bit rate of 4 Mb/s, as all other traces can run with video files which are less than 40GB - the space available on the worker nodes' local disk. The configuration will be such that downloading video files is minimised - each worker node will simulate network traces which use the same video.

The control code will then copy the network traces, if they are not already on the local

disk or on the school file server (mounted on the worker nodes). Finally the control code will invoke the simulation code, which will run the simulation and write the VQA scores to a file on the school file server.

# 6 Work Plan

The project will last 12 weeks. The project can be split into several distinct tasks, as detailed below. For each task the amount of time required has been estimated. I believe that a generous amount of contingency has been built into each task. The total amount of time adds up to 12 weeks, but that is not to say that the tasks will be performed consecutively - several tasks may interleaved as required. The Gantt chart below shows the rough planned order of the tasks.

1. Parse network traces into appropriate format and find/capture sample video - *one week*
2. Write network simulator code, test that it works, write control scripts and test that the simulations can be successfully distributed - *three weeks*
3. Run the core test cases, analyse them and write up findings - *two weeks*
4. Run further test cases, analyse them and write up findings - *two weeks*
5. Write up dissertation - *four weeks*

## 6.1 Gantt Chart

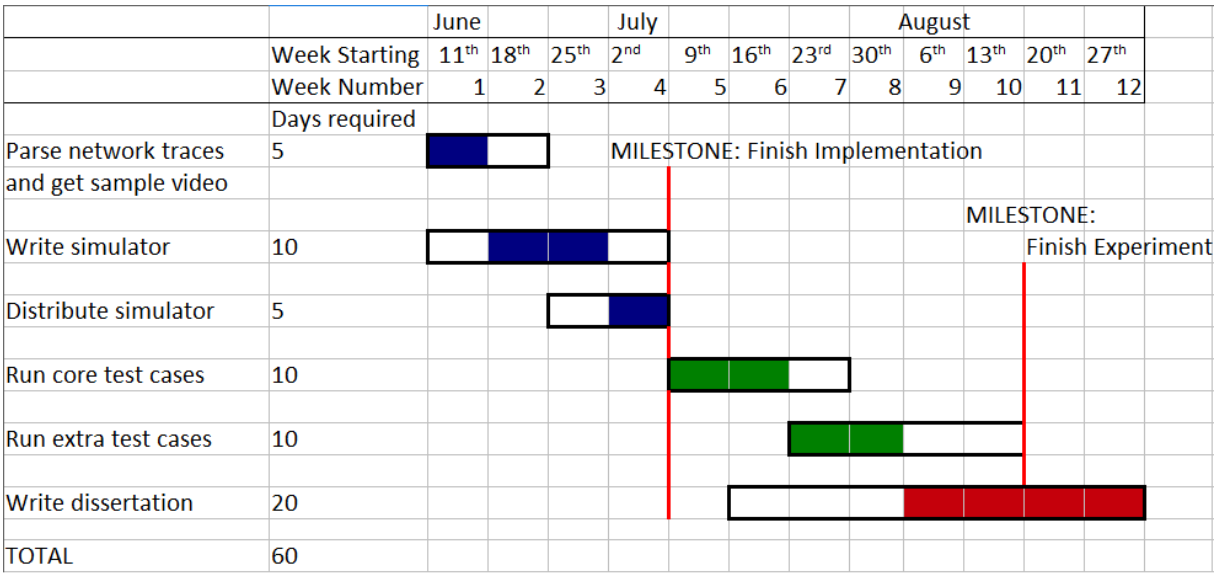


Figure 4: Gantt chart

## 6.2 Risk Assessment

There are a number of risks which could prevent the successful completion of the project. This section documents some of the major risks, and how they have been mitigated.

*Martin Ellis - the primary author of the previous research, upon which this project builds, is unavailable for clarification if required.*

The author has already discussed the previous research with Martin Ellis and clarified details, as well as arranging to get a copy of the files used. As such it is not expected that any further clarification will be necessary. However, Martin Ellis will be available until the end of June, though he will likely be very busy writing a PhD thesis. One of the secondary authors of the research is supervising the project, so will be available for questions.

*The simulator code is hard to get working, given that it is written in C and uses open source libraries which may not be well documented and may contain bugs.*

Some prototype code has already been written which makes use of both the libavcodec and libx264 libraries. This prototype code reads a YUV video file, encodes it using the H.264 codec, decodes it again into YUV format and writes the resulting video to a file. While it is not polished code, it provides a useful starting point and has verified that the open source libraries are not too difficult to use or obviously buggy. Both open source libraries are widely used, which suggests crude bugs are unlikely. The author is reasonably experienced with the use of C in a Linux environment, and writing the prototype code proved a little easier than expected (though still a little fiddly).

*The open source libraries do not provide the required function*

The prototype code which has been written proves that the open source libraries provide at least the base functionality required for the core test cases, for example allowing the size of the encoded H.264 NALUs to be specified.

*The simulation takes too long to run*

The estimation for the time required (in section 5.5.3) is based on the time taken for the prototype code to run on a mid-range dual-core processor, and is therefore likely to be reasonably accurate. Moreover, there are around 100 unused machines which can be utilised for the experiment, meaning that the elapsed time of the simulation can be reduced by a factor of around 100. The VQA metrics to be used have been chosen for their low execution times. Finally the test cases have been split into core and additional to ensure that the most important test cases are executed first.

*The code which is written, and/or the results which are generated, is lost due to a hard disk crash or similar.*

Both the code written and the results generated will be backed up regularly to a geographically different location from the primary copy (the school file server). This may be

achieved, for example, via use of a repository service such as SourceForge or Git Hub, or a cloud storage provider such as DropBox.

*Suitable videos cannot be found*

There exist video databases available to the general public and the research community. The experiment will need to make use of a 10 minute long video, which may exceed that stored in these databases. Possible solutions include concatenating videos or capturing video from another source e.g., video from a DVD.

# References


[1] Apple H.264 implementation. http://www.macupdate.com/app/mac/20273/x264-quicktime-codec. [Online; accessed 19-March-2012].

[2] FFmpeg video library. http://ffmpeg.org/documentation.html. [Online; accessed 19-March-2012].

[3] Gstreamer multimedia framework. http://gstreamer.freedesktop.org/. [Online; accessed 19-March-2012].

[4] H.264 : Advanced video coding for generic audiovisual services. http://www.itu.int/rec/T-REC-H.264. [Online; accessed 19-March-2012].

[5] Image quality assessment C library. http://tdistler.com/iqa/index.html. [Online; accessed 19-March-2012].

[6] Information technology – coding of audio-visual objects – part 14: MP4 file format. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38538. [Online; accessed 22-March-2012].

[7] Information technology  generic coding of moving pictures and associated audio information: Systems. http://www.itu.int/rec/T-REC-H.222.0. [Online; accessed 22-March-2012].

[8] J.247 : 'objective perceptual multimedia video quality measurement in the presence of a full reference'. http://www.itu.int/rec/T-REC-J.247/en. [Online; accessed 19-March-2012].

[9] J.341 : 'objective perceptual multimedia video quality measurement of HDTV for digital cable television in the presence of a full reference'. http://www.itu.int/rec/T-REC-J.341-201101-I. [Online; accessed 19-March-2012].

[10] RFC 2474, 'definition of the differentiated services field (DS field)'. http://www.ietf.org/rfc/rfc2474.txt. [Online; accessed 19-March-2012].

[11] RFC 3550, 'RTP: A transport protocol for real-time applications'. http://www.ietf.org/rfc/rfc3550.txt. [Online; accessed 19-March-2012].

[12] RFC 3711, 'the secure real-time transport protocol (SRTP)'. http://www.ietf.org/rfc/rfc3711.txt. [Online; accessed 19-March-2012].

[13] RFC 3984, 'RTP payload format for H.264 video'. http://tools.ietf.org/html/rfc3984. [Online; accessed 19-March-2012].

[14] RFC 5170, 'low density parity check (LDPC) staircase and triangle forward error correction (FEC) schemes'. http://tools.ietf.org/html/rfc5170. [Online; accessed 19-March-2012].

[15] RFC 5510, 'reed-solomon forward error correction (FEC) schemes'. http://tools.ietf.org/html/rfc5510. [Online; accessed 19-March-2012].

[16] RFC 6184, 'RTP payload format for H.264 video'. http://tools.ietf.org/html/rfc6184. [Online; accessed 19-March-2012].

[17] Video quality experts group (VQEG). http://www.its.bldrdoc.gov/vqeg/vqeg-home.aspx. [Online; accessed 19-March-2012].

[18] VQUAD-HD video quality assessment metric. http://www.vquad-hd.info/. [Online; accessed 19-March-2012].

[19] Windows H.264 implementation. http://msdn.microsoft.com/en-us/library/windows/desktop/dd797816(v=vs.85).aspx. [Online; accessed 19-March-2012].

[20] x264 open source H.264 C library. http://www.videolan.org/developers/x264.html. [Online; accessed 19-March-2012].

[21] SMPTE 2022-1, forward error correction for real-time video/audio transport over IP networks, 2007.

[22] K Amerasinghe. H.264 for the rest of us. Technical report, Adobe Systems Incorporated, November 2009.

[23] A.C. Begen. Error control for IPTV over xDSL networks. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 632 –637, jan. 2008.

[24] S. Chikkerur, V. Sundaram, M. Reisslein, and L.J. Karam. Objective video quality assessment methods: A classification, review, and performance comparison. *Broadcasting, IEEE Transactions on*, 57(2):165 –182, june 2011.

[25] M. Ellis and C. Perkins. Packet loss characteristics of IPTV-like traffic on residential links. In *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pages 1 –5, jan. 2010.

[26] M. Ellis, C. Perkins, and D. Pezaros. End-to-end and network-internal measurements of real-time traffic to residential users. In *ACM Multimedia Systems 2011 (MMSys'11)*, February 2011.

[27] M. Ellis, D. P. Pezaros, and C. Perkins. Performance analysis of AL-FEC for RTP-based streadming video traffic to residential users. In *Proceedings of the 19th International Packet Video Workshop, to appear*, May 2012.

[28] M. Jeffrey, S. Park, K. Lee, G. Adams, and S. Savage. Content security for IPTV. *Communications Magazine, IEEE*, 46(11):138 –146, november 2008.

[29] A. C. Bovik K. Seshadrinathan, R. Soundararajan and L. K. Cormack. Study of subjective and objective quality assessment of video. *IEEE Transactions on Image Processing*, 19:1427–1441, June 2010.

[30] M. Kalman, E. Steinbach, and B. Girod. Adaptive media playout for low-delay video streaming over error-prone channels. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(6):841 – 851, june 2004.

[31] Jirka Klaue, Berthold Rathke, and Adam Wolisz. EvalVid - a framework for video transmission and quality evaluation. In *In Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 255–272, 2003.

[32] Deer Li and Jianping Pan. Performance evaluation of video streaming over multi-hop wireless local area networks. *Wireless Communications, IEEE Transactions on*, 9(1):338 –347, january 2010.

[33] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. Towards automated performance diagnosis in a large IPTV network. *SIGCOMM Comput. Commun. Rev.*, 39(4):231–242, August 2009.

[34] K. Matsuzono, J. Detchart, M. Cunche, V. Roca, and H. Asaeda. Performance analysis of a high-performance real-time application with several al-fec schemes. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 1 –7, oct. 2010.

[35] Sue B. Moon, Jim Kurose, and Don Towsley. Packet audio playout delay adjustment: performance bounds and algorithms. *ACM/Springer Multimedia Systems*, 6:17–28, 1998.

[36] K. Ahmed R. Kooij and K. Brunnstrom. Perceived quality of channel zapping. In *Proc. 5th IASTED Intl. Conf. Commun. Sys. and Networks*, August 2006.

[37] M.N.O. Sadiku and S.R. Nelatury. IPTV: An alternative to traditional cable and satellite television. *Potentials, IEEE*, 30(4):44 –46, july-aug. 2011.

[38] K. Seshadrinathan and A.C. Bovik. Motion tuned spatio-temporal quality assessment of natural videos. *Image Processing, IEEE Transactions on*, 19(2):335 –350, feb. 2010.

[39] E. Shihab, Fengdan Wan, Lin Cai, A. Gulliver, and N. Tin. Performance analysis of IPTV traffic in home networks. In *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pages 5341 –5345, nov. 2007.

[40] A. Shokrollahi. Raptor codes. *Information Theory, IEEE Transactions on*, 52(6):2551 –2567, june 2006.

[41] Han Hee Song, Zihui Ge, Ajay Mahimkar, Jia Wang, Jennifer Yates, Yin Zhang, Andrea Basso, and Min Chen. Q-score: proactive service quality assessment in a large IPTV system. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 195–208, New York, NY, USA, 2011. ACM.

[42] C.J. Sreenan, Jyh-Cheng Chen, P. Agrawal, and B. Narendran. Delay reduction techniques for playout buffering. *Multimedia, IEEE Transactions on*, 2(2):88 –100, jun 2000.

[43] Zhou Wang, Ligang Lu, and A.C. Bovik. Video quality assessment using structural distortion measurement. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages III–65 – III–68 vol.3, 2002.

[44] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):560 –576, july 2003.

[45] Yun Wu, Yong Zhao, and Jianshi Li. Brief analysis of the H.264 coding standard. In *Intelligent Information Hiding and Multimedia Signal Processing, 2007. IIHMSP 2007. Third International Conference on*, volume 2, pages 154 –157, nov. 2007.

[46] Yang Xiao, Xiaojiang Du, Jingyuan Zhang, Fei Hu, and S. Guizani. Internet protocol television (IPTV): The killer application for the next-generation internet. *Communications Magazine, IEEE*, 45(11):126 –134, november 2007.

[47] M.C. Yuang, S.T. Liang, Yu.G. Chen, and C.L. Shen. Dynamic video playout smoothing method for multimedia applications. In *Communications, 1996. ICC 96, Conference Record, Converging Technologies for Tomorrow's Applications. 1996 IEEE International Conference on*, volume 3, pages 1365 –1369 vol.3, jun 1996.