REAL-TIME VIDEO STREAMING USING PEER-TO-PEER FOR VIDEO DISTRIBUTION

J HINDS

2008

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

REAL-TIME VIDEO STREAMING USING PEER-TO-PEER FOR VIDEO DISTRIBUTION

By

Jeffrey Alec Stanley Hinds

Studyleader: Professor J.C. Olivier

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Computer)

in the

Department of Electrical, Electronic & Computer Engineering

in the

School of Engineering

in the

Faculty of Engineering, Built Environment & Information Technology

UNIVERSITY OF PRETORIA

July 2008

SUMMARY

REAL-TIME VIDEO STREAMING USING PEER-TO-PEER FOR VIDEO DISTRIBUTION

by

Jeffrey Alec Stanley Hinds

Studyleader: Professor J.C. Olivier

Department of Electrical, Electronic & Computer Engineering

Master of Engineering (Computer)

The growth of the Internet has led to research and development of several new and useful applications including video streaming. Commercial experiments are underway to determine the feasibility of multimedia broadcasting using packet based data networks alongside traditional *over-the-air* broadcasting. Broadcasting companies are offering low cost or free versions of video content online to both guage and at the same time generate popularity. In addition to television broadcasting, video streaming is used in a number of application areas including video conferencing, telecommuting and long distance education.

Large scale video streaming has not become as widespread or widely deployed as could be expected. The reason for this is the high bandwidth requirement (and thus high cost) associated with video data. Provision of a constant stream of video data on a medium to large scale typically consumes a significant amount of bandwidth. An effect of this is that encoding bit rates are lowered and consequently video quality is degraded resulting in even slower uptake rates for video streaming services.

The aim of this dissertation is to investigate peer-to-peer streaming as a potential solution to this bandwidth problem. The proposed peer-to-peer based solution relies on end user co-operation for video data distribution. This approach is highly effective in reducing the outgoing bandwidth requirement for the video streaming server. End users redistribute received video chunks amongst their respective peers and in so doing increase the potential capacity of the entire network for supporting more clients. A secondary effect of such a

system is that encoding capabilities (including higher encoding bit rates or encoding of additional sub-channels) can be enhanced. Peer-to-peer distribution enables any regular user to stream video to large streaming networks with many viewers.

This research includes a detailed overview of the fields of video streaming and peer-to-peer networking. Techniques for optimal video preparation and data distribution were investigated. A variety of academic and commercial peer-to-peer based multimedia broadcasting systems were analysed as a means to further define and place the proposed implementation in context with respect to other peercasting implementations.

A proof-of-concept of the proposed implementation was developed, mathematically analyzed and simulated in a typical deployment scenario. Analysis was carried out to predict simulation performance and as a form of design evaluation and verification. The analysis highlighted some critical areas which resulted in adaptations to the initial design as well as conditions under which performance can be guaranteed. A simulation of the proof-of-concept system was used to determine the extent of bandwidth savings for the video server.

The aim of the simulations was to show that it is possible to encode and deliver video data in real time over a peer-to-peer network. The proposed system achieved expectations and showed significant bandwidth savings for a substantially large video streaming audience. The implementation was able to encode video in real time and continually stream video packets on time to connected peers while continually supporting network growth by connecting additional peers (or stream viewers). The system performed well and showed good performance under typical *real world* restrictions on available bandwidth capacity.

Keywords:
Peer-to-peer, video, streaming, bandwidth, encoding, IPTV, live, multimedia, peercasting, multicast.

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

OPSOMMING

REAL-TIME VIDEO STREAMING USING PEER-TO-PEER FOR VIDEO DISTRIBUTION

deur

Jeffrey Alec Stanley Hinds

Studieleier: Professor J.C. Olivier

Departement Elektriese-,Elektroniese- & Rekenaar Ingenieurswese

Meester in Ingenieurswese (Rekenaar)

Die onlangse toename in die gebruik van die Internet het gelei tot navorsing in, en die ontwikkeling van, verskeie nuwe en bruikbare toepassings soos videostroming. Tans word kommersiële proeflopies uitgevoer om die lewensvatbaarheid van multimedia uitsendings deur middel van pakket-gebaseerde datanetwerkte, tesame met tradisionele luguitsendings, te bepaal. Uitsaaimaatskappye lewer videomateriaal aanlyn teen lae of selfs geen koste om sodoende die populariteit van sulke dienste te bepaal en terselfdertyd nuwe markte te ontwikkel. Videostroming word saam met televisie-uitsendings gebruik in ’n aantal toepassings wat videokonferensie, telewerk en afstandsonderrig insluit.

Grootskaalse videostroming is nog nie wydverspreid geïmplementeer soos aanvanklik vermoed was nie. Die rede hiervoor is die hoë bandwydte vereistes (gekoppel met hoë kostes) wat gepaardgaan met video data. Die daarstel van ’n konstante videodatastroom op ’n medium tot groot skaal, gebruik ’n noemenswaardige hoeveelheid bandwydte. ’n Newe-effek hiervan is dat enkoderingbistempos verlaag word en videokwaliteit ook afneem, wat kan lei tot verlaagde opneemtempos van videostromingsdienste.

Die doel van dié verhandeling is om eweknie-netwerk of *peer-to-peer* stroming te ondersoek as ’n potensiële oplossing vir die bandwydteprobleem. Die voorgestelde eweknie-netwerk oplossing is afhanklik van eindgebruiker samewerking vir videodataverspreiding. Hierdie benadering is hoogs effektief in die verlaging van die uitgaande bandwydte vereistes van die videostromingsbediener. Eindgebruikers herversprei videobrokkies tussen hulle afsonderlike

ewekniëe en sodoende verhoog hulle die totale kapasiteit van die hele netwerk om meer kliënte te kan ondersteun. 'n Sekondêre effek van so 'n stelsel is die enkoderingsvermoë (insluitende hoër enkoderingbistempos of die enkodering van subkanale) wat verbeter kan word. Eweknie-netwerkverspreiding maak dit moontlik vir gewone gebruikers om videos te stroom na groot netwerke met baie meer gebruikers.

Hierdie navorsing gee 'n omvattende oorsig oor die areas van videostroming en eweknie-netwerke. Tegnieke vir optimale videovoorbereiding en dataverspreiding word ondersoek. 'n Verskeidenheid akademiese en kommersiële eweknie-netwerkgebaseerde multimedia uitsaaistelsels is geanaliseer om die konteks te definieer waarbinne die voorgestelde eweknie-uitsendings geïmplementeer kan word.

'n Konseptuele ontwerp van die voorgestelde implementering is ontwikkel, wiskundig geanaliseer en gesimuleer vir 'n tipiese uitrolscenario. Analises is uitgevoer om te bepaal wat die gesimuleerde werkverrigting sal wees, sowel as om die ontwerp te evalueer en te verifieer. Die analises het onder andere die volgende uitgewys: kritieke areas wat daartoe gelei het dat die oorspronklike ontwerp aangepas moes word sowel as die kondisies waaronder werksverigting gewaarborg kon word. 'n Simulasie van die konseptuele ontwerpstelsel is gebruik om te bepaal wat die bandwydtebesparing vir die videobediener sou wees.

Die doel van die simulasies was om te bewys dat dit moontlik is om videodata te enkodeer en intyds (*real-time*) te versprei oor 'n eweknie-netwerk. Die voorgestelde stelsel het aan verwagtings voldoen en 'n noemenswaardige bandwydtebesparing getoon vir 'n betekenisvolle groot videostromingsgehoor. Die geïmplementeerde ontwerp was ook in staat om video intyds te enkodeer en deurgaans videodatapakkies intyds te stroom na verbinde eweknieë. Terselfdetyd het dit ook netwerkgroei ondersteun deur verbinding van addisionele eweknieë (of videogebruikers) toe te laat. Die stelsel het goed presteer en het goeie werkverrigting getoon onder tipiese realistiese beperkinge van die beskikbare bandwydtekapasiteit.

Sleutelwoorde:
eweknie-netwerk, stroming, bandwydte, enkodering, IPTV, lewendig, multimedia, eweknie-uitsendings, multi-uitsending.

To my parents, my sister and Tamara

who supported me through my years of study and through the trying times during my work on this dissertation

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

ACKNOWLEDGEMENT

Firstly, I would like to thank Telkom SA for accepting me into the Centre of Excellence programme. Without this post-graduate study bursary, completion of this work would not be possible.

Furthermore, the NSFAS was a lifesaver during my undergraduate studies.

I appreciate the willingness of my supervisor, Professor J. C. Olivier to supervise my work even though he already had a large number of students and limited time available. His answers to my questions and the time provided for discussion (even though my topic is not within one of his research focus areas) proved to be invaluable.

I would like to thank Hans Grobler for his input and comments regarding my implementation as well as his assistance in setting up a suitable simulation environment for my implementation using the high performance computing cluster (EERC Grid).

My co-students in the CoE programme need to be thanked for allowing time for discussion and for their valuable input. To Jonathan and Christoph, it has been enjoyable working with you and discussing issues related to our respective topics at length. To Auralia, thank you for your support and words of encouragement throughout - especially towards the end.

A special word of thanks to Joe van Zyl for all his guidance and advice over the years. I would not be where I am without you. To Johan Schoeman, thank you for your advice and help with the development (and testing) of my dissertation work. I would also like to acknowledge the support of Johan Slabbert, thank you for listening and motivating me throughout the course of my dissertation.

Lastly, I would like to thank my family for always being supportive and caring during the best and the worst of times throughout my academic career.

CONTENTS

List of Abbreviations

AB	Available Bandwidth
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
AFR	Arrival Frame Rate
AMB	Arrival Media Bandwidth
ASF	Advanced Systems Format
ASP	Advanced Simple Profile
AVC	Advanced Video Coding
AVI	Audio Video Interleave
BP	Baseline Profile
bps	Bits per second
BPB	Bits per Byte
BPP	Bits per Pizel
CABAC	Context-Adaptive Binary Arithmetic Coding
CAN	Content Addressable Network
CDN	Content Delivery Network
CPU	Central Processing Unit
CR	Compression Rate
CRC	Cyclic Redundancy Check
CSP	Current Streaming Point
DCT	Discrete Cosine Transform
DHT	Distributed Hash Table
DoS	Denial of Service
DPCM	Differential Pulse Code Modulation
DRM	Digital Rights Management
DSL	Digital Subscriber Line
DTV	Digital Television
DV	Digital Video
DVB-T	Digital Video Broadcasting-Terrestrial
DVD	Digital Versatile Disk
ESM	End System Multicast

FEC	Forward Error Correction
FIR	Finite Impulse Response
FLAC	Free Lossless Audio Codec
FLV	Flash Video
FMO	Flexible Macroblock Ordering
FMV	Full Motion Video
FPS	Frames Per Second
FTTC	Fiber to the Curb
GMC	Global Motion Compensation
GOP	Group of Pictures
HD DVD	High Definition Digital Versatile Disc
HDMI	High Definition Multimedia Interface
HDTV	High Definition Television
HDV	High Definition Video
HLP	High Latency Profile
HSDPA	High Speed Downlink Packet Access
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDCT	Inverse Discrete Cosine Transform
IETF	Internet Engineering Task Force
IPTV	Internet Protocol Television
IRC	Internet Relay Chat
ITU-T	International Telecommunication Union-Telecommunication
JPEG	Joint Photographic Experts Group
kbps	kilobits (1000) per second
Kbps	Kilobits (1024) per second
kBps	kilobytes (1000) per second
KB	Kilobytes
KBps	Kilobytes (1024) per second
LAN	Local Area Network
LIFO	Last In First Out
LIMD	Linear Increase Multiplicative Decrease
MAC	Medium Access Control
Mbps	Megabits (1048576) per second

MB	Macroblock
MB	Megabytes
MDT	Minimum Distribution Time
MMS	Microsoft Media Services
MP	Main Profile
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group
MSE	Mean Square Error
NAL	Network Adaptation Layer
NAT	Network Address Translation
NSV	Nullsoft Video
OBMC	Overlapped Block Motion Compensation
OCN	Open Content Network
OPEX	Operational Expenditure
OSN	Overlay Subscription Network
P2P	Peer-to-peer
PFR	Playback Frame Rate
PFSP	Partial File Sharing Protocol
PKI	Public/Private Key Infrastructure
POP	Point of Presence
PSNR	Peak Signal to Noise Ratio
PR	Playback Rae
QoE	Quality of Experience
QoS	Quality of Service
QPel	Quarter-sample motion compensation
R-D	Rate Distortion
RAID	Redundant Array of Inexpensive Disks
RDT	Real Data Transport
RGB	Red Green Blue
RIFF	Resource Interchange File Format
RFC	Request For Comments
RLC	Run Length Coding
RSB	Required Server Bandwidth
RSS	Really Simple Syndication

RSVP	Resource ReserVation Protocol
RTCP	Real-time Transport Control Protocol
RTMP	Real Time Messaging Protocol
RTSP	Real Time Streaming Protocol
RTP	Real-time Transport Protocol
SAD	Sum of Absolute Differences
SDP	Session Description Protocol
SHA	Secure Hashing Algorithm
SNR	Signal to Noise Ratio
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
THEX	Tree Hash EXchange
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VCL	Video Coding Layer
VLC	Variable Length Coding
VLC	VideoLAN Client
VLD	Variable Length Decoding
VLS	VideoLAN Server
VOB	Video Object
VOD	Video On Demand
WiMAX	Worldwide Interoperability for Microwave Access
WMA	Windows Media Audio
WMV	Windows Media Video
XML	Extensible Markup Language
XOR	Exclusive OR
XP	Extended Profile

CHAPTER ONE
RESEARCH OVERVIEW

"I think that the Internet is going to effect the most profound change on the entertainment industries combined. And we're all gonna be tuning into the most popular Internet show in the world, which will be coming from some place in Des Moines. We're all gonna lose our jobs. We're all gonna be on the Internet trying to find an audience."

STEVEN SPIELBERG

1.1 INTRODUCTION

This dissertation examines video streaming over a peer-to-peer network architecture. The rapid growth of broadband Internet access technologies (such as ADSL, Cable Internet and Wi-Fi Internet access) as well as mobile high speed connection options (3G, WiMAX and HSDPA are examples) worldwide has resulted in the development of an industry for content rich multimedia creation and delivery over IP networks (IPTV). Providing this content to consumers on a large scale has been problematic due to congestion issues arising when using a traditional *one-to-many* server-client unicast multimedia streaming infrastructure [1]. Multicast streaming solves a number of these problems but is costly and complex to implement. Peer-to-peer networks address this problem directly and reduce the bandwidth[1] load on servers by making use of co-operative data re-distribution during streaming. It has been shown that peer-to-peer networks scale very well [2] making them highly effective for broadcast-size video streaming networks.

Peer-to-peer networking research, development and implementation typically focus on file and distributed content sharing. Current research trends indicate a move towards developing practical peer-to-peer systems in the application layer [3] to enable greater functionality beyond simple file sharing. This dissertation proposes an example application

[1] Where bandwidth is defined as the data volume transfer capability over unit time *i.e.* bits per second

of peer-to-peer networking to facilitate live streaming of any video (primarily a live video source) to a large group of viewers.

Broadcasting video over peer-to-peer is referred to as *peercasting*. Existing peercasting methods and applications are investigated and documented as possible solutions to the large-scale streaming problem. Various complex issues encountered when using peercasting for streaming are addressed and possible solutions provided. The progress towards an example implementation as well as due considerations, features and technical aspects are documented in detail. Performance and functionality is simulated and analysed as a means to present the benefits as well as strengths and weaknesses of implementing a peer-to-peer based video streaming platform.

1.2 PROBLEM STATEMENT

The problem addressed by this work is that of streaming video to a medium to large sized audience or network using relatively low bandwidth Internet connections. The problem of traffic congestion (under load) for servers of multimedia content (specifically for real time or *live* video streaming) is directly addressed.

Various aspects of the problem will be discussed in this dissertation. This discussion is used to drive development of a solution while also offering a framework for analysis and development of a streaming system. Problem areas addressed include the following:

- Choice of video codec and compression technology for streaming
- Optimised distribution scheme for encoded video including efficiency of bandwidth utilisation
- Management of video packets as they travel from the media server to viewers (or clients)
- Ability to encode video from multiple source types by way of modular design
- Seamless playback of streamed data packets
- Managing dynamic peers *i.e.* peer churn

These problems are the main focus of the design, development, analysis and simulation processes for the final proposed implementation.

1.3 OBJECTIVES OF THE RESEARCH

As a starting point for the design process, video streaming and peer-to-peer technologies will be introduced including investigation of relevant implementations as a means to reveal potential pitfalls and possible advantageous solutions to problems that are expected to arise. A variety of peercasting applications are compared and briefly documented to provide a background of selected prominent implementations. The main aim of this investigation is to highlight a need for a simple peercasting application and to develop a basic understanding of the state of the art in the field of peer-to-peer based video streaming.

The main objective of this work is to design and develop a real time video encoding and stream preparation application as well as the necessary management and control applications for distributing encoded video data to connected peers in a peer-to-peer network. This implementation should be simple to implement, perform efficiently and not have any exceptional platform or environmental requirements. Development should be of an open source nature to allow future expansion. Finally, the proposed implementation should be fully scalable supporting both low end networks with few peers as well as networks with a relatively large footprint.

The implementation will be analysed and a basic performance model will be developed accordingly. The research concludes with simulation and experimental results as well as observations made after completing and evaluating the developed implementation.

Research outputs include:

- Overview of peer-to-peer networking and video streaming
- A wide-coverage survey of related peercasting implementations and applications
- Video source server which provides the video stream using real time compression of source video frames
- Tracking server responsible for controlling peer connection requests as well as initialisation, control and management of the streaming network
- Peer application supporting stream construction using received data parts as well as structured communication with the tracking server, source peers and downstream peers that request streaming data parts

- Peer classification to enable peers with higher resources to act as reliable sources and fortify the overall network
- Peer data redistribution architecture for delivering requested stream chunks to connected peers
- Complete reference design description with system specifications, operational characteristics and performance results derived by simulation and experimentation

1.4 ORGANISATION OF THE DISSERTATION

The dissertation commences with **Chapter 2** which focuses on video encoding and streaming. Aspects related to the handling and processing of various types of video are discussed. This chapter introduces concepts that are used to differentiate video compression schemes. The most prevalent and widely supported codecs are compared. Video streaming is introduced (in the context of deployment) in this chapter. Video quality and encoding performance measurement are investigated.

Chapter 3 is an introduction to peer-to-peer technologies and implementations. Peer-to-peer network models and concepts are introduced. Several types of unique peer-to-peer systems are investigated. Performance characteristics and challenges are discussed. Several peercasting schemes are introduced. This examination of peer-to-peer concepts is used as the fundamental base for the design of the peer-to-peer algorithm in the proposed implementation.

Chapter 4 presents a survey of the literature related to the research being carried out. Aspects relevant to the design of the system are studied. Various options for video streaming and peer-to-peer networking are presented and relevant concepts are introduced. An overview of related implementations is provided. Shortcomings of other implementations will be determined such as to pave the way and further develop objectives that are important for the design of a peer-to-peer video streaming system.

Chapter 5 is the starting point for the description of the peer-to-peer streaming model. Operational modules are specified and described on a systems level. This chapter serves as an introduction to the proposed prototype design. The design is compared to other related designs and thus placed in context within the field of peer-to-peer video streaming.

Chapter 6 encompasses the complete design, specification and operational description of the proposed peer-to-peer multimedia streaming model. Detailed models of each functional module are provided. Operational flow diagrams are used to clarify the flow of information

through the system. The design of the system as a whole is presented. Detailed system specifications are provided including the design specifications, implemented technologies, details of the deployment environment as well as specifications for inputs and outputs. The complete peer-to-peer algorithm design is documented.

Chapter 7 is the collation of all results and experimental observations derived by analysing and simulating the proposed system and all relevant parts within the system. The analysis section provides a mathematical overview of all the parts within the system and strives to predict performance under expected deployment conditions. The simulation environment is described and an overview of the methodology used to obtain results and performance characteristics is included. Verification procedures are described. Results are discussed and evaluated for correctness and applicability based on design expectations. Relevant implementation problems and challenges encountered (as well as solutions) are briefly documented. The results and conclusion lead in to the summary of the work as a whole.

Chapter 8 concludes the dissertation with a summary of the work completed as well as the suitability of peer-to-peer video streaming based on investigation of the literature and proof-of-concept analysis and testing. The dissertation concludes with pointers for future development of the topic and related sub-topics.

CHAPTER TWO
VIDEO ENCODING AND STREAMING

2.1 INTRODUCTION

An application that handles video relies heavily on video compression. Raw video is digitally stored frame by frame leading to excessive use of storage space for a long sequence of individual video frames. Video compression or encoding aims to reduce the overall data size of digitised video to make better use of available bandwidth and storage space. Compression also allows for more versatility when working with video.

Streaming of video is dependent on how the video is encoded. Several forms of compression (or more accurately, container formats) make use of indexing and tracking to provide seeking capabilities and greater redundancy for video data. The following sections define and describe the processes of video compression and encoding with the focus on encoding a video source for streaming.

2.2 ENCODING OF VIDEO

Video encoding is the process of converting (by compression) an input video stream so that it is more efficient for transport (by streaming) in terms of data volume, video specification and packaging of video data. The aim of video compression is to produce high quality output without consuming large amounts of data. Highly efficient complex compression algorithms are designed to take advantage of limited storage space and bandwidth by providing a suitably high quality output video stream using complex compression techniques such as prediction and motion compensation.

Several key concepts in the complex process of video compression are introduced below. Almost all video coding methods are based on the following processes: prediction, interframe/intraframe compression, motion compensation and entropy coding.

2.2.1 Prediction

Video compression is typically achieved by implementing a form of intelligent prediction whereby the compressor/encoder searches for redundancies and predictable parts in an incoming video sequence. If such redundancies are found, **only** the difference between consecutive frames needs be stored (*i.e.* encoded). Prediction is the starting point for application of various compressive techniques (*e.g.* Entropy coding).

2.2.2 I-frames, B-frames and P-frames

I-frames (intra coded frames) are encoded without reference to other frames. These frames are individually encoded using lossy compression but without temporal prediction. These frames are used to create a *starting point* for the receiver (decoder) and for recovery from errors [4].

P-frames (predicted frames) are encoded relative to the nearest previous I- or P-frames using forward prediction processing as indicated in Figure 2.1. By using motion estimation, P-frames provide more compression than I-frames. P-frames are also used as a reference for B-frames and for future P-frames. Switching P-frames (SP) are introduced in H.264/AVC coding [5] and are used with switching I-frames (SI) to efficiently switch between bit streams coded at differing bit rates.

B-frames (bi-directional frames) use the nearest past and future I- or P-frames as a reference resulting in bi-directional prediction. This allows B-frames to achieve a high level of compression. Two B-frames are typically found between I- and/or P-frames. B-frames are not used for prediction or encoding of other frames.

Video frames are not necessarily decoded or displayed in the same order as they were encoded. The reason for this is the bi-directional prediction used when encoding B-frames. A typical MPEG-2 encoding example is shown in Table 2.1 where the encoding order differs from the input and playback frame ordering.

2.2.3 Intraframe versus interframe compression

Figure 2.2 shows the basic video compression process which begins with inter- and intraframe compression. Interframe compression is the method where consecutive frames are compared and only the differences encoded. This process is described in detail below. The main

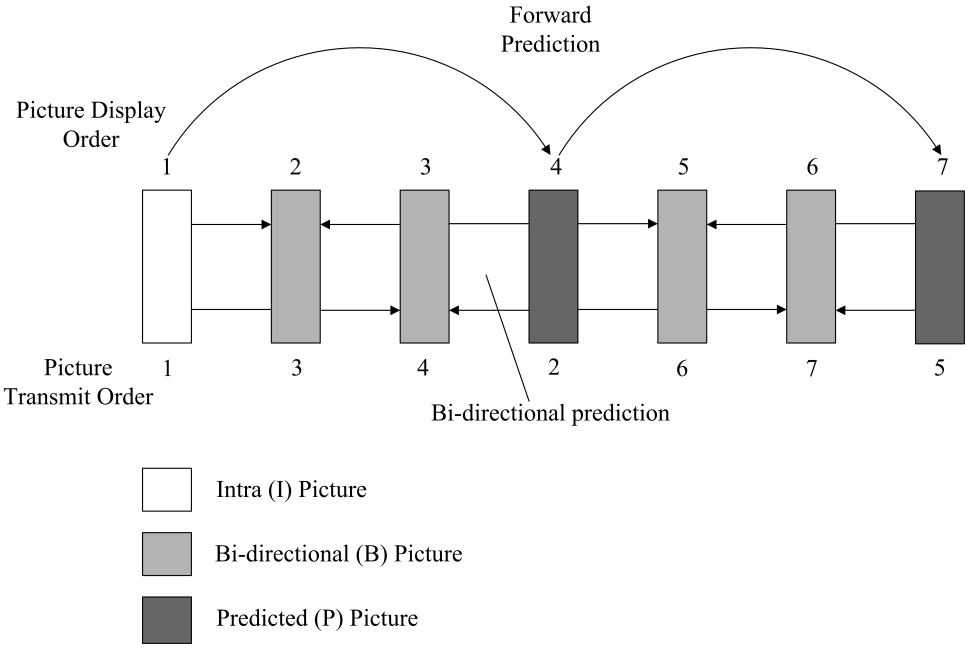


FIGURE 2.1: A sample transmitted segment of I-, B-, P-frames. Arrows indicate inter-frame dependencies (from [4], p. 549)

TABLE 2.1: Comparison of the I/B/P frame encoding and decoding/playback order

Source order (encoder input)	I-1	B-2	B-3	P-4	B-5	B-6	P-7	B-8	B-9	P-10	B-11	B-12	I-13
Encoding order (coded bitstream order)	I-1	P-4	B-2	B-3	P-7	B-5	B-6	P-10	B-8	B-9	I-13	B-11	B-12
Decoder output (display order)	I-1	B-2	B-3	P-4	B-5	B-6	P-7	B-8	B-9	P-10	B-11	B-12	I-13

drawback of using interframe compression arises when frames are lost prior to video reconstruction. Decoding the video data stream relies on having an approximation of the original frame available (derived from the previously decoded frame and delta information) along with the differences for consequent frames. If this reconstructed original frame is not properly and reliably transmitted, then the reconstruction of subsequent frames becomes impossible leading to a large discontinuity in the video stream. Interframe is a highly efficient compression scheme in terms of storage space or equivalently transmission bit rate consumption.

Intraframe compression handles each individual frame separately and compresses frames one by one as if they were normal images. Frames encoded using this method will have a size which is dependent on the nature of the image (textures, details, colours *etc*). The basic video

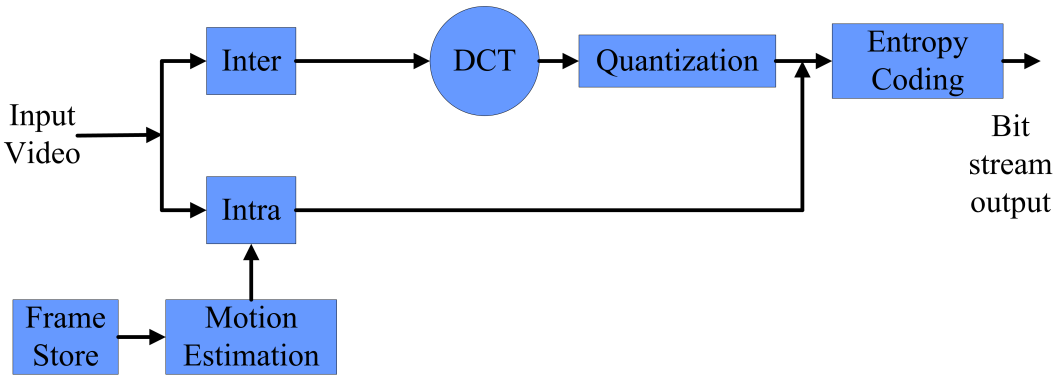


FIGURE 2.2: The basic video compression process

coding steps are indicated in the flow diagram in Figure 2.2.

2.2.4 Keyframes

Keyframes are a special type of I-frame which are only spatially compressed (the complete frame is encoded). These frames are used as reference points for subsequent interframes and are used to manage seeking within a video stream or segment. While still compressed, keyframes are large in comparison to P- and B-frames and thus, a high keyframe frequency produces relatively large output stream size. For low motion video, a high keyframe interval provides adequate picture quality whereas a smaller keyframe interval is required for encoding high or fast motion video.

2.2.5 GOP

The Group of Pictures (GOP) is the encoding block sequence of I-, B- and P-frames (the GOP length is known and thus allows the encoder to contain bi-directional and predictive encoding to a fixed length around the target frame for encoding). For random access within the group, the sequence must begin with an I- or a B-frame and end with I- or P-frames. B- and P-frames are found between these I-frames. The GOP can be of variable length depending on the type of video being encoded. An example is fast moving or complex video which is more efficiently encoded using a short GOP length. Typical GOP length is between eight and 25 frames depending on the nature of the playback system. A smaller GOP size results in lower compression since fewer frames are used for prediction.

2.2.6 Macroblocks

A macroblock is an 16 x 16 block of pixels and is used for motion estimation. Each and every frame is divided into these macroblocks which are the basic unit of encoding for a video stream. Macroblocks specify the Y luminance component and the two chrominance components C_b, C_r. Macroblocks may be organised in slices representing subsets of each frame which can be decoded independently. H.264/AVC takes this macroblock division a few steps further by sub-dividing each macroblock in additional phases (*i.e.* a 16x16 macroblock becomes four 8x8 smaller partitions and possibly even sixteen 4x4 sized blocks). The reason for doing this is to obtain a higher level of precision for motion estimation as a means to encode high definition video content (HDV or HDTV).

Flexible Macroblock Ordering (FMO) is a technique (used by H.264/AVC) where macroblocks in a frame are assigned to one or several slice groups which are transmitted separately. If a slice group is lost in transmission, samples in spatially neighbouring macroblocks that belong to other successfully transferred slice groups are used for efficient error concealment.

2.2.7 Motion estimation and compensation

Motion estimation entails finding optimal or near-optimal motion vectors by measuring the movement of elements within each frame sequence. Optimal motion vectors have the most acceptable compromise between the amount of block prediction error and the number of bits required to store the motion vector data. Figure 2.3 shows a basic example of motion estimation. A *full search* is a motion estimation technique where all possible motion representations are exhaustively tested. A faster but less optimal method (with respect to rate distortion) utilizes a coarse search grid for first approximation with refinements to this grid in subsequent steps.

Motion compensation is used to describe the difference between consecutive frames based on the changes of each section within the original frame as determined using motion estimation. The compensation of the effect of motion on the new frame is a way of removing temporal redundancy. Basic motion compensation implies subtracting the reference frame from the following frame to produce a *residual* which occupies less space and can be compressed and stored (or transmitted) using a low bit rate. The decoder adds the residual to the reference frame to produce the original video sequence. Figure 2.4 shows a basic example

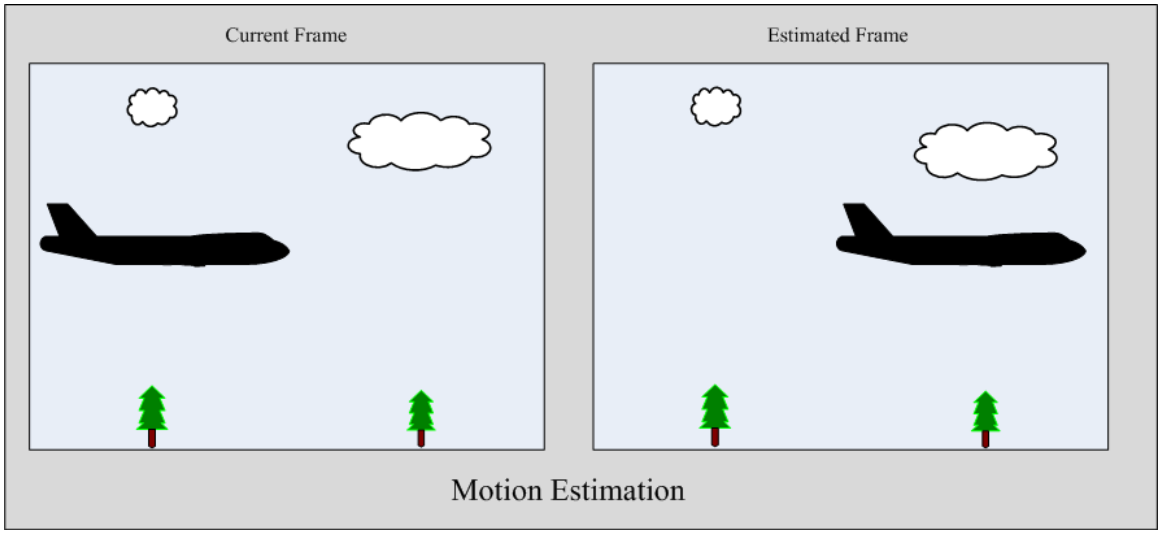


FIGURE 2.3: Simple example of motion estimation for a single frame where the subsequent frame is predicted

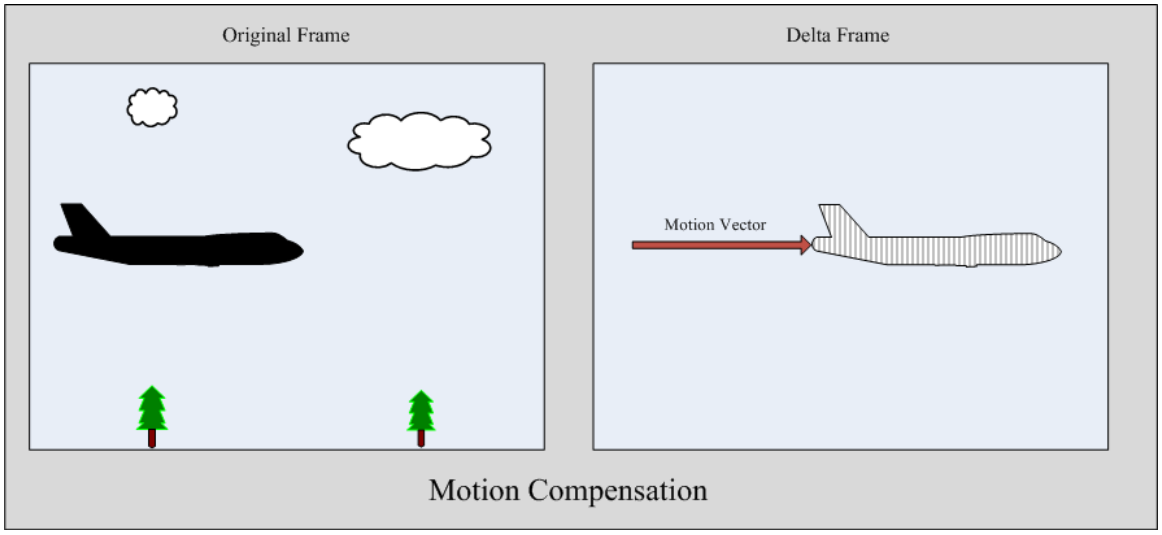


FIGURE 2.4: Simple example of motion compensation where the delta frame as determined by estimation is encoded using a motion vector

of compensation after estimation. Global Motion Compensation (GMC) is a process whereby the reference frame is panned and shifted such as to create a prediction of future frames. This movement is based on so-called *warp* points and is based on per-pixel estimation as opposed to macroblock estimation.

Motion compensation divides the frame into blocks of pixels which are individually compared to the associated block in the reference frame. The difference in blocks caused by movement across frames is represented by a motion vector which is encoded in the bitstream. This method has a negative side effect where discontinuities are introduced at block borders

which translate to highly visual artefacts in encoded video.

2.2.8 Discrete Cosine Transform

The first step in the image/video compression process is the subdivision of the image data into macroblocks as discussed earlier. Each macroblock is transformed into its frequency domain representation using the discrete cosine transform (DCT). The DCT operates on a typical macroblock size of 8x8 which produces 64 frequency domain co-efficients for 64 input signals.

The DCT helps separate more perceptually significant information (low frequency DCT co-efficients) from less perceptually significant information (high frequency DCT co-efficients or smaller visual changes). Subsequent steps in compressive algorithms encode lower frequency DCT co-efficients with high precision while allocating fewer or no bits to high frequency components thus discarding information that has a lower perceptual effect. The inverse DCT (IDCT) recovers the coded DCT co-efficients to an 8x8 block of pixels.

The *N* x *N* two-dimensional DCT [6] is defined as:

$$F(u,v) = \frac{2}{N}C(u)C(v)\sum_{x=1}^{N-1}\sum_{y=1}^{N-1} f(x,y)[cos\frac{(2x+1)u\pi}{2N}][cos\frac{(2y+1)v\pi}{2N}] \tag{2.1}$$

$$C(u), C(v) = \left\{ \begin{array}{ll} \frac{1}{\sqrt{2}} & \text{for } u, v = 0 \\ 1 & \text{otherwise} \end{array} \right)$$

The inverse DCT is defined as

$$f(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)F(u,v)[cos\frac{(2x+1)u\pi}{2N}][cos\frac{(2y+1)v\pi}{2N}] \tag{2.2}$$

where

x,y are spatial co-ordinates in the *N* x *N* image macroblock and

u,v are co-ordinates in the DCT co-efficient block

Figure 2.5 shows how the DCT is used to map values obtained from an 8x8 pixel macroblock to an 8x8 block of DCT co-efficients. One-dimensional DCT requires 64 multiplications and eight one-dimensional DCTs for each 8x8 block. Fast two-dimensional DCT requires 54 multiplications and 468 additions and shifts.

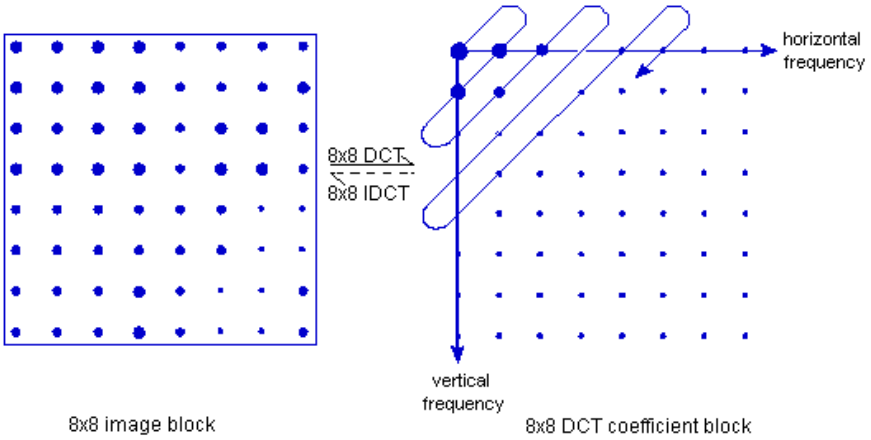


FIGURE 2.5: The relationship between DCT and IDCT. Pixel value and DCT co-efficient magnitude are represented by dot size (from [6], p. 258)

After the DCT process, the top left corner has the highest co-efficients and the bottom right the lowest which makes compression easier. Co-efficients are numbered in a *zig-zag* way from top left to bottom right so that the smaller co-efficients will be at the end *i.e.* so that the greatest number of zeros can be grouped together. After quantisation, it is easier to drop the smaller co-efficient values in order to reduce precision.

2.2.9 Quantisation

Quantisation is the first step in encoding DCT co-efficients where perceptually insignificant information is discarded. The process of quantisation results in co-efficients being scaled in such a way that any extraneous information can be discarded. Typically, DCT co-efficients are scaled from a large numeric range (such as positive and negative whole numbers of a certain size *e.g.* 16-bit co-efficients) to a smaller range of rounded values (in the example of 16-bit co-efficients, these can be quantized to values between -255 and 255). De-quantisation implies the inverse process where the scaling factor is applied producing a similar image with minimal visual artefacts.

2.2.10 Coding

The second step in the video compression process (after DCT quantisation) is the encoding of the quantised DCT co-efficients using as few bits as possible. Statistical properties of the quantised co-efficients are exploited to minimize the number of bits required. An example of such a property is run length coding (RLC) where runs of repetitive numbers (for example, zero) are encoded using a specific unique value that when decoded results in the same

repetitive sequence in the data stream.

Variable length coding (VLC) is used after RLC to encode common sequences with short code words and lesser frequent sequences with longer code words for efficiency. Huffman coding is an example of VLC where the number of bits in each code word is optimized based on the frequency of occurrence of each symbol.

Variable length decoding (VLD) is computationally expensive as it typically requires an average of 11 operations per input bit [7]. Processing requirements for decoding are therefore proportional to the encoding bit rate. VLD suffers heavily upon occurrence of bit errors in the middle of an encoded stream due to the nature of the reconstruction process. This is overcome by injecting *resynchronization markers* into the encoded bit stream. If an error is detected, the decoder searches for the next resynchronization marker and continues decoding the video stream from that point onwards.

2.2.11 Entropy coding

Typical compression schemes aim to achieve minimal redundancy in order to be as storage space efficient as possible while retaining a high level of quality. Lossless compression aims to achieve compression on data without any loss of information while still offering some storage space saving. The smallest theoretical size for lossless compression is known as the entropy of the source. Any compression beyond this point will result in the compression becoming lossy.

2.2.12 H.263

Having been designed for very low bit rate applications, H.263 uses a block motion-compensated DCT structure for encoding. H.263 is significantly optimized for coding at low bit rates. Notable features of H.263 include motion compensation with half-pixel accuracy and bi-directionally coded macroblocks.

2.2.13 H.263+ (H.263v2)

H.263+ is an extension of H.263. SNR scalability and spatial scalability are provided. Advanced intra coding is implemented to improve compression efficiency for intra MB encoding by using spatial prediction of DCT co-efficient values.

2.2.14 MPEG-2 encoding

The MPEG-2 encoding process encompasses most of the techniques and steps described above. Input video can be in one of two formats: progressive scan or as an interlaced video stream. In progressive scan streams, a frame is the basic unit of encoding whereas for interlaced streams the basic unit may be either a field or a frame. The assumption for MPEG-2 encoding is that the video stream is already digitized and typically transmitted as a sequence of uncompressed frames (or can be considered relatively uncompressed).

Incoming video frames are encoded using the GOP approach with I-, B- and P-frames. Motion estimation is used to implement compression on consecutive frames and groups of frames in a scene. For MPEG-2, each image or frame is separated into one luminance and two chrominance channels which make up the macroblocks. Macroblocks are used to determine motion vectors used in the encoded bitstream.

Compression efficiency is achieved for I-frames by exploiting spatial redundancy in images by averaging sections (differences within a single frame) where the human eye is unable to distinguish minor changes in colour. P- and B-frames take advantage of temporal redundancy where adjacent video frames are often very similar to each other. P-frames can typically be 10% of the size of I-frames while B-frames are 2% of the typical size of an I-frame. This space saving is offset by the relative computational complexity of encoding B-frames and P-frames when compared to that of I-frames.

The entire encoding process is summarised in the block diagram in Figure 2.6.

2.2.15 MPEG-4 coding

MPEG-4 uses reversible VLC where code words are chosen such that they can be decoded in the normal forward direction as well as backwards. This allows the decoder to decode frames in both directions from the synchronization marker immediately after an error in the encoded bit stream is encountered. This results in less dropped video if an error occurs.

Given that MPEG-4 is an improvement on MPEG-2, the basic steps are identical (as indicated in Figure 2.2). MPEG-4 has three groups of tools that are used to provide robustness. Re-synchronisation tools enable the re-synchronisation of the bitstream by the

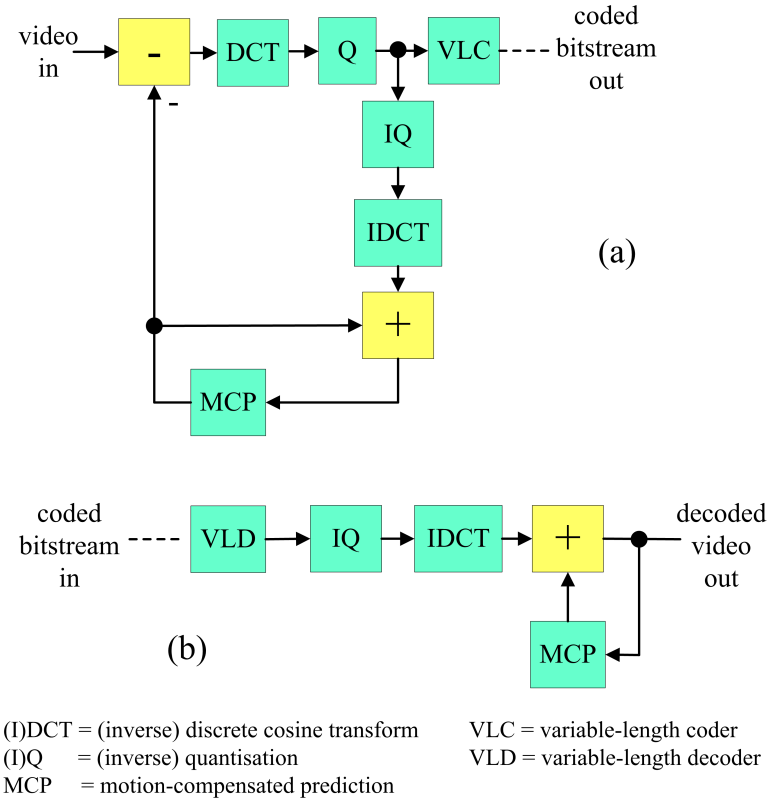


FIGURE 2.6: The MPEG-2 bitstream structure: (a) motion compensated DCT coder and (b) motion compensated DCT decoder

decoder after error. Data recovery tools can be used to recover data lost due to errors while error concealment tools hide the lost data.

Other MPEG-4 enhancements include:

- Shape coding: for describing the boundaries of objects within a frame
- Sprite coding: ability to handle static (or near-static) text or graphic overlays separately for one time encoding (instead of every frame)
- Scalability: a baseline stream is encoded with several optional enhancements allowing for real time transmission channel adaptation
- Profile encoding: specifies which of the MPEG-4 tools are used, resolution levels for coding processes as well as type of video compression implemented

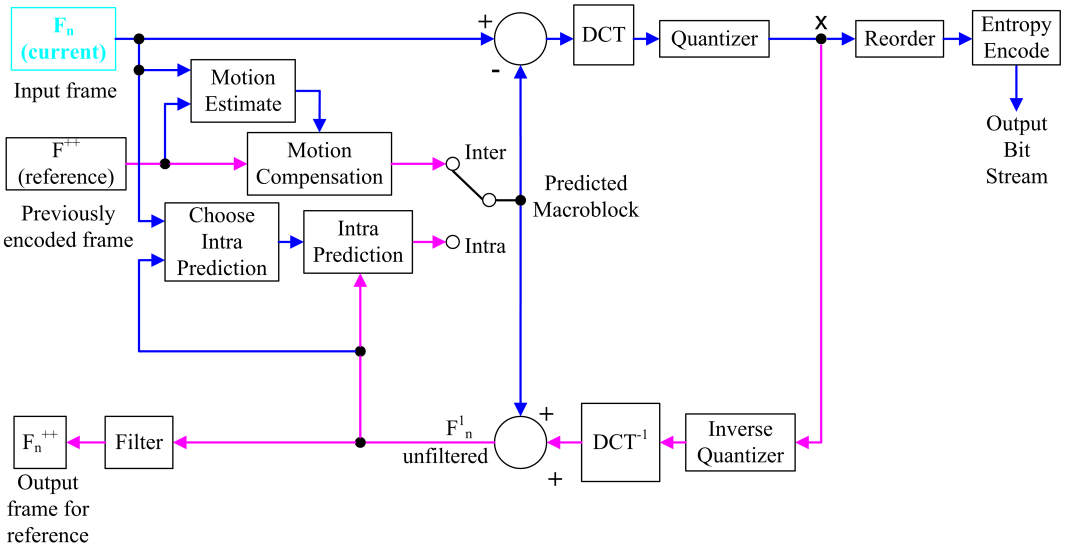


FIGURE 2.7: The MPEG-4 AVC approach employs some prediction within a video frame as well as motion compensation from other frames to form predicted frames. This approach is similar to but slightly more optimized than MPEG-2.

2.2.16 H.264/AVC encoding

The basic H.264/AVC (MPEG-4 Part 10) coding structure is shown in Figure 2.9. Notable differences from the encoding processes of other schemes (such as MPEG-2) include spatial domain prediction (by referring to neighbouring samples of already coded blocks) instead of prediction in the transform domain [8]. During transform coding of the prediction residual, H.264/AVC uses a separable integer transform on 4x4 blocks instead of the 4x4 DCT transform.

As with other MPEG schemes, AVC uses mode selection to choose between inter-frame and intra-frame compression. The way that I-frames are encoded is unique. Instead of encoding each frame in isolation, the AVC approach uses information from nearby blocks in the frame to predict the current block resulting in I-frames being more optimally compressed. During inter-frame operation, AVC can use up to five frames for motion estimation (as compared with two for MPEG-2 *i.e.* one before and one after) as shown in Figure 2.8. AVC can work with sub-blocks within the 16x16 macroblock allowing for motion estimation and compensation accuracy up to $\frac{1}{8}$-pixel.

The final improvement made by AVC over other MPEG methods is found in the entropy coding phase. Instead of schemes such as VLC and Huffman coding, MPEG-4/AVC uses context-adaptive binary arithmetic coding (CABAC) which enables the coding process to

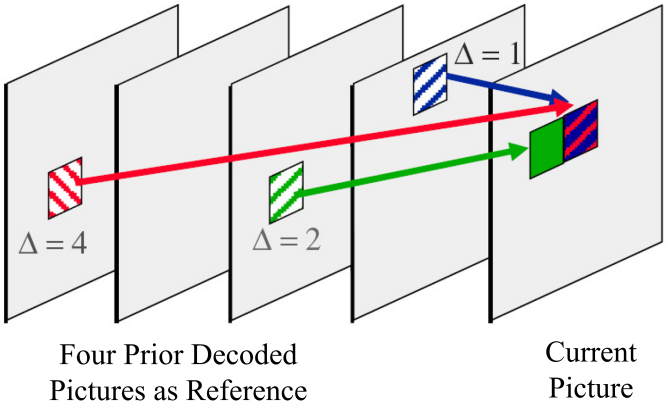


FIGURE 2.8: MPEG-4 H.264/AVC multi-frame prediction ([9], p. 2)

achieve efficiencies of less than one-bit output per input bit [8] or a 10-15% reduction in overall bit rate [9].

In a comparison with MPEG-2 Visual (MP), H.263++ (High Latency Profile) and MPEG-4 Visual (ASP with Quarter Sample or QPel motion compensation, GMC and deblocking/deringing filters applied), H.264/AVC (indicated by the ITU-T name: H.26L) consistently outperformed all other standards [9]. The results of the comparison are indicated in Figure 2.10 where Luma PSNR (Y-PSNR) is compared at different encoding bitrates. H.264/AVC requires a lower bitrate for the same or higher video quality. In addition, a very high rate saving (on average 70%) is achieved compared to MPEG-2. These gains are offset by the increased encoding complexity and resultant processing overhead.

2.2.17 Encoding profiles and levels

Video coding schemes are typically designed to support various encoding profiles. This allows for standardization of the bitstream (for decoding) and thus leaves the design of the encoder up to various individual implementations. Applications ranging from low bit rate streaming to HDTV broadcasting have given rise to the creation and specification of a variety of profiles and levels.

The **Simple Profile** uses low bit rate and low resolution for bandwidth limited applications such as cell phones as well as low end video conferencing and streaming systems. The **Advanced Simple Profile** features the following:

- Support for B-frames

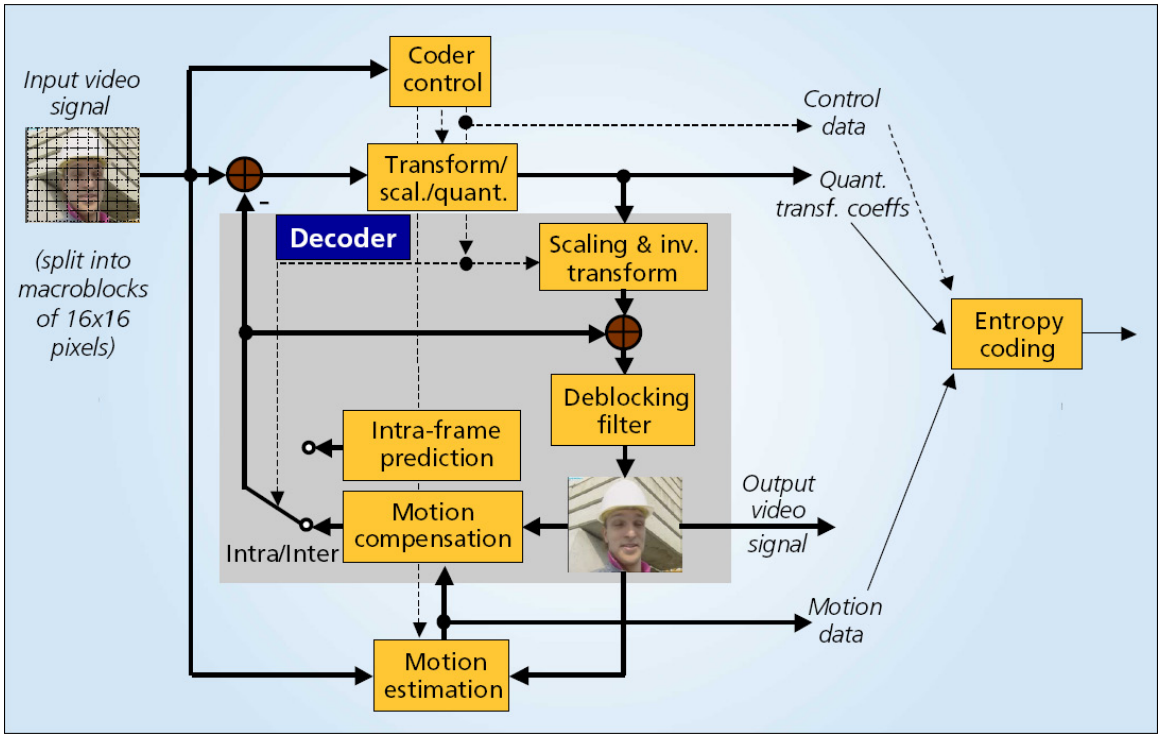


FIGURE 2.9: Basic coding structure for H.264/AVC ([9], p. 2)

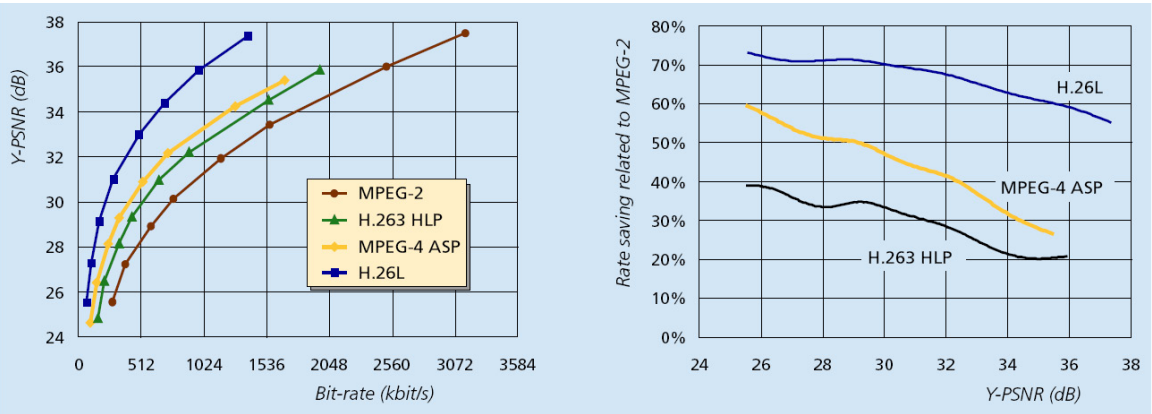


FIGURE 2.10: Y-PSNR comparison for various codecs at various bitrates and the bitrate savings achieved compared to MPEG-2 ([9], p. 9)

- Global motion compensation
- Quarter-sample motion compensation
- MPEG-compatible quantisation
- Support for interlaced video

These two profiles are defined as part of MPEG-4 Part 2.

H.264/AVC (MPEG-4 Part 10) includes the following commonly used profiles (amongst others):

1. Baseline Profile (BP): video conferencing and mobile applications, low bit rate and minimal processing load.
2. Main Profile (MP): original consumer grade profile for storage and broadcasting.
3. Extended Profile (XP): streaming profile, high compression with additional robustness.
4. High Profile (HiP): primary profile for storage and broadcasting of high definition video (including HD DVD and Blu Ray).

Levels specify certain encoding characteristics as a way to achieve standardization in decoding. Major aspects specified include: maximum macroblocks per second, maximum video bit rate (within profile) and maximum resolution. Each encoding profile has a number of levels specified and thus every single encoding process should meet the specifications of a specific profile and level.

2.2.18 Reducing decoding artefacts

Artefacts in decoded video streams occur due to poor encoder implementation, source video that is challenging to encode (*i.e.* high motion video) or too low an encoding bit rate for the given video resolution and frame rate. This is often the case given that many designers and developers sacrifice video quality to save storage space and/or streaming bandwidth.

Macroblocking (or blocking artefacts) occurs when important information is deemed less important by the encoder. The result of this is components of the video or image where macroblocks are heavily compressed and thus poorly represented. This produces sharp differences on macroblock boundaries which makes individual macroblocks distinguishable

FIGURE 2.11: Performance of an H.264/AVC de-blocking filter. Left: original Foreman image. Right: after de-blocking filter ([9], p. 7).

in the resultant output image. Deblocking filters can be implemented as an additional step in the decoding process (post-processing) or can be integrated into the decoder itself (in-loop implementation).

Ringing artefacts occur when the decoder discards too much information when dequantising the DCT co-efficients. These artefacts are visual discontinuities and distortions around the edges of features within the video.

De-blocking and de-ringing are accomplished using finite impulse response (FIR) filters to remove visual artefacts. De-blocking filters are applied at the edges of macroblocks and work by blending the block edges with the neighbouring blocks. De-ringing is typically an adaptive process where a low pass filter smooths areas near the edges that cause artefacts. This smoothing barely affects the pixels themselves in order to prevent blurring. An example of the effect of a de-blocking filter is shown in Figure 2.11 where an H.264/AVC de-blocking filter is used to improve a highly compressed sample video clip.

2.3 VIDEO STREAMING

Video streaming refers to playing video from a source without having to have or download the entire video beforehand. When compared to stored video, streaming video presents a number of unique challenges and is fundamentally different (for example, video is only seekable in one direction). The unique nature of streamed video results in a slightly different encoding method and places greater emphasis on the ratio between encoding speed and output quality of

encoded video as well as a number of other trade-offs. This section introduces video streaming as well as some basic concepts, encoding considerations and stream deployment scenarios.

2.3.1 Basic concepts

Streaming systems are designed with the eventual implementation network in mind. These systems are designed based on a number of metrics, the most important being available bandwidth for stream utilization. Packet latency, packet loss rate and protocol overhead/efficiency are also important design considerations.

Traditional unicast streaming systems are designed based on the following calculation for required server bandwidth:

$$\text{RSB} = \text{streaming bit rate} * \text{number of clients} \tag{2.3}$$

For example: RSB = 512Kbits/s * 200 = 100Mbits/sec to provide each client with an identical stream. It can be seen that a unicast streaming server typically reaches the upper limit in terms of available outgoing bandwidth relatively quickly for a typical Internet connection. Packet latency (due to congestion) is a key factor due to the effect it has on stream reconstruction and synchronisation between clients and the server (especially when packets are dropped).

2.3.2 Types of streaming

There are two distinct ways of implementing video streaming. The first option is progressive download (or pseudo-streaming) where the stream is downloaded (not necessarily in play order) into a buffer from beginning to end. Once the buffer is sufficiently full, playback of the stream can begin. The progressive download method has the following characteristics:

- suitable for small files only
- for longer streams, average data download rate must match or exceed encoding bitrate
- uses TCP/IP for data transfer (reliable, retransmissions will be sent if required)
- simple to implement (the streaming server can be a regular HTTP web server)
- playback only begins when a certain amount of data has been successfully received
- streaming media is saved to a file on the viewer's PC

The second approach is known as traditional or regular video streaming. This approach is more suited for streams with a longer lifetime. Typical characteristics of a live stream include:

- any length stream can be supported
- higher implementation complexity since a specialised streaming server is required
- UDP/IP is used for data transfer (unreliable, no retransmissions)
- playback begins immediately with minimal buffering
- streamed data is played back and subsequently discarded (thus, not stored to file)

2.3.3 Streaming protocols

Several protocols have been designed specifically for streaming or with streaming applications in mind. Along with these protocols, a number of existing protocols have also been applied and/or adapted for streaming. The following is the list of protocols that are used in video streaming as well as their respective RFC numbers and implementation notes.

- **RSVP**
  Resource ReserVation Protocol (RFC 2205 [10]), RSVP applicability statement (RFC 2208) provides a guide to RSVP deployment, Message processing rules (RFC 2209)
- **RTP**
  Real-Time Transport Protocol (RFC 1889) with RTP Payload type (RFC 1890) and other RTP fields (RFC 2250)
- **RTCP**
  Real-Time Control Protocol (RFC 1889 [11]) which is a part of RTP
- **RTSP**
  Real-Time Streaming Protocol (RFC 2326 [12]) - the primary streaming protocol
- **SDP**
  Session Description Protocol (RFC 2327 [13] and RFC 4566 [14]) used primarily in video conferencing and RTSP streaming session establishment
- **HTTP**
  HyperText Transport Protocol (RFC 2616 [15]) typically used for progressive download

The principal network-layer communication protocol (as with all other Internet applications) is IP. For RTP/UDP streaming, each IP packet is made up of a header (40 bytes) and a payload.

Transport-layer protocols include TCP and UDP. Notable differences between the two protocols are highlighted below:

- **TCP** (RFC 793 [16])
  Reliable, two way communication with FEC. Data flow is controlled to manage the download rate allowing increasing of the packet rate until congestion is encountered at which point data rate is aggressively slowed down. Allows repeat requests for lost or heavily delayed packets.

- **UDP** (RFC 768 [17])
  Unreliable, one way packet delivery with no flow control or support for repeat requests. Packets are smaller and thus more efficient. Basic error detection. Packet delivery rate should exceed encoded stream rate as there is no rate transfer control (for optimal use of the transmission channel) in the UDP protocol.

2.3.4 Architecture for streaming

A framework architecture for video streaming can be divided into six distinct areas [18] and defined as follows:

1. **Media compression and encoding**
   The choice of compression scheme or codec as well as encoding bitrate (and various other parameters) is a critical design element within the streaming architecture.

2. **Application-layer QoS control**
   Involves congestion and error control. Congestion control prevents packet losses and reduces packet delay while error control includes techniques such as FEC, retransmission, error-resilient encoding and error concealment to improve video quality if packets are lost.

3. **Media distribution services**
   Adequate network layer support is essential to reduce transport delays and packet loss in order to provide more reliable stream delivery.

4. **Streaming servers**
   These are servers that operate under timing constraints and handle interactive stream

controls (seeking, play/pause *etc*). Typically, streaming servers wait for RTSP requests (using SDP) and serve requested streams using RTP.

5. **Media synchronisation at receiver side**
   The receiver should be able to decode received streams and synchronise received audio and video tracks.

6. **Protocols for streaming media**
   Protocol requirements include addressing (IP), transport (UDP) and session control (RTSP). RTP is typically used for transporting streaming data.

2.3.5 Encoding for streaming

Most video coding methods find application in video streaming. However, for optimized video streaming, several modifications to the encoding process can be integrated.

Packet loss can often disrupt video streams by degrading I-frames and thus a batch of P-frames and often B-frames. One small error becomes a significantly large disruption to the video stream [19]. Video coding needs to be carried out in such a way that the effects of such packet loss are minimised.

For video streaming, it is essential to support some form of buffering to compensate for an unstable delivery platform. In order to function with a buffer, the GOP and video *chunk* sizes need to be sufficiently small. Each chunk (segment of video of a pre-defined size) that is streamed needs to be playable as soon as possible without relying on a large amount of subsequent data. Management of the minimum video chunk size is therefore a critical consideration in video streaming.

In order to maintain a constant video stream, video needs to be encoded to be adaptable to the delivery system's available bandwidth. This can be achieved by encoding at the minimum bitrate (*i.e.* to match the guaranteed value for minimum available bandwidth to each node) or using a coding system that supports dynamic control of the bitrate (such as layered encoding). Real time encoding for streaming ideally requires a feedback path as well as a bandwidth sensing system such that small changes in available bandwidth can be accounted for with compensating bitrate changes during adaptive real time encoding.

Reassembly of received video chunks should be very simple. There should be little or no dependency between video chunks so that while a given video chunk or set thereof is being played, subsequent chunks can be prepared (including error correction or data re-transmission if required) for playback. The process of segmenting initial encoded video (and reassembling before playback) should be computationally simple to avoid any additional processing complexity during both encoding and decoding.

2.4 REAL-TIME CONSIDERATIONS

Encoding of video data in real time is subject to two main factors: encoding complexity and available processing resources. Real time encoding is highly dependent on the implementation platform and available processing power. In general, various adjustments can be made to encoding algorithms to lessen processing load (at the expense of video quality) if required.

Adjustments that can be made for faster encoding speed include actively encoding fewer B-frames and more I-frames (and consequently increasing the overall bitrate requirement).

H.264/AVC has been developed with support for real time applications (such as TV broadcasting) and is designed to fully support real time decoding. Several enhancements can be turned off to enable support in lower grade decoding hardware. The CABAC entropy coding component in H.264/AVC continually updates the statistics of incoming data to the encoder and adaptively adjusts the algorithm for better performance using a process known as context modelling.

2.5 QUALITY

Video quality can be measured in a few basic ways. The concept of quality has different meanings in different contexts. Fluid delivery of video (*i.e.* no skips or pauses) implies good quality. Reduced effects of noise (such as sharp edges, blocking and ringing) by smoothing produces a higher overall perceivable video quality.

A traditional metric for visual quality is PSNR (peak signal to noise ratio) which is a measurement of the quality of re-construction of an image after compression and decompression. PSNR is a measurement of the relative effect of noise on the optimally

encoded video signal when compared to the original source (uncompressed) signal.

The calculation used to determine PSNR is the mean squared error (MSE) which is calculated for two *m* x *n* images *I* and *K* where one image is an approximation (with noise) of the other derived by decompressing the originally compressed image. This calculation is carried out as follows:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \| I(i,j) - K(i,j) \|^2 \tag{2.4}$$

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{MAX}_I^2}{\text{MSE}} \right) = 20 \log_{10} \left(\frac{\text{MAX}_I}{\sqrt{\text{MSE}}} \right) \tag{2.5}$$

MAX_I is the maximum pixel value of the image (255). Typical values for PSNR in image compression lie between 30dB and 40dB. Typically, the PSNR is calculated for the Y component and thus indicated as Luma-PSNR or Y-PSNR since the human eye is most sensitive to Luma information.

2.6 ENCODING PERFORMANCE

As encoding performance is highly dependent on system architecture, this section only describes metrics for measuring and estimating the performance of a video encoding scheme. In general, a well performing video encoder is able to receive and process incoming video in both compressed (using a frame-by-frame frameserving process) and uncompressed (such as RGB or YC_bC_r) formats.

Encoding speed is typically measured in frames per second. The FPS measure should be normalised to the colour encoding scheme used (PAL or NTSC) to provide a useful measure of encoding speed. For example, an encoding rate of 32 FPS under on an NTSC system implies an encoding speed of 32 / 29.97 = 1.067x true encoding speed. This means that video is sent out of the encoder 1.067 times faster than it is received at input. Thus, the real time encoding requirement is satisfied.

An encoding speed of $\leq$ 1.0x will likely result in skips or pauses in the video stream as the distribution channel waits for the encoder to *catch up* with the incoming video source. The end effect in this situation will be a stuttered stream at the streaming server's output.

2.7 SUMMARY

This chapter introduced the concepts of video encoding and video streaming. Several aspects related to video encoding were introduced. These include the steps during encoding as well as reference methods of encoding used by common video encoders. Elements in the encoding process as well as encoding techniques and processes were discussed.

Different encoding schemes (*i.e.* codecs) are described. Along with these codecs, the concepts of encoding profiles and levels are introduced. Throughout the chapter, the focus is on encoding in real time or more specifically encoding for streaming.

The last part of the chapter describes encoding specifically for streaming. This includes several streaming considerations as well as a reference architecture for a stream encoding system. The chapter is concluded with a description of two important encoding measures: encoding quality and encoding performance (speed). The following chapter describes peer-to-peer as a network architecture and includes peer-to-peer implementation examples.

CHAPTER THREE
PEER-TO-PEER

"Give and ye shall receive"

BRAM COHEN, BITTORRENT CREATOR

3.1 INTRODUCTION

The notion of peer-to-peer has existed as long as networks have been constructed using regular workstations. Several legacy systems are based on peer-to-peer (Usenet being the largest and oldest). The massive growth of the Internet has resulted in the unicast (or traditional server-client) model becoming the primary deployment approach for all web applications. This has led to congestion and restrictions (or ever increasing resource requirements) on the number of users making use of the provided service when the user count increases.

Peer-to-peer technologies exploit the available resources of connected peers within the network and thus aim to reduce the load on content servers. Additional benefits include privacy, security and redundancy. This chapter presents selected relevant aspects of peer-to-peer leading to an introduction to implementation of peer-to-peer for multimedia streaming.

3.2 PEER-TO-PEER MODELS

The true definition of peer-to-peer implies connectivity to a large network using only connections with neighbours and without requiring a server-based network structure. However, the definition is typically extended to include all situations where network nodes interface directly and pass messages directly to other nodes (as compared to sending messages to others through the server such as is the case with IRC, FTP *etc*).

Peer-to-peer networks typically fall into two categories based on their structure and whether or not a server forms part of the network. These two main network structures are described below.

3.2.1 Pure peer-to-peer

A *pure* peer-to-peer network has no server but instead all nodes act as servers and clients (sometimes referred to as *servents*). In this model, all nodes in the network are identical within the hierarchy. Conceptually, all nodes are directly connected in a mesh structure to offer redundancy and to remove the need for intelligent routing. However, in practice this is not the case. Routing and node management is accomplished by using the notion of an overlay network with distributed network control and routing tables. Typically, the peer-to-peer network is an overlay network and makes use of an underlying network for message passing and communication control.

As discussed earlier, significant research into the application and development of peer-to-peer overlay networks is being carried out. Several overlay networks have been proposed and implemented (Chord, Tapestry, Kademlia *etc*) using DHT for management of information within the network.

3.2.2 Hybrid peer-to-peer

A peer-to-peer network is known as a *hybrid* network when the network makes use of a centralised server for establishment, routing, management and control of the peer-to-peer network. The server co-ordinates peer actions within the network and does not take part in the actual message passing or data transfer within the network (Figure 3.1). A hybrid peer-to-peer network has a single point of failure (server) and typically offers connected peers less privacy since the server can log all connections and resource requests within the network.

The most prevalent hybrid peer-to-peer networks are Napster, IRC @find and networks constructed using Bittorrent. There are two types of hybrid peer-to-peer networks:

1. **Centralised indexing**

   A server is used to maintain a database of all peers and the data that they are sharing. This can be extended to a server that maintains updated information on exactly which stream peers are receiving and at which point in the stream an end user is at. This system

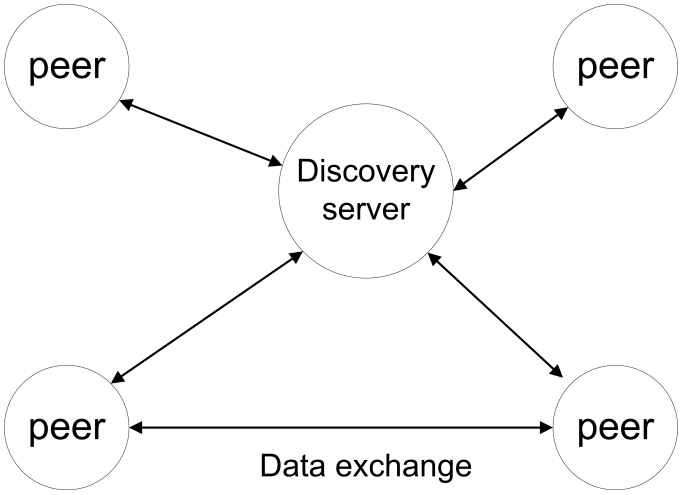


FIGURE 3.1: A basic hybrid peer-to-peer structure where a server is used for node discovery and all data is transferred directly between peers

requires continuous refreshing of stored information. The basic network structure is shown in Figure 3.1.

2. **Decentralized indexing**

   Similar to centralised indexing networks, peers connect to the server for discovery of other peers. However, subsequent requests (for data or streams) are passed to other peers or distinguished peers (sometimes called supernodes or super peers). These supernodes are arranged in a distributed way and form a layer between the level of regular peers and the discovery server. A node can automatically become a supernode if it is detected that the node has sufficient resources. An example of the basic network structure is shown in Figure 3.2.

3.3 RESOURCE SHARING

Peer-to-peer networks are designed and constructed with resource sharing in mind. The main objective is to redistribute the cost or load when compared to a typical one-to-many client-server network. Other forms of distributed processing are also made possible using peer-to-peer networks.

3.3.1 Bandwidth

The most important benefit derived from implementing peer-to-peer (as opposed to other methods) for building networks is bandwidth saving on the server side. This is achieved by

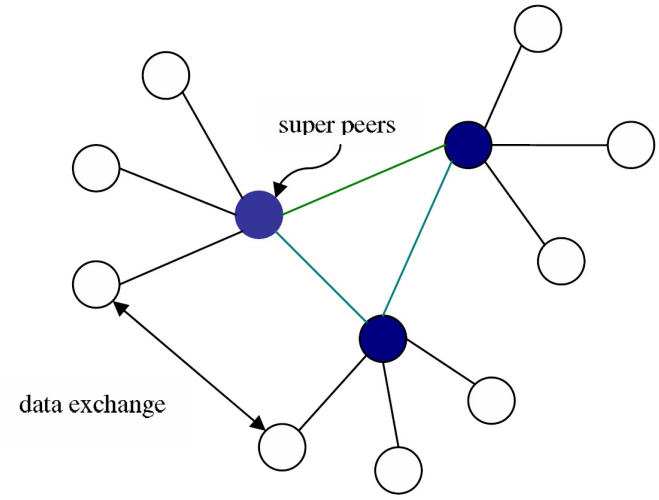


FIGURE 3.2: A decentralised hybrid peer-to-peer network segment showing distinguished supernodes

sharing the requirement for distribution bandwidth amongst the peers in the network. As each peer contributes upload bandwidth, they effectively share the task of data distribution for the entire peer-to-peer network.

3.3.2 Processing resources

Another resource that is typically shared or redistributed (by using peer-to-peer as a network infrastructure) is processing power. Diverse peers contribute resources in order to form a type of supercomputer where all nodes receive instructions and source data from a central server. These instructions are carried out and the source data is manipulated as required. The output of the process is then returned to the server for integration. Essentially each node in the network acts like a process in the supercomputer.

3.3.3 Storage

Although not as common, peer-to-peer can be used to establish distributed storage networks where peers contribute free storage space that they are not using (and not desired to be used in future). All nodes within the network are combined to form a large storage array which essentially becomes one large storage device. This can be implemented using an intelligent vastly distributed filesystem. If each node within the network only contributes a few gigabytes of free space (not uncommon with modern hardware), it is possible to build networks of a few hundred nodes that together create a multi terabyte storage system. These networks typically face a transfer speed challenge in comparison with other large storage networks due to the

relatively low throughput capability of broadband Internet connections for mass data transfer (specifically upstream bandwidth).

3.4 PERFORMANCE CHARACTERISTICS

Peer-to-peer performance is measured using a few basic metrics. The most important performance characteristic is functionality. A streaming system should be able to provide a fluid, reliable stream. For a data stream, packet delivery rate (or conversely, packet loss rate) along with packet delivery delay are the most important metrics. Within the limits of a data buffer, packets should be delivered such that the stream can be constructed slightly faster than the packet playback rate in a continuous manner for real time streaming.

Security and data errors are secondary performance measures. The correct packets should be transferred without any corruption or possible malicious packet injection (such as rogue advertising). Recovery from node failure is always a very important consideration when gauging the performance of a peer-to-peer platform. If a node goes down, this should only lead to minimal disruption (if any) to the live stream for downstream peers.

3.5 PROTOCOL CHOICES

Protocol choices for peer-to-peer networks include decisions for the communication (or message passing) between nodes within the network and for data transfer. Communication protocols will typically be based on TCP/IP or implement TCP/IP directly due to the reliability of the connection-oriented protocol.

When considering a protocol for data transfer, the nature of the application will dictate the choice of protocol. File sharing or content delivery applications will typically implement TCP/IP since delivery time is not as essential as reliable delivery. However, the excessive overhead and rate scaling functions of TCP/IP can become problematic. A streaming network will typically use UDP/IP for data packets to minimise transit time. In a streaming system, packet re-transmission is meaningless and packets are only data-oriented making UDP the ideal protocol choice.

3.6 CHALLENGES

The nature of construction of peer-to-peer networks gives rise to a number of important challenges and shortcomings. The main difficulties are experienced when network failures occur due to the lack of global control of a peer-to-peer network. Managing the flow of information between nodes can be difficult and complex. Therefore, routing of messages and data for peer-to-peer networks is a specialised field and requires careful design to prevent message looping and excessive broadcasting of updates. Any peer-to-peer system will also encounter packet loss (with effects ranging from lost protocol messages to missing video frames) and needs to have the ability to recover gracefully without affecting downstream peers.

Security is another problem encountered by peer-to-peer systems. There is often little inherent protection against malicious attacks on nodes or resource usage by unauthorised users (typically without sharing received data). New methods for dealing with security and trust issues within a peer-to-peer environment are required. Load distribution is another challenge faced by peer-to-peer networks which are prone to loss of balance with the typical heterogeneity of peers and global instability of the streaming network. Effective distribution is tied to fair resource sharing which can be controlled using incentive mechanisms for stimulating sharing of resources.

3.7 STREAMING SOLUTIONS

This section presents the main options for constructing streaming systems. A variety of solutions are included as a means to highlight the benefit gained from using peer-to-peer for multimedia streaming.

3.7.1 Media Server Farm

A media server farm makes use of a unicast approach to streaming. One or more clustered servers act as the media server to provide content over a single Internet path. The bandwidth cost of such a system is high due to the individual bandwidth required for each individual streaming client. The scale of this solution has an upper limit which depends on the capacity of the server's bandwidth to the Internet (available outgoing bandwidth). The network layout of this streaming solution is shown in Figure 3.3.

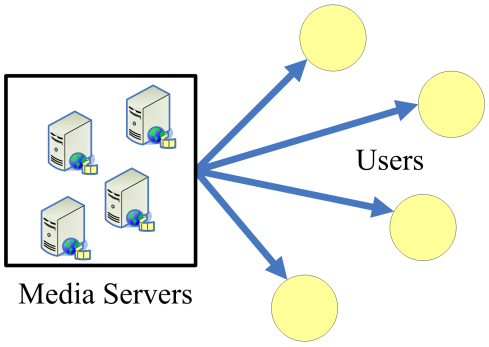


FIGURE 3.3: Media Server Farm Streaming Topology (taken from [1], p. 2)

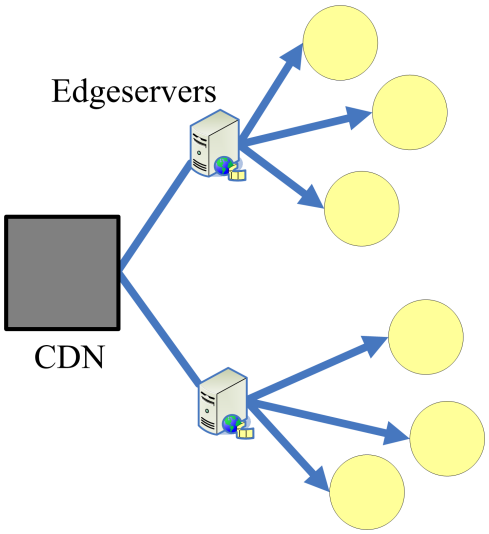


FIGURE 3.4: Content Delivery Network layout (taken from [1], p. 3)

3.7.2 Content Delivery Networks

This solution is functionally similar to the media server farm described in the previous sub-section. The differing characteristic is the layout of the system. Content delivery networks (CDN) make use of edgeservers which are directly connected within the CDN. When a user requests a stream, it would be provided by the nearest edgeserver. The delivery network is thus distributed such that content can be delivered to the user from the nearest POP.

The CDN approach offers much greater scalability due to the addition of servers with their own Internet bandwidth within the CDN. Greater redundancy is offered by the multiple points of failure while localisation improves latency. The CDN network requires a continual mirroring system from the primary server to outlying edgeservers. A typical simplified CDN layout is shown in Figure 3.4.

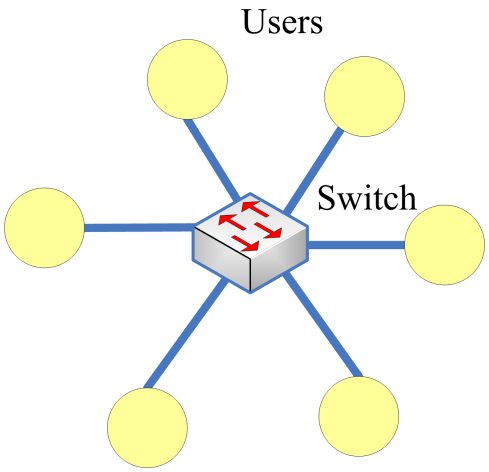


FIGURE 3.5: Hardware multicast network (taken from [1], p. 4)

3.7.3 Multicast: One stream, many addresses

Multicast allows users or clients to belong to a specific address group that receives a single stream. A copy of the stream is sent to each address that belongs to the multicast group in a way that optimises bandwidth utilisation. Multicast streaming operates by creating small sub-networks which form multicast groups. Only one copy of the stream is sent over the Internet connection to each group. The stream is then sent from a distribution point to each member of the multicast group.

Implementations of multicast vary from software based solutions to switched hardware solutions. Hardware multicast solutions are expensive as new or upgraded switching hardware is required along with complete re-configuration of the packet handling systems within the network. Figure 3.5 shows a basic hardware multicast network. The users belong to a virtual sub-network which is addressed by the multicast address. The streaming source need only contact the group using the multicast address.

3.7.4 Gridcasting

In gridcasting, each participating node becomes part of a vast, managed content distribution grid [20]. This approach facilitates the creation of a peer-to-peer built CDN where each node is a mini-server that redistributes small segments of video. This method of video distribution offers greater overall network capacity, faster delivery speed and higher reliability than traditional CDNs. The gridcasting platform maintains centralised *command-and-control* of video streams. The main problem with the gridcasting approach is that it requires a

significantly large network to function effectively (*i.e.* the network will perform relatively poorly initially while the distribution network grows).

3.7.5 Peer-to-peer live streaming

Basic peer-to-peer live streaming offers enormous scale and extremely low cost as primary advantages. However, such a network has significant drawbacks. Practical results [21] do not meet theoretical expectations due to synchronisation issues and large overheads for a complex peer-to-peer network.

Many commercial applications [22, 23] and open source [24] video streaming applications make use of this peer-to-peer live streaming approach. Experimentation [21] has shown that the peer-to-peer live streaming method is successful on a small scale (less than 100 peers) and if applied using low bit rate encoding, reduces bandwidth requirements by up to 50% when compared to a media server farm.

Challenges faced by basic tree-based peer-to-peer live streaming include:

- NAT issues for users connecting through routers and gateways
- Unbalanced upload to download ratio due to typical DSL connections being asymmetric
- Tree structure means that eventually many users will be at the bottom of the tree and only a limited subset of users will be contributing idle upload bandwidth
- Users with upload bandwidth smaller than the video stream (*i.e.* upload 256Kbps for a 384Kbps video stream) will not be able to contribute effectively
- If a node in the tree goes down, the entire downstream network is disconnected requiring a re-organisation of the tree

The theoretical basic peer-to-peer live streaming network structure is shown in Figure 3.6.

3.8 PEER-TO-PEER MULTICAST STREAMING

Networks that employ multicast streaming using a peer-to-peer approach have one of three main structures. These network structures are named based on the layout and operation of the network. The tree-based system relies on a tree like network where data is disseminated

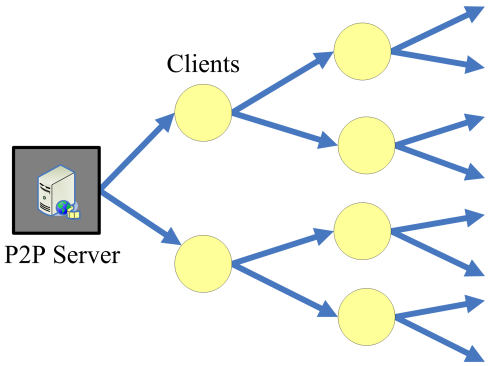


FIGURE 3.6: Peer-to-peer live streaming network (taken from [1], p. 5)

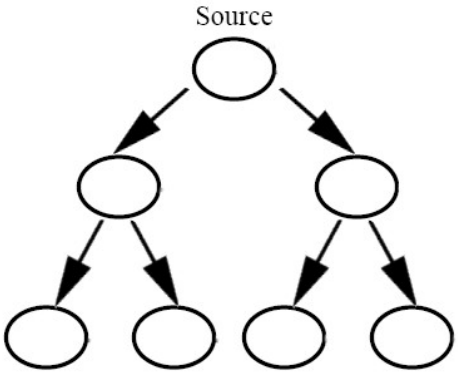


FIGURE 3.7: A tree-based multicast system

to branches and to leaves. The split streamed network makes use of multiple paths across the network by using the multipath diversity offered by a randomly organised peer-to-peer network. Mesh-based streaming relies on a peer-to-peer overlay and typically requires a server for control and management. The mesh overlay organises the network to facilitate streaming for any number of connected peers.

3.8.1 Tree based system

This streaming network is based on a single tree architecture with a top level node, interior nodes and leaf nodes as shown in Figure 3.7. This is the general peer-to-peer streaming model. The tree-based approach has two major drawbacks: as it scales, the tree becomes heavily unbalanced with the majority of the load being placed on nodes that are *higher* up and secondly, interior nodes become bandwidth bottlenecks as the overall tree grows larger (assuming these nodes are regular nodes).

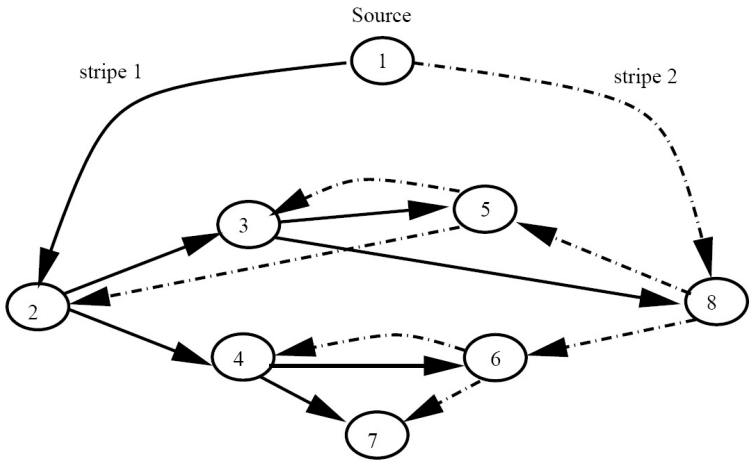


FIGURE 3.8: The splitstream approach where the original source is split into two stripes with individual multicast trees

3.8.2 Split streamed multicast streaming

An improvement to the tree-based system is proposed in [25] where the stream is split into multiple stripes which individually use separate multicast trees to distribute each stripe of the stream to downstream peers (see Figure 3.8). The ideal in this model is that most peers are interior nodes in only one tree and leaf nodes in all other trees.

This system distributes the forwarding workload amongst all peers and provides maximum benefit when a large number of co-operative peers request a single stream [26]. The advantage is that peers can choose to join a subset of the provided stripes and thus control their available bandwidth. Robustness is improved due to the fast recovery from a stripe failure. The advantages are offset by the difficulty in finding an optimal *forest*.

3.8.3 Mesh-based streaming

The notion of mesh-based streaming is based on the concept of an overlay network [27]. This overlay network can be formed in a number of ways where the simplest is based on a bootstrapping node or tracking server that facilitates construction of the overlay network. Typically, content is delivered by means of a *pull* based system where each peer in the overlay requests certain packets from its upstream peers (reporting the availability of these packets as they are received). The requests for video packets are based on a well defined packet scheduling system.

Mesh-based overlays can be represented using an organised diagram as shown in Figure 3.9.

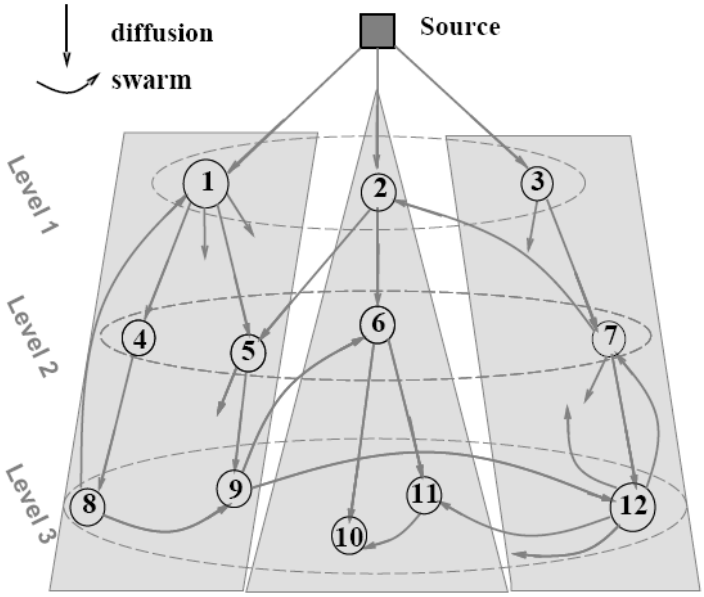


FIGURE 3.9: A typical organised view of a mesh-based overlay with 12 peers (taken from [27], p. 2)

In this model, nodes are numbered and shown as belonging to a level. This level is based on the number of hops from the source. Each additional hop results in a incremented level number. The direction of data flow is indicated (though not all connections are indicated in Figure 3.9). Since the overlay is a mesh-based network, there may be connections between peers. This results in a complex streaming structure as compared to the relatively simple triangular structure of a tree-based streaming system. The directed diffusion shown is the path of video from the source to end viewers. Swarm connections are used to construct the mesh.

The key difference with a mesh-based system is that a peer need not only connect to a peer with a higher degree or level. Data can be distributed horizontally or even in the opposite direction to the stream diffusion.

3.9 SUMMARY

Peer-to-peer networks are based on a client-client (or many-to-many) approach in contrast to the traditional client-server (one-to-many) network. The emphasis for this study on peer-to-peer networks was multimedia streaming. The different possibilities and types of peer-to-peer network are described.

This chapter also addressed the challenges that a peer-to-peer network designer faces and briefly described several peer-to-peer network designs. A number of key benefits were derived leading up to the coverage of peer-to-peer as an architecture for streaming. Different approaches to streaming (using peer-to-peer) were described such as to evaluate the suitability of these approaches for the proposed implementation.

The proposed implementation design uses elements of other peer-to-peer network architectures. The following chapter provides an overview of the related fields in the available literature.

CHAPTER FOUR
LITERATURE STUDY

"All hell's about to break loose"

BRAD BURNHAM, A VENTURE CAPITALIST WITH UNION SQUARE VENTURES IN MANHATTAN, WHICH STUDIES THE IMPACT OF NEW TECHNOLOGY ON TRADITIONAL MEDIA

4.1 INTRODUCTION

The field of video streaming is a fairly new one. The same can be said about the concept of using peer-to-peer networks as an alternative to the traditional client-server approach. These two technologies are enjoying concentrated research at present. Peer-to-peer networks have been thoroughly researched and reported on due in part to the success of file sharing networks.

In a complete survey of peer-to-peer networks, [28] aims to contrast various peer-to-peer overlay technologies. [29] highlights peer-to-peer implementations, network structures and challenges facing peer-to-peer networks. New ways of exploiting peer-to-peer (for example, using peer-to-peer as the underlying network in live video streaming [30]) have also been proposed.

Several factors which play a role in video streaming are also thoroughly investigated in the literature. These factors include streaming concerns for lossy networks and consequently providing a high QoE [31] as well as adapting the architecture of the chosen peer-to-peer overlay network for multimedia streaming [32]. This literature survey highlights the results of other related research and forms a base for further development of the topic and eventual description of the proposed implementation.

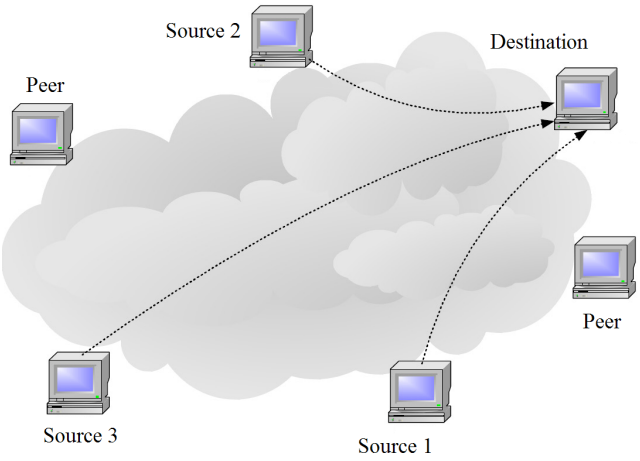


FIGURE 4.1: Multi-source peer-to-peer streaming model ([33], p. 44)

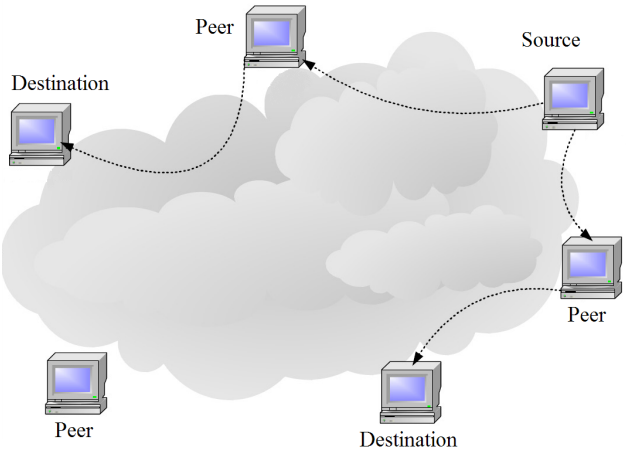


FIGURE 4.2: Single-source peer-to-peer streaming model ([33], p. 44)

4.2 MULTIMEDIA STREAMING TECHNOLOGIES AND TECHNIQUES

Multimedia streaming is driven more by the delivery method and less by the type of media being streamed. A number of streaming methods and network architectures have been developed and implemented. In a discussion on the issues that should be taken into account when considering peer-to-peer streaming [33], the following generic peer-to-peer streaming architectures are introduced: multiple-source peer-to-peer streaming (Figure 4.1) and the single-source peer-to-peer streaming model (Figure 4.2).

4.2.1 Streaming protocols

The choice of protocol is an important driver for the design of a streaming system. Datagram protocols (such as UDP [17]) use small packets (datagrams) for data. Datagrams are an efficient and simple way to carry streaming data to clients but are vulnerable to packet loss (due to the lack of built-in reliability measures) and may require additional design complexity to provide reliability. Client-side error correction or redundancy techniques can be incorporated but stream dropout would still be likely under heavy packet loss.

The Real-time Transport Protocol (RTP) [11] and Real-time Transport Control Protocol (RTCP) [34] are built on top of UDP and are specifically designed for media streaming. The Real-time Streaming Protocol (RTSP) [12] is an application-level protocol which provides control over streamed data with real time properties. RTSP defines parameters for media delivery (aided by the Session Description Protocol [14]) between the RTSP server and clients while RTP (or the proprietary RDT [35]) protocol is used as the underlying transport protocol.

The Flash Media Server video streaming system makes use of the proprietary Real Time Messaging Protocol (RTMP) for communications between the Flash player and the Flash Communication Server [36]. Microsoft streaming services (also known as NetShow services) makes use of the Microsoft Media Services (MMS) protocol to stream unicast data via TCP or UDP (initially UDP and resorting to MMS over TCP if UDP fails). MMS is designed to switch to lesser efficient HTTP streaming if all else fails.

[37] proposes an MPEG-4 standards compliant (referring to the MPEG-4 Delivery Multimedia Integration Framework) multicast system which enables transparent multicast streaming on the Internet. An application-level multicast system called DHCM (density-based hierarchical clustering multicast) is proposed in [38] and uses a peer-to-peer scheme for data transmission. Results indicate that this multicast system is effective as an efficient and robust peer-to-peer based media streaming system.

4.2.2 Streaming technologies

Streaming technologies generally focus on providing a relatively high QoS (or more accurately, QoE) under variable network conditions. Allowances are usually made for lost, damaged or out of order data sequences or packets. This section provides a brief overview of

a number of streaming performance-specific techniques that have been developed and that can be applied to improve the overall streaming success rate.

The Automatic Repeat-reQuest (ARQ) protocol is an error control method often used in data transmission. When a receiver detects errors in incoming data, a retransmission is automatically requested from the transmitter. An efficient rate-sensitive ARQ algorithm specifically for real time video streaming applications is proposed by [39] and has been shown to significantly reduce packet loss rate and improve QoE (even with a relatively small playback buffer).

Forward Error Correction (FEC) is a method of correcting data errors during transmission and does not rely on the entire volume of data to be transmitted (as with CRC correction). Data errors are corrected using previously received data blocks which have built-in redundancy for this purpose.

RTP specifies a payload format for generic FEC for encapsulated media [40] based on the XOR parity operation. A novel approach to FEC-based video streaming is presented in [41] whereby a pre-interleaving scheme is used to effectively improve end-to-end streaming performance in typical Internet and wireless networking scenarios.

A common method used to achieve a reliable QoE is Multiple Description Coding (MDC) or layered multimedia encoding [42]. This encoding method allows for multiple streams to be concurrently encoded and streamed in such a way that at the onset of congestion, the viewer is dynamically switched to the lower stream and does not suffer video dropouts. A generalized MDC coding system is shown in Figure 4.3. An example of a dual encoded stream can be found in End System Multicast (ESM) [24].

Several designs attempt to find ways to overcome the challenge of TCP's Linear Increase Multiplicative Decrease (LIMD) behaviour where transmission rate is scaled down by a large factor when packet loss is detected. LIMD/H (with history) [43] is a congestion control algorithm that uses the history of packet losses to distinguish between congestion-induced and non congestion-induced packet losses and consequently provides a gentle reaction to non congestion-induced losses to maintain minimal transfer rate variation in the multimedia stream. The reaction to congestion is quick and the algorithm delivers comparable

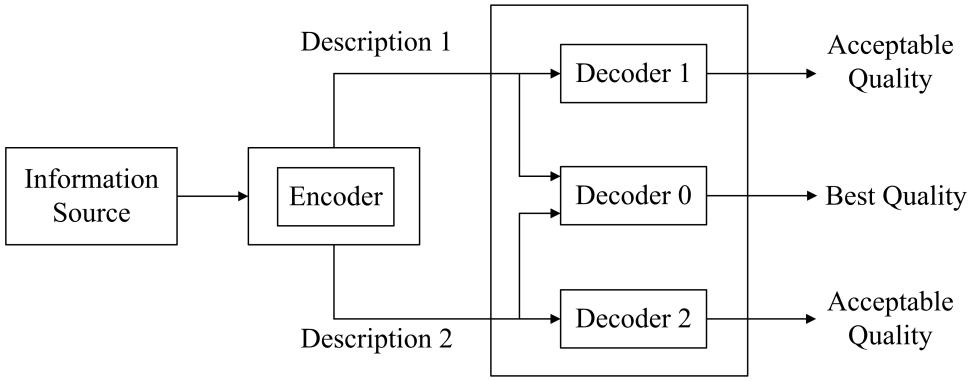


FIGURE 4.3: A general MDC coding system with two descriptions

performance to traditional LIMD making it TCP-friendly.

Interleaving is a technique used to re-distribute errors to many channel blocks such as to change burst errors into random errors over many channel blocks. This process leads to better performance in channel decoding at the receiver by breaking bursty error patterns up into reliable error trends [44].

Optimal rate control [45] for stream transmission is used to minimize both the client side buffer utilization and the transmission rate under variable packet loss. This method is a bandwidth saving scheme and works most efficiently with variable bit rate encoding. Streaming buffers are used as efficiently as possible thus preventing buffer underflow, overflow and dropouts in the transmitted stream.

4.2.3 Peer-to-peer streaming system design considerations

The following list of design considerations that should be taken into account when creating a peer-to-peer streaming system are proposed in [33]:

- Appropriate video coding scheme
  The video coding scheme must be flexible and reliable.
- Managing peer dynamicity
  Management of unpredictable peer movements in and out of the network to ensure smooth playback at all times.
- Peer heterogeneity
  Peer selection and control should account for the variety of peers within the network.

- Efficient overlay network construction
  Supporting networks should not hinder the peer-to-peer network expansion and should support peer-based streaming.

- Selection of the best peers
  Optimal peers should be relocated higher up in the stream hierarchy to minimise end-to-end delay and global overheads.

- Monitoring of network conditions
  A knowledge of network performance in varying conditions can allow for better utilisation of available resources as they become available or as they change when peers join and leave the network.

- Incentives for participating peers
  Incentive mechanisms result in a willingness to share resources which in effect increases the service offering level to all end users.

4.3 MULTIMEDIA STREAMING APPLICATIONS

Various commercial, open source and academic streaming applications have been designed and implemented. Typically, these applications enable the streaming of media over the Internet as well as the offering of Internet television (also known as IPTV) broadcasting using multimedia streaming. Implementation scenarios vary from academic proof-of-concept to full blown commercial deployment.

4.3.1 Streaming container formats

Container formats specify how multimedia is packaged, stored and transmitted. Containers control how video and audio are interleaved and indexed for file storage, conversion, playback and streaming. Different container formats find application in different areas. Simple container formats can store different types of audio data whereas more advanced containers support audio, video, subtitles, chapters and meta-data along with the synchronization information which is used to play the various included streams concurrently.

The more advanced container formats are designed to fully support encoding and packaging for streaming. Video and audio data are interleaved in such a way that playback can begin

from any point within the stream. Interleaving the various media tracks in this way also provides simpler recovery from error as the stream can be decoded from almost any given point.

The primary multimedia containers for multimedia are the following:

- AVI (based on RIFF)
- MOV (Apple QuickTime container)
- MPEG-2 TS (Transport Stream, used for digital broadcasting)
- MP4 (MPEG-4 container)
- OGG (open source Xiph.org container)
- ASF (Microsoft container for WMA and WMV)
- RealMedia (standard container for RealAudio and RealVideo)
- Matroska (open container standard)
- 3gp (used by mobile phones)

The most commonly used containers for multimedia streaming are MP4, OGG, ASF, Matroska and RealMedia. Other pure streaming containers have been developed including NSV [46] (Nullsoft Streaming Video) and the Flash Video format (FLV).

4.3.2 Multimedia streaming codecs

Multimedia codecs are the compression algorithms used to compress incoming data in preparation for storage or distribution over a relatively low bandwidth transmission medium. Codecs are separated into two distinct categories: audio and video. A wide variety of audio and video codecs are available for use. However, these codecs are not all compatible with each other resulting in specific configurations for encoding. Many codecs are designed with streaming in mind and can be used to encode incoming video/audio data directly for streaming in real time. Popular codecs are discussed below (with emphasis on streaming).

Experimental results have shown that H.264 achieves a 50% average coding gain over MPEG-2, 47% (4dB PSNR gain) over H.263 baseline, 24% over H.263 high profile

encoding [47] and 3.7dB PSNR gain over H.263+ [48].

Application of H.264 video coding for video distribution over best effort IP networks has been shown to be highly successful due to efficiency techniques inherent to the encoding design of H.264. In particular, the concepts of FMO (flexible macroblock ordering), NALUs (NAL units) on the NAL (network adaptation layer), data partitioning on the VCL (video coding layer) and parameter sets significantly improve H.264 performance in the IP environment [49].

4.3.3 Open source codecs and containers

The nature of the encoding process with its respective formats and specifications dealing with various types of media lends itself to licensing regulations. A number of popular formats require costly licenses for full usage. For this reason, open source codecs are mentioned separately in order to group the number of low-cost options available to the average streaming system developer or media producer. Streaming systems are typically heavily dependent on the chosen codec and encoding format or container.

The list of free-to-use containers includes Ogg (developed by Xiph.org), Matroska (well supported and feature rich) as well as NUT (a new patent-free container format created by the developers of MPlayer [50] and FFmpeg [51]) which is designed to be simple, extensible, compact and error resistant without placing restrictions on supported formats.
The most commonly used and regularly updated open source video codecs include:

- x264
  H.264/AVC (MPEG-4 Part 10) implementation created by the developers of the VideoLAN [52] server/client platform.

- XviD
  MPEG-4 Part 2 codec which is format compatible with DivX.

- Theora
  Implementation developed by the Xiph.org foundation. Based on the freely available VP3 codec (originally a proprietary codec created by and belonging to On2 Technologies).

- Huffyuv
  Very fast lossless codec based on Huffman encoding of uncompressed *YCbCr* video.

- FFmpeg codec library
  The libavcodec library provides Snow, MPEG-1, MPEG-2, MPEG-4, H.264, WMV2, M-JPEG, Indeo and others.

4.4 PEER-TO-PEER CONCEPTS

Research into ways of implementing peer-to-peer networks has resulted in the development of a number of specific network architectures and operational algorithms. These technologies form the basis of most efficient peer-to-peer networks and offer several advantages over traditional network architectures. Overlay networks are the predominant area of focus within the scope of peer-to-peer research. This section includes an overview of relevant research and introduces peer-to-peer concepts and networks.

4.4.1 Overlay networks

Overlay networks are networks that are constructed on top of existing networks and use the underlying network structure as a base architecture. Message passing and control as well as routing and addressing are carried out using information within the overlay network. Packets are routed within the overlay network by lower layer protocol software which manages routing and communications on the underlying network (for example: using IP addressing and routing on an underlying IP network). Essentially, all peer-to-peer networks are overlay networks.

Benefits of overlay networks include reliable, efficient content delivery as well as efficient routing [53] and improved QoS guarantees [54] (for better multimedia streaming), guaranteed data retrieval, self organization, load balancing and provable lookup-time horizons (typically $O(log(n))$ for n nodes) [55].

4.4.2 Distributed hash table

Distributed hash tables (DHTs) are decentralized distributed systems that use keys to efficiently route messages within a network of participating nodes. DHTs are designed to scale to large numbers of nodes and seamlessly handle node arrivals and failures. Due to the nature of DHT networks, complex services such as peer-to-peer resource sharing and

distributed multicast systems can be built.

In a study of SplitStream [25] (which is based on the Pastry [56] overlay network), various implementation problems are highlighted of which high rates of peer transiency is the most difficult obstacle [57]. It is thus suggested that deployment of multicast and other peer-to-peer services over a DHT network can be vulnerable to a number of challenges and should be designed with aspects of transiency and dynamic resource availability in mind.

The most prominent DHT implementations are CAN, Chord, Pastry and Tapestry. DHT technology has also found application in BitTorrent [58] and the Coral CDN [59]. The following properties characterise DHTs:

- **Decentralization**
  There is no need for a single central server or server network.
- **Scalability**
  Efficiency can be maintained even when the network extends to millions of nodes.
- **Fault tolerance**
  The network should be relatively reliable when nodes continually join, leave or fail.

DHT networks achieve the above objectives by coordinating with only a few other nodes so that only a small amount of configuration changes need to be made when the profile of connected nodes changes. The overlay network operates by generating combinations or pairs of keys (produced by using SHA-1 for example) with blocks of data. Keys are used to index data blocks in the network. The management of these pairs and their creation is dependent on the particular DHT network implemented.

4.4.3 Incentive mechanisms for fair resource sharing

In peer-to-peer networks, it is imperative that peers actually contribute to the network. Proper operation and good QoE is reliant on peers contributing in an active way. Mechanisms need to be put in place in any peer-to-peer network (especially multimedia streaming networks) that reward good resource contributors with better service. Incentive mechanisms for fair resource sharing have been designed [60] where a middleware layer is implemented that brings together aggregation, semantic group membership and tracking.

Fair resource sharing schemes improve the level of service to all peers within the network and as such cannot be ignored. Fairness metrics can be used to measure whether or not peers are receiving a fair share of the stream. Two fairness metrics that can be used include Jain's fairness index [61] (Equation 4.1 is used to determine the level of fairness offered by a sharing scheme where fairness varies between 1/n for the worst case and optimally 1.0)

$$\text{fairness} = \frac{(\sum x_\text{i})^2}{\text{n} * \sum x_\text{i}^2}$$

where:

$$\begin{aligned} x_\text{i} &= \frac{\text{measured value}}{\text{optimal value}} \\ \text{n} &= \text{number of measurements} \\ \text{i} &= 1..\text{n} \end{aligned} \tag{4.1}$$

and max-min fairness where small flows receive what they request and larger flows share the remaining capacity equally. Bandwidth is equally allocated until one flow is satisfied after which the bandwidth is equally increased among the remainder until all flows are satisfied or the bandwidth is exhausted.

4.5 PEER-TO-PEER NETWORKS

This section documents a basic peer-to-peer reference architecture and subsequently introduces the most commonly deployed, developed and researched peer-to-peer networks.

4.5.1 Peer-to-peer reference architecture

Peer-to-peer applications are typically designed with only a specific architecture in mind. [62] proposes the peer-to-peer reference architecture shown in Figure 4.4 which is based on three key components: common runtime, security and core servents. The common runtime specifies the messaging infrastructure for establishing connections and for inter-peer communications. The security component specifies how authentication, authorisation and secure message transfer are implemented. The core servents specify peer-to-peer functionality while the application servents are responsible for application logic (video streaming is an example). This reference architecture is further expanded in [63] including examples of a number of peer-to-peer systems and applications specified within the complex architecture.

A reference architecture for peer-to-peer overlay networks along with conceptual models is

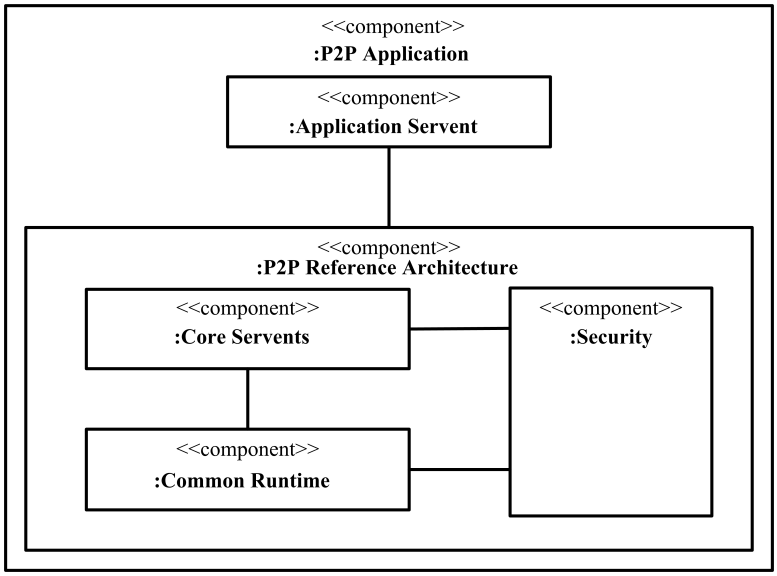


FIGURE 4.4: Peer-to-peer reference architecture ([62], p. 4)

presented in [64]. A simple layered architecture is proposed. This layered model comprises (in order) an application layer, peer-to-peer storage, peer-to-peer basic and network (TCP/IP) layers. This basic model allows separate specification of the overlay network (peer-to-peer basic) and related operations from the peer-to-peer application (peer-to-peer storage). The underlying network is clearly separated from the overlay and peer-to-peer networks.

4.5.2 Chord

Chord is a DHT based robust peer-to-peer system which is completely decentralized [65]. The Chord DHT primitive allows for retrieval of data using only *log(N)* messages where *N* is the number of nodes in the system. This lookup system is designed to be fully robust even with frequent node failures or with a highly dynamic network of nodes. The strength of Chord lies in the way that so-called predecessor and successor nodes are stored allowing the Chord network to adapt efficiently as nodes join and leave the system. Queries can be answered even while the system is continually changing. This is achieved by storing keys using redundancy within nodes.

4.5.3 Pastry

Pastry is a typical overlay network much like Chord. The distinguishing characteristic is the routing overlay concept built on top of the DHT network. This network design allows for the use of traditional routing metrics (for example, ICMP packets) for more efficient routing and elimination of packet flooding (a characteristic of other overlay networks) [56].

Applications based on Pastry include PAST (a distributed file system) and SCRIBE (a decentralized publishing/subscription system which uses Pastry for underlying route management and host lookup). Multimedia peercasting applications based on Pastry include GhostShare and SplitStream. [66] exposes weaknesses in the form of simulation results for a heterogeneous Pastry overlay network. Weak nodes are shown to have a significant effect on message delay for a small increase in hop count. This effect needs to be taken into account when designing a peer-to-peer network that is built using Pastry.

4.5.4 Tapestry

The second generation of peer-to-peer networks was designed to overcome traditional limitations such as centralisation, single points of failure, limited scalability and poor redundancy. Tapestry [67] is a second generation peer-to-peer network (along with Chord, Pastry and CAN). Highly optimized routing and low message latency are the key design areas and distinguishing features of Tapestry within the familiar group of peer-to-peer overlay networks. In addition, Tapestry provides object distribution determination and multicasting features (within the overlay network) to applications.

The main benefits of implementing Tapestry as the peer-to-peer overlay network include network stability, security (using trusted PKI for node ID assignment), redundant dual-path routing [68] and the use of MAC addresses to maintain overlay traffic integrity. Tapestry offers resilience against server and network failures by using multiple servers with redundant paths and intelligent routing.

4.5.5 Content addressable network

Content addressable networks are large networks with highly scalable indexing systems. These indexing systems perform well for any large scale network and thus are highly efficient for a peer-to-peer network. Much like its counterparts, CANs use {key,value} pairs (stored in a large hash table) for addressing and data location. Each CAN node stores a segment or *zone* of the complete hash table and holds information about a small number of co-located zones in the table. Requests for a key are routed through intermediate CAN nodes towards the CAN node with the zone that contains the desired key. A reference CAN design in [69] is designed to be completely distributed (requires no centralized control, co-ordination or

configuration), scalable (nodes maintain a small list of adjacent nodes independent of the number of total nodes in the system) and fault tolerant (nodes are able to route around failures).

When a node opts to leave the CAN, it hands over its zone and associated {key,value} database to a nearby neighbour with sufficient capacity to merge the two zones. The CAN handles node failure by implementing an immediate takeover algorithm such that surrounding nodes independently initiate a takeover timer and upon expiry, the replacement node communicates its own zone volume to all of the failed node's neighbours. This process is initiated when neighbour nodes no longer receive regular periodic updates from the failed node. Replacement nodes are efficiently chosen such that they are reliable and have sufficient zone space to take over the failed node's zone.

4.5.6 Kademlia

Much like other DHT based peer-to-peer networks, Kademlia makes use of a node ID based system where nodes store {key, value} pairs based on the closeness (within the ID space) of nodes. A node ID routing system is implemented which allows any node to locate servers near a destination key. The creators of Kademlia claim that it is the first peer-to-peer network to exploit the fact that node failures are inversely proportional to uptime [70]. A concurrency parameter α allows nodes to trade a constant factor in bandwidth for asynchronous lowest latency hop selection and delay free fault recovery.

In the development of a general framework for scalability and performance analysis of DHT routing systems [71], it was found that unlike other DHT routing algorithms, Kademlia proves to be scalable in a large network where other algorithms experienced a loss of routing capability when the probability of random node failure was non-zero.

4.6 PEERCASTING METHODS

Several different network architectures can be implemented to enable peercasting. These architectures differ by their implementation but tend to offer the same overall functionality and benefits (primarily bandwidth load sharing over a distributed peer-to-peer network). This section describes these peercasting architectures.

4.6.1 Stream relays

Peercasting is enabled by peers that relay the received stream to other peers. The peer-to-peer network and/or server(s) provide and connect additional peers and also facilitate a peer's initial connection to the stream. This method is prone to poor quality of service when stream relays are broken by disconnecting peers or stream cancellation when relay peers decide to switch to another stream.

4.6.2 Minute swarming

Minute swarming is an intermediary type of peercasting whereby a live stream is broken up into one minute portions. These portions are distributed as single entities or chunks of the stream. These minute chunks are distributed independently using a peer-to-peer approach similar to Bittorrent. This method suffers from excessive overhead since the streaming network needs to be effectively re-created for each streamed chunk.

4.6.3 Stream striping

Striping is a technology that makes use of multiple sub-streams. The live multimedia stream is split up or striped into these sub-streams (similar to RAID striping). Timing information and FEC codes are sent with the sub-streams and are used when forming the original stream from received sub-streams. Typically, the original stream can be formed using all but one (*i.e.* n-1 sub-streams out of n total sub-streams) of the sub-streams. Rateless erasure codes (Digital Fountain codes [72]) can be used to efficiently create and combine sub-streams.

4.6.4 Resumable relay streaming

This peercasting concept allows peers to connect to a new relay and resume the received stream from the point where they left off at a previously lost or disconnected relay. Typically, relays in this architecture make use of a back buffer to allow new clients to resume streaming at any point within the buffer. This method has several serious drawbacks including the requirement for a significantly large back buffer. Another problem is that the network will need to be relatively large to offer acceptable redundancy. Real time streaming is also difficult or impossible using this method due to the use of buffering and maintenance of redundancy information.

4.6.5 Distributed web caching and proxying

Although not strictly peercasting, web caches and proxies specifically designed to cater for streaming are often included as peercasting architectures. Peers connect to the peer-to-peer network and together form a large wide area storage network where they typically (often without realising) store parts of various video content. They need not necessarily store complete multimedia files. These peers then send these parts that they store to connected peers (on request). The distribution of video through this method is often incorrectly referred to as peercasting. The most popular implementation of this approach is Veoh [73].

4.7 PEERCASTING APPLICATIONS

A number of peer-to-peer multimedia streaming implementations have been developed. The focus of this work is to design and develop a similar system which aims to be simple, implementable and provides solutions to the problems experienced by others. The first step toward achieving this goal is carry out a thorough investigation of the existing implementations in the literature.

The study includes open source implementations as well as a closed source implementation included purely for comparison. The details of each implementation are included below. Each application is described in a slightly different way by highlighting different elements. These different aspects highlight criteria that allow for specification of a suitable peercasting platform. The proposed implementation is intended to provide equivalent functionality to that exhibited by other peercasting implementations as a means to demonstrate how peer-to-peer can be effective in supporting video streaming.

4.7.1 PeerCast

PeerCast is an open source peercasting application used for multicast streaming of audio (Ogg Vorbis, MP3, WMA) and/or video (Ogg Theora, Nullsoft Video, WMV) over the Internet [74]. This application uses a stream relay technique where users relay the received stream to other connected users. Users may choose how many relays they allow connections to based on how much of their outgoing bandwidth they are willing to contribute.

The main problem with using PeerCast is that when a relay is lost, all downstream

peers will need to reconnect to another relay thus subjecting the user to skips or repeated sections as the user will not necessarily pick up the stream at the same point. Implementation is problematic due to limited peer upload bandwidth as well as overheads caused by the loss of subtrees when a node goes down.

A preliminary design for a file dissemination system also named PeerCast is presented in [75] along with simulation results based on a two week testing period. The focus in this paper is on fast dissemination of new files and is thus not a proper peercasting application. However, the results do indicate some design challenges and experiences. The most important results contrast peer-to-peer network performance based on a single reliable source with that of a system with multiple initial sources as well as the behaviour of the network when the peer-to-peer network reaches critical mass.

4.7.2 Alluvium

Most streaming servers require specific software to provide streaming services. Alluvium differs in that it provides a multimedia stream using only a web server on the server side. Alluvium plays files in the Alluvium playlist format which is based on RSS 1.0 [76]. This playlist file specifies the remote location of files containing content as well as timing information. Files are downloaded using the open content network (OCN) which utilises swarming download mechanisms [77] such as the Partial File Sharing Protocol (PFSP) and Tree Hash EXchange (THEX) format. The client software first checks the OCN gateway for the specified URL and if not found, the gateway fetches the necessary information from the URL and then caches it. The information stored by the gateway contains information needed to swarm download the file.

The information obtained by the client from the OCN gateway contains a list of addresses for other clients who are also downloading or have recently requested the stream (or file). Clients download multiple parts of the stream simultaneously from each other on the peer-to-peer network. When a certain part of the stream is unavailable from other clients, a client will fetch it from the original source URL and then share that specific part with the other clients, minimizing the load on the original server which stores the content files. The majority of data transfer happens between connected peers. Additionally, priority for downloading is given to chunks earlier in the file/stream. This means that while the stream does generally transfer out of order, it is sequential enough for file playback to happen almost immediately upon loading

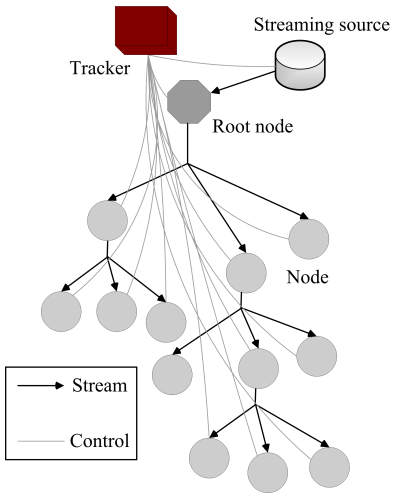


FIGURE 4.5: FreeCast Network Overview

of the content stream.

The ACTLab TV project [78] aims to provide peer-to-peer based TV broadcasting and streaming services on the Internet. The main philosophy of the ACTLab TV project is to merge open source software with copyleft media. This project uses Alluvium as a peer-to-peer base for peercasting.

4.7.3 FreeCast

FreeCast is a free peercasting application used for broadcasting audio (Ogg Vorbis) or video (Ogg Theora) to a large number of users using a single DSL connection. The main benefit of FreeCast is that the client software does not require any configuration to work, supports NAT traversal for connected nodes and is based on Java (compatible with most operating systems).

FreeCast uses a collaborative transport system whereby each user can provide stream information to three other listeners [79]. Nodes connect to a tracker which maintains the network using HTTP requests from nodes. The root node creates the stream from a multimedia source (such as an Ogg file). A FreeCast network has a hierarchical tree structure where the root node has order 0 and nodes receiving data from the root node have order 1 and so on. An order management system prevents nodes from connecting to another nodes with lower node order. TCP is used for communication between nodes. Figure 4.5 shows an overview of the FreeCast network.

4.7.4 P2PCast

P2PCast is a decentralized, scalable, fault-tolerant self-organising system designed to support streaming of content to thousands of nodes from a relatively low bandwidth network [80]. Users contribute their available bandwidth to this splitstreamed system. A novel approach is used to manage the stripes as a forest of multicast trees.

4.7.5 Joost

Joost is a commercial Internet based peer-to-peer video broadcasting system [81]. As Joost is developed by Joltid (the company that developed Skype and KaZaa), it is assumed that a hybrid peer-to-peer architecture with supernodes is implemented. Video files are also served from dedicated video servers owned by Joltid. The Joost video streaming platform aims to provide an alternative to traditional TV broadcasting and implements the CoreAVC H.264 video codec for improved video quality.

4.7.6 Babelgum

Babelgum is a direct competitor to Joost and is developed by Babel networks [82]. The streaming platform is based on peer-to-peer video streaming and is in beta testing. Babelgum implements an H.264 codec for high quality streaming at a relatively low bit rate (640Kbits/sec).

4.7.7 Veoh

Veoh claims to be a new implementation of peercasting which aims to provide an alternative to traditional television broadcasting. The main benefit is that anyone can broadcast video to consumers at no charge.

The creators of Veoh promote it as a method of peercasting. This is not strictly true since the method of peer-to-peer used is more of a wide area storage network or cache. This distributed cache is used to deliver content using peer-to-peer distribution. The Veoh system does not cater for real time streaming but instead streams stored video to connected peers using a multicast approach. This approach is based on the distributed nature of the stored video and makes Veoh a peer-to-peer video-on-demand (VOD) network.

Content providers merely upload their video to the network where it is checked and stored for future streaming to clients when requested. The primary benefit of this is that providers need not stay online continually to provide the video stream. Veoh's widespread content delivery network is similar to other wide area peer-to-peer content distribution networks: Coral [59] and the Dijjer peer-to-peer web cache [83]. Veoh is positioned to be a low OPEX cost alternative to YouTube and strives to offer a similar service.

4.7.8 Octoshape

Octoshape is a fully commercial peercasting implementation that makes use of peer-to-peer grid distributed bandwidth for video and audio streaming. Octoshape is typically used for live streaming and can be used to stream both audio and video (typical formats as with Freecast above). For more than 300 nodes, Octoshape results in a 97% bandwidth saving compared to traditional server farms [1]. The Octoshape peercasting system makes use of stream relays.

4.7.9 GnuStream

GnuStream is a prototype receiver-driven peer-to-peer media streaming system [84] built on top of *Gnutella*. GnuStream integrates dynamic peer location and streaming capacity aggregation and thus is highly efficient at load distribution and recovering from changes in the structure of the source network. A dedicated buffer management mechanism is employed to handle network heterogeneity and the variation in multiple sources (peers).
The design of GnuStream encompasses three layers (as indicated in Figure 4.6):

1. *Network abstraction layer (NAL)* - provides services such as locating, routing and retrieving to the SCL.

2. *Streaming control layer (SCL)* - the core layer responsible for the allocation of whole media to different sources and for synchronising the co-operation between peers.

3. *Media player layer (MPL)* - a built-in media player with delicate buffer control for resistance to stream failure.

Before a streaming session begins, the receiver uses Equation 4.2 to determine the aggregated streaming bandwidth which is used to compute how many upstream peers are needed. To ensure full quality media streaming, the receiver can opt to wait until sufficient peers (based on their contributed upload bandwidth) are available to make up the required available media

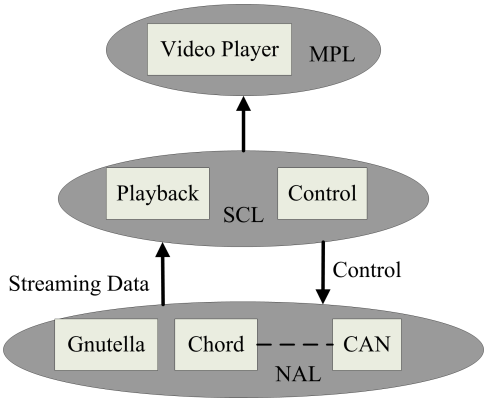


FIGURE 4.6: Layered GnuStream architecture ([84], p. 2)

bandwidth.

$$\text{AMB} = \frac{\text{f}_{size} * \text{BPP} * \text{YUV}_{sampling} * \text{PR}}{\text{CR} * \text{BPB}}$$

where

AMB: Arrival Media Bandwidth

f_{size} : Frame Size

$\text{YUV}_{sampling}$: YUV sampling rate

BPP: Bits Per Pixel (4.2)

PR: Playback Rate

CR: Compression Rate

BPB: Bits Per Byte

To achieve continuous and jitter-free media playback, the receiver implements buffering before and during playback. Initial buffering delay is determined using Equation 4.3:

$$\frac{\text{delay}_i}{\text{frames}_{tot}} = \begin{cases} 0 & \text{if AFR } \geq \text{PFR} \\ \frac{1}{\text{AFR}} - \frac{1}{\text{PFR}} & otherwise \end{cases}$$

where

delay_i : Initial Delay (4.3)

frames_{tot} : Total number of Frames

AFR: Arrival Frame Rate

PFR: Playback Frame Rate

4.7.10 GhostShare

GhostShare is a peer-to-peer platform designed to support video distribution based on the Pastry substrate [85]. Key features of GhostShare are secrecy and anonymity of operation. GhostShare achieves privacy protection using a probabilistic publishing mechanism. Another important characteristic of GhostShare is load balancing and good multipath performance. GhostShare offers anonymous video distribution over a peer-to-peer network but focuses more on privacy protection than reliable peer-to-peer streaming.

4.7.11 ESM: End System Multicast

ESM is a system designed to enable anyone to broadcast high quality multimedia to a large number of users. The system is based on an overlay tree peer-to-peer distribution network. Network management and continuous re-organisation results in high efficiency for content distribution [24].

Another useful aspect of this system is the ability to broadcast multiple video streams. This is typically used for supporting both low and high bandwidth peers at the same time. All nodes relay both streams subject to bandwidth availability. Initially, all nodes will view the higher quality stream until the ESM software detects a significantly high packet loss rate at which point the stream will be switched to the lower quality version. In this broadcast profile, the audio and lower quality video streams receive highest priority. This method of dual streaming is a crude approach to scalable quality streaming and is a first step to layered video coding and other related video scaling techniques.

The dual stream method is shown in Figure 4.7 where the San Jose viewer is unable to receive the full quality stream (the Boston viewer has no such problem).

4.7.12 OSN: Overlay Subscription Network

A framework for live Internet TV broadcast using a peer-to-peer overlay network is proposed in [86]. This is an example of a subscription overlay network (SON) where viewers subscribe to and permanently contribute resources (even when idle) as described in [87]. The source server maintains a topology graph (of which Figure 4.8 is an example) of all connections in the overlay network (including connections between subscribers). This allows the server to optimise the multicast process used for video distribution and make better use of idle nodes.

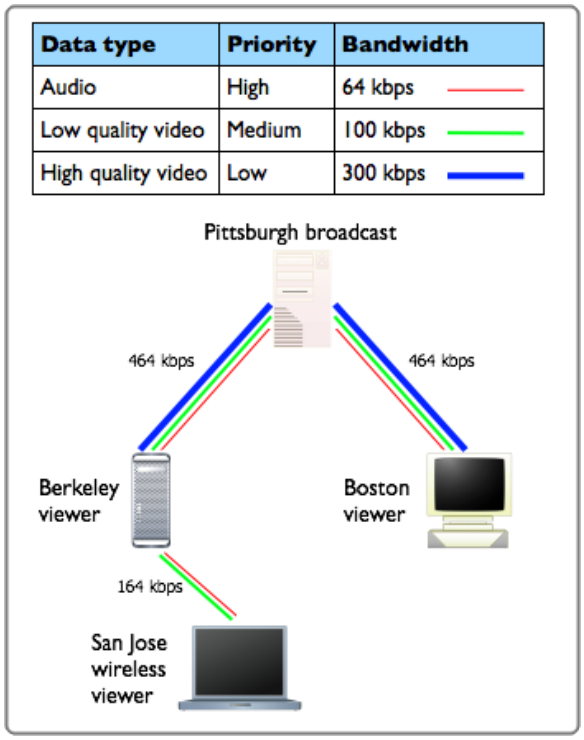


FIGURE 4.7: The ESM dual quality streaming approach [24]

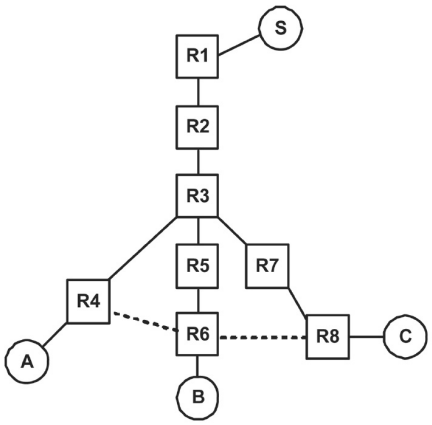


FIGURE 4.8: An example OSN topology graph (taken from [86], p. 2)

The overlay multicast technique used by OSN is unique for the following reasons:

1. Distribution is based on the topology graph and takes advantage of the knowledge of the network layout

2. The overlay scheme is able to find and use appropriate idle nodes to better distribute network traffic

3. A node can be used in numerous multicast trees for data forwarding to improve overall stream performance

This OSN framework has shown excellent performance as a video streaming tool and paves the way for an effective Internet based TV broadcasting subscription service.

4.8 PERFORMANCE OBSERVATIONS

Performance measurements and results obtained from the literature indicate that none of the peercasting implementations operate without difficulty. This is most likely due to the challenging heterogeneous nature of a peer-to-peer network. A high peer churn rate tends to impede good performance due to the ever changing availability of sources for video. Thus, performance is typically influenced by two factors: how long a peer stays within the chosen network and how much this peer will contribute in terms of resources.

Peercasting performance metrics include the following:

- Ability to maintain the stream for both a low and high peer count network based on number of stream dropouts
- Encoding and delivery of video data packets in real time (packet delivery rate)
- Buffering time should be minimal when requesting the stream
- Source switching speed on node failure
- Minimum and optimal number (or ratio) of sources for smooth streaming

These metrics are not readily available for each and every peercasting implementation. This (coupled with the fact that each network design is relatively complex) has the end result that a direct comparison is difficult or impossible. The complexity of simulation (due to the required network parameters such as dynamicity, scale *etc*) further hinders the derivation of measurable and repeatable results. Typically, only a small subset of results can be derived during proof-of-concept testing.

4.9 SUMMARY OF SHORTCOMINGS

While all of the above-mentioned streaming and peercasting systems are useful and effective for large scale streaming in general, it is important to highlight shortcomings or potential

pitfalls that will be faced when deploying networks based on these technologies. The nature of a peer-to-peer streaming infrastructure implies that it would be advantageous for the network software to be open source (due to its complexity) for lower total cost and for adaptability. Therefore, the fact that Octoshape is a closed source solution tends to hinder development and adoption to some degree.

Java is an excellent application environment for achieving multiple platform compatibility. However, the virtualised nature of Java makes it bulky and comparatively slow. Freecast (an otherwise excellent peercasting platform) is implemented in Java. Many (if not all) peercasting applications are typically married to one codec solution. This is not desirable since not all codecs are created equal and it is possible that various different codecs can show better performance relative to the nature of the video being encoded.

Veoh has (by design) a major shortcoming in the fact that this network is unable to provide live streaming. By definition, peercasting implies broadcasting which is typically live. The Veoh network requires significant time to pre-load or cache (by peer-to-peer distribution) source video for general purpose streaming.

Table 4.1 provides a brief comparison of the different comparison with respect to several important criteria.

TABLE 4.1: A basic comparison of peer-to-peer streaming applications

Applications	**Open source**	**Platform Independent**	**Pure**	**Hybrid**	**P2P Overlay**	**Codec Selection**
PeerCast	Y	Y	X		N	Y
FreeCast	Y	Y		X	N	Y
Alluvium	Y	Y		X	Y	Y
Joost	N	N		X	N	N
Babelgum	N	N		X	N	N
Veoh	N	N		X	N	N
Octoshape	N	Y	X		N	N
GnuStream	Y	Y	X		Y	Y

4.10 EXCEPTIONAL CHARACTERISTICS

The part request model utilised by Alluvium is effective in dealing with randomness in the delivery network. Support for out-of-order transfer of parts is an essential function for any real-world peercasting application.

The network model and structure of FreeCast will serve as a basic early reference for the proposed implementation. This is due to the simplicity and adaptability of the design. The combination of the FreeCast network model and the Bittorrent-based model has many advantages for the design of the proposed implementation (these advantages are discussed in a later chapter).

FreeCast (and Octoshape) claim to offer streaming using a variety of codecs. This is notable as the deployment environment need not be attached to a single codec. The layered encoding model used by ESM allows for additional support of very low capability viewers. This is beneficial in any network. The above characteristics are all exceptional and serve as excellent design pointers for the proposed implementation.

4.11 SUMMARY

This chapter provided a literature survey of the various aspects and concepts in the paradigms of video streaming, peer-to-peer and peercasting. The work of others was summarised and placed in context with that of fellow researchers. Several comparisons were drawn based on service level offerings and support.

It was found that there has been extensive research carried out in the fields of video compression, encoding, peer-to-peer networking and peer-to-peer based video streaming. Most notably, research on overlay networks and the provision of streaming over these networks has enjoyed a great deal of attention in recent years. The comparisons between several peercasting applications resulted in a well defined set of required performance parameters and desired characteristics.

These parameters will be used in the design and development process for the proposed implementation. The following chapter initiates the description of the proposed implementation by introducing the peer-to-peer video streaming model.

CHAPTER FIVE
THE PEER-TO-PEER VIDEO STREAMING MODEL

"At any point in time there are in the region of six million users logged on to peer-to-peer networks"

CACHELOGIC RESEARCH, JULY 2003

5.1 INTRODUCTION

This chapter introduces the proposed implementation which is the main focus of this work. The high-level design for the implementation is provided and is based on the selection of relevant technologies in the fields of video encoding/streaming and peer-to-peer networks. The complete technical design for the prototype peer-to-peer based video streaming system is provided in the next chapter. The following sections briefly introduce and describe the different elements in the proposed implementation.

5.2 OVERVIEW OF MODEL

The peer-to-peer streaming model in this implementation comprises two primary servers (separated by function but can be a single server). These servers play an important role in video streaming but the majority of the distribution work load is carried out by the peers themselves. Implementation of servers implies that the design is based on a hybrid peer-to-peer architecture. This approach is similar to Bittorrent and PeerCast where tracking servers are implemented.

Figure 5.1 shows a basic overview of the network model. The different components are shown along with their interactions. A sample multiple hop path for video data is shown on the model.

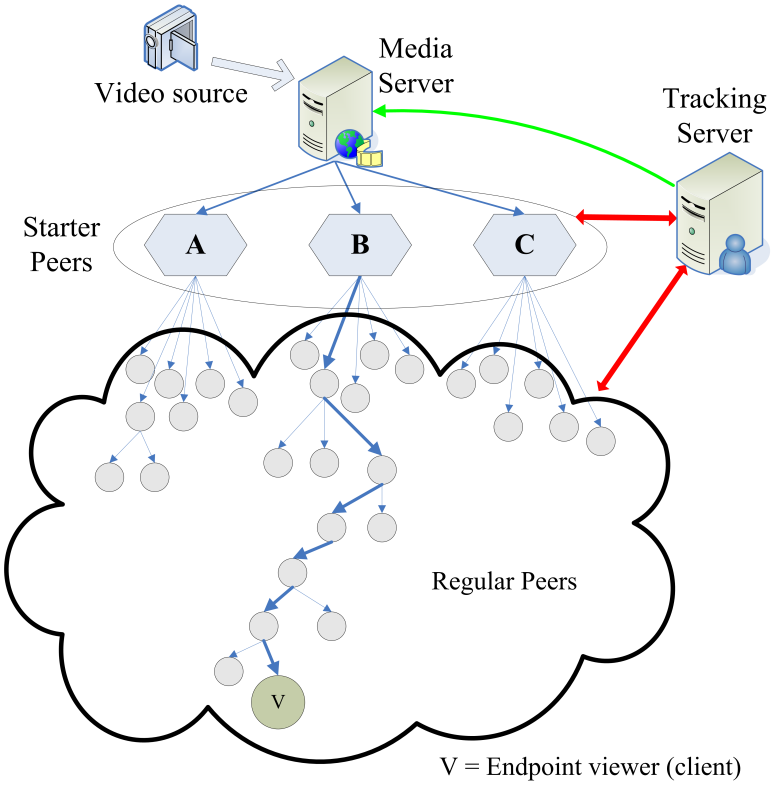


FIGURE 5.1: The network model indicating the components that make up the real-time peer-to-peer video streaming network

5.3 OPERATIONAL DESCRIPTION

This section describes the basic operation of the entire network. Details for each component are provided in the following section.

The main function of the network is to deliver video content to viewers or clients in real-time. The drivers for development using peer-to-peer are scalability as well as the removal of barriers to entry for potential peercasters. The video source supplies video data to the network via the entry point. A media server facilitates the transfer of video from the source to the entry point peers. These peers are distinguished by their available resources and stability. Similar high-capability peers will be used to fortify the peer-to-peer network during expansion.

These peers re-distribute the video data to other connected peers in a hierarchical downward fashion. All peers in the network are viewers and every peer maintains a connection to a number of upstream peers. The entire video stream is sent (piece-by-piece) to each and every peer in the network via other peers.

5.4 OPERATIONAL DESIGN

This section describes the proposed implementation starting with the video source and ending with the viewer/client (in this system, clients can also be referred to as nodes or peers given the nature of the network). All aspects that are required are specified and described as requirements. The exploded view provides a greater level of detail for each and every component (as well as sub-components and functions) within the proposed implementation.

5.4.1 Media server

The media server is a dedicated server that encodes video directly from the video source. The video is compressed and packaged in a format that is suitable for streaming. The video stream is divided into small uniform-size chunks for more controlled distribution to peers (n chunks to each peer) and for lesser net effect of an error in the data stream. The design includes a separate media server (as opposed to an *all-in-one* server) such as to allow the media encoder to be on its own dedicated platform for faster encoding (including the encoding of multiple concurrent real time streams if required). This split-server or dual-server approach is one of the basic but distinguishing characteristics of this implementation.

5.4.1.1 Source interface

A generic video input interface allows the media server to encode any live input video. Video sources include:

- **Live analog video feed** from a camera, tape or a TV broadcast.
- **Digital video (DV) camera output** direct digital stream from a camera (such as a handheld camera).
- **Stored video file** such as typical MPEG playable files with various container formats (MPEG-1/2/4, DivX, VOB are examples).
- **DVD** for broadcasting a regular DVD to many users.
- **DVB** (digital video broadcasts) such as DTV or DVB-T/DVB-H.
- **An existing video stream** where the objective is to function as a stream repeater (or *booster*) or to convert a unicast stream to peer-to-peer multicast.

The interface should be scalable and adaptable to any type of standardized input video. Input stream handling should have low or negligible complexity and should provide the video frames to the encoder in an equivalent way for all input video types.

5.4.1.2 Video handler

This component determines the type of compression used to compress input video. If the input video is an analog source, the video handler will perform analog to digital conversion using high precision quantization. The video handler performs any pre-processing functions and delivers the video stream to the encoder for further preparation and compression.

5.4.1.3 Audio handler

The audio handler interprets the form of audio associated with the incoming video stream. Audio is decoded if required and passed on to the audio encoding component of the encoder.

5.4.1.4 Encoder

The encoder is made up of two components: the video compressor and the audio encoder. Video data is received and compressed in the minimum amount of time using the codec which has been chosen for the design. The implementation design allows for compressive codec selection which is not typically the case in the majority of peercasting systems.

5.4.1.5 Delivery component

This component is the element of the media server which is responsible for connecting to starter peers and supplying the starting peers with video chunks. The video chunks are received from the encoder and delivered directly to connected peers.

A single connection to each starter peer is established and maintained for the lifetime of the video stream. This connection is always kept open and used only for the unicast delivery of encoded video chunks.

5.4.1.6 Performance requirements

The media server needs to be capable of handling real-time decoding of incoming video (video sources) and subsequent real-time encoding of this video source to the desired output format.

The other functions (communication with tracking server and starter peers) do not contribute significantly to the overall processing load.

5.4.2 Tracking server

The tracking server manages peer connections to the network as well as initiating peer-to-peer communications. All communications happen directly between peers after an initial communication session establishment carried out by the tracking server. The tracking server also connects the starter peers and associates them with the already set up media server. The tracking server is responsible for building and managing the complete peer-to-peer streaming network.

5.4.2.1 Media server interface

This interface is very simple and merely provides the media server with connection information which is used to establish communications between the media server and the starter peers.

5.4.2.2 Communication with starter peers

Starter peers connect to the tracking server much like any peer. The difference being that starter peers announce to the tracking server that they have additional capabilities (as a super peer) and are willing to contribute a higher amount of resources.

5.4.2.3 Communication with regular peers

Regular peers initially connect to the tracking server in order to join the network. They are provided with the connection details of other peers within the network. This information is used to establish a link to the network.

Peers also announce their capabilities and willingness to provide upstream bandwidth for video data re-distribution. Periodic performance metrics (such as packet losses, latency, decoding efficiency *etc*) can be sent to the tracking server by peers. This information allows the tracking server to compute performance characteristics or to optimize operation of the network as a whole. The tracking server uses known information, feedback and statistical data to determine an ideal placement point for each and every new connecting peer.

5.4.2.4 Network management

The network management aspect of the tracking server deals with the initial setting up of the network based on the following peer characteristics:

- geographic location of peer
- historical information
- peer capabilities
- time of joining

In addition to initial setup, network management results in a continual re-shaping and re-optimization of the network when a node fails or when performance for a particular node drops below a certain level.

Performance indicators can be used to determine how efficiently video data is being transmitted and to ascertain if there are any *trouble areas* within the overall network structure.

5.4.2.5 Performance requirements

The tracking server is the main module within the network that has a direct impact on scalability. The performance of the media server is not dependent on the size of the network whereas for the tracking server this is not the case. A large network will require a very powerful tracking server due to the continual intensive connection management involved as well as the large number of peers that need to be controlled.

5.4.3 Starter peers

Super peers are regular peers with a greater level of available resources for contribution to the network. Starter peers are by default super peers and are expected to remain on the network for the entire duration of the media broadcast.

5.4.3.1 Media server interface

This interface is used to receive all original video data for re-distribution to the peer-to-peer network. The connection to the media server is a unicast connection over which video chunks are streamed in order. The starter peer should be able to receive the video chunks in the

specified format from any media server and as such, can be switched between media servers as required.

5.4.3.2 Connection with regular peers

Starter peers connect to regular peers using an identical interface to that which regular peers use to connect to each other. This way, the initial regular peers (connecting to the n starter peers) would receive video chunks as if they were receiving them from any normal peer within the network. This *hiding* of the starter peer's status allows for uniform peer interface design and removes the need for a specialized interface for starter and other super peers.

Video chunks are delivered to regular peers (known as *downstream* peers/nodes) in a one-to-many way. Starter peers are generally nodes with vast resources allowing distribution of video chunks to a greater number of downstream peers (*i.e.* 12) compared to regular peers (typically three) as shown in Figure 5.2.

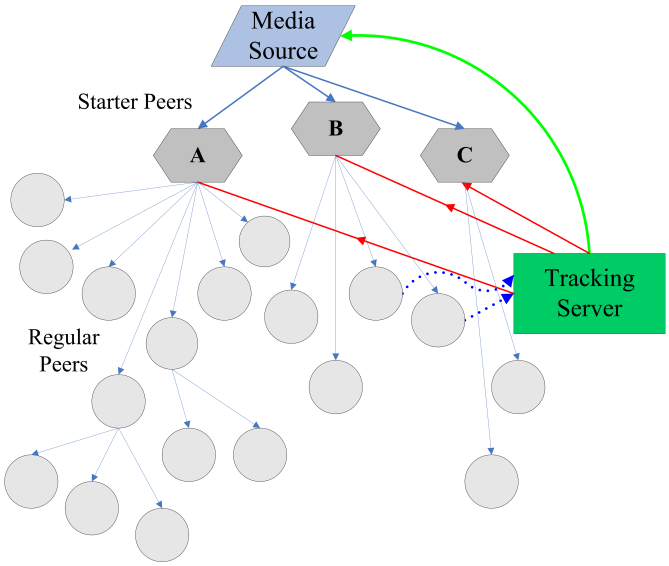


FIGURE 5.2: Starter peers connect to a large number of downstream peers while regular peers connect to only two or three downstream peers

The relatively large first or uppermost level of the peer-to-peer network prevents the formation of an unbalanced network from the outset. This is further highlighted by Table 5.1 (based on the structure of Figure 5.3) where the number of peers below each node is shown for a 1092 peer network. Figure 5.3 is based on the simplified structure where each regular peer supports only three downstream peers. The layout is simplified for clarity.

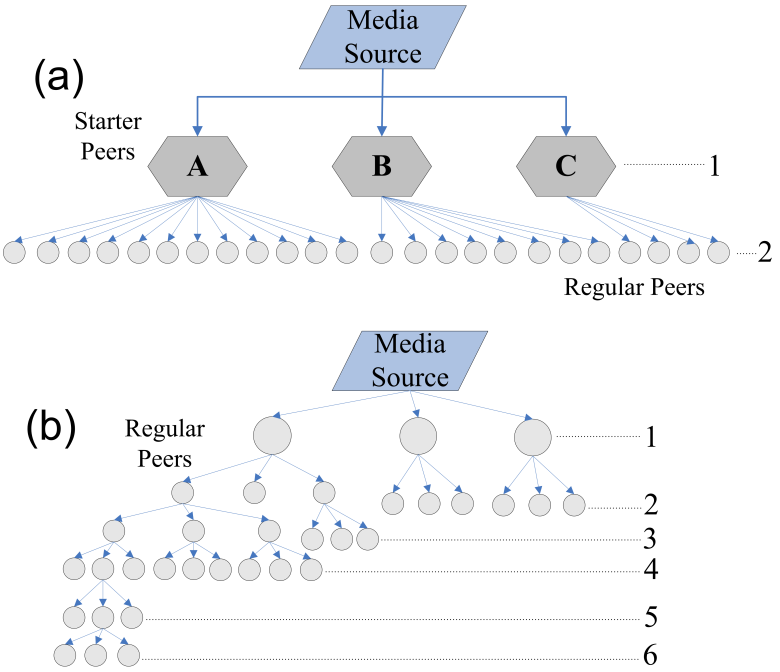


FIGURE 5.3: (a) High capacity starter peers with fewer levels (b) Regular starter peers

It can be seen that using high-capacity dedicated starter peers removes a large majority of the load (up to 67%) on regular peers thus allowing for greater scalability and fewer failures due to overloading. It is also evident that starter peers need to have significantly more resources than regular peers to be able to cope with the excessive load. Application of a stable layer of high-resource peers is unique to this implementation. According to research findings, no other peercasting implementations make use of a multicast layer and peer-to-peer layer configuration (as described above).

TABLE 5.1: Downstream peers per node for a 1092 peer network (refers to Figure 5.3)

Type of network	L1	L2	L3	L4	L5	L6	Total
(a) no distinct starter peers	972	81	27	9	3	-	1092
(b) dedicated starter peers	729	243	81	27	9	3	1092

It is important to note that in Table 5.1, the relative downstream network segment size for each peer is indicated. The network segment size does not place any direct load on the peer in question due to the nature of peer-to-peer. This number can be considered a rating of the severity of the possible failure of the given node (as this is the number of nodes that will lose connection to the network). It can therefore be seen that the reliable starter peer method offers a more robust network since fewer nodes are lost (and consequently need to be reconnected)

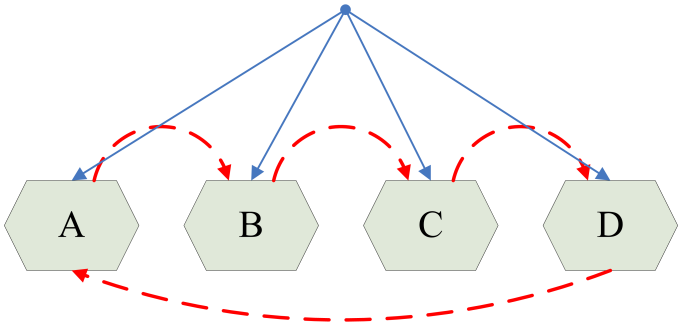


FIGURE 5.4: The redundancy relationship between starter peers (formation of a ring system)

while also improving overall performance.

5.4.3.3 Connection with other starter peers

Starter peers are assumed to be fully reliable at all times. However, due to the importance of the starter peers, it is imperative to make allowances for failure (however unlikely). Essentially, in a similar way to the redundancy method employed by regular peers (discussed later in this chapter), starter peers can maintain links between themselves and also maintain an updated database of their *partner* peer's connected downstream network. In a small network, these starter peers can connect in a fully meshed network structure.

In this partnership, each starter peers maintains two complete connection lists but only maintains connections to the primary list. Upon starter peer failure, new connection requests (directed at the backup starter peer) are initiated by downstream peers. While the original failed starter peer is considered under recovery, the secondary and primary connection groups are merged. This is loosely modelled on the peer joining method used in the SplitStream overlay network.

When the original starter peer resumes operational status, the backup peer begins a process of handing over a number of peers (based on several algorithms and metrics) in a *splitting* process. This way, downstream peers experience a service with only minimal stream loss during failure.

The redundancy architecture implies a ring approach (see Figure 5.4). If additional redundancy is desired, a secondary backup connection can be established where each starter peer maintains redundancy with two separate starter peers (*i.e.* an additional backup link).

5.4.3.4 Communication with tracking server

The starter peers behave similarly to regular peers. They connect to the tracking server in order to announce their presence and to obtain the address of the media server as well as addresses of other starter peers for redundancy (if required). The tracking server treats starter peer (and other super peer) connections with the highest priority due to their importance within the network.

The tracking server maintains constant communication with the starter peers. If a starter peer has extra capacity due to for example, a network segment failure, the next peer connecting to the streaming network will be connected to the starter peer in this open *slot*. In addition, starter peers can regularly update the tracking server with the number of successful video chunks delivered for performance measurement (as with any peer).

5.4.3.5 Performance requirements

The requirements for a starter peer concentrate on providing stability and reliability while not hindering the scalability of the network. It is assumed that powerful starter peers will be the backbone of the network and that there will be a very small number of them.

Starter peers need to:

- be dedicated servers
- be able to manage a large number of connections
- have very high upstream bandwidth
- have high uptime and reliability
- have very low latency switching capability for error recovery

5.4.3.6 Conversion during expansion

As the streaming network expands, the balance of the network will shift towards the top. In order to compensate for this, new peers that join and meet certain resource availability criteria can be converted to super peers. In the same way, connected peers with high resource availability can also be converted to super peers during expansion. These super peers then act as starter peers. They connect directly to the media server on joining. At the point where the

media server connection capacity is reached, these super peers slot in as regular peers but are exceptional in that they supply a far greater number of downstream peers with video chunks (in comparison to regular peers).

5.4.4 Regular client nodes

These are the peers that make up the bulk of the network. Regular nodes are referred to as clients or *viewers*. These nodes communicate only with the tracking server and other nodes like themselves. Their main function is to receive video data and play it back. Their secondary function is to re-distribute received video chunks. Regular nodes never come into contact with the media server and also see starter peers as regular neighbouring peers.

5.4.4.1 Video data transport module

This module is responsible for receiving and assembling video chunks. These chunks are used for playback and are thus fed into the decoder module. These video chunks are also repackaged (a hop tag is added if required) and sent untouched to downstream peers. The video transport module controls the reception and transmission of video data packets.

5.4.4.2 Communication with tracking server

Regular peers are largely controlled by the tracking server. When a viewer decides to view a particular stream, the stream is initially requested (by the client) from the tracking server. The tracking server returns a suitable list of peers which are offering to supply the viewer with a copy of the video stream.

The viewer can continually update the tracking server with various information including activity status, performance indicators and the status of downstream connections. At any point in time, the tracking server may initiate changes in the network which could include sending instructions to regular peers requiring re-configuration of the network by switching to other upstream peers (such as in the case of network re-balancing).

5.4.4.3 Communication with neighbouring peers

There are three classes of connections with neighbouring peers:

Upstream connection

In the context of the upstream connection, the peer connects to an upstream peer which will supply streaming video chunks. The establishment information for this connection is supplied by the tracking server. The peer attempts to connect to a list of provided peers. Once a successful connection has been established, the peer begins to request and subsequently receive video packets from the connected upstream peer (at the point at which the upstream peer is viewing/receiving packets within the stream or at the point in time where the stream was located when the peer joined the network).

Downstream connection

A downstream connection is established when the peer becomes a provider of video data. In this case, the peer advertises open slots to the tracking server. The peer then allows connection by a requesting client peer. After the connection has been established, the peer begins to deliver received video packets based on requests received from these connected downstream peers. This is a slightly more intelligent distribution system and offers error recovery. The request and deliver approach makes more efficient use of the peer-to-peer distribution system (when compared to other peer-to-peer part distribution schemes).

Redundancy connection

To prevent complete stream loss when upstream nodes fail, each and every peer can maintain a complex backup link table. This table lists the primary upstream peer as well as several backup peers. If the primary peer were to fail, the peer would attempt to connect to the upstream peers in the backup list. After successful reconnection, the peer receives video data at the point at which the newly connected node is delivering the stream. This redundancy architecture is loosely based on the recovery algorithms that form part of many other peercasting system designs.

All peers act as providers and therefore also become secondary and tertiary backup links for other peers. Continual updating of backup status is carried out at periodic intervals. If a node were to fail, the listed backup node will attempt to accommodate any peers that request the stream (within pre-determined limits *i.e.* only two additional peers are taken on).

5.4.4.4 Video decoding and playback

Video data packets received from upstream peers are processed to extract video chunks. These chunks are sequentially arranged and transferred into the playback buffer. The video decoder then decodes (by decompressing) these video chunks in order from the video data buffer. This allows playback of received video while more video chunks are being received and existing video chunks are transferred to connected downstream peers.

5.4.4.5 Performance requirements

Regular nodes have little or no performance requirements. It is not mandatory for a node to become a provider and to deliver packets to other nodes. However, with the minimal resources required to perform this function, it is highly unlikely that any regular node will be unable to redistribute received data packets to at least one downstream peer.

Basic requirements include:

- ability to maintain an information-only connection with the tracking server
- capability to decode received video chunks for playback
- resources to buffer video for playback
- ideally, sufficient resources to allow provision of video chunks to a reasonable number of downstream peers (for example, between three and five peers)

5.4.5 Privacy and security

Privacy and security are optional in the core design of the streaming system. If required, privacy can be ensured by implementing end-to-end security using SSL or PKI for message encryption. This secure channel can be used for sharing keys for data chunk encryption (using AES encryption with a sufficiently long key *i.e.* 256 bits). Each video chunk will be encrypted individually in case a set of chunks are lost. Chunk based encryption minimises the impact that secure mechanisms have on the functionality of the system.

Implementation of message privacy and data security adds additional processing overhead and increases latency by a small amount. However, security is essential in environments where users attempt to view the stream without contributing, try to view sensitive material or seek to maliciously tear down the streaming network (DoS attack).

5.5 ALTERNATIVES

There are a number of alternative designs or operational characteristics which are realisable within the framework of the current implementation design. By default, the streaming network exists in its entirety on a per-stream basis. This can make multiple channel support overly complex due to the complete network re-structuring for each stream.

A suitable alternative is the creation of a stable network over which multiple channels are streamed. This network type is also complex as not all connected nodes want to view every channel and often do not want to contribute upstream bandwidth if they are not active viewers. A node skipping algorithm can be designed where a non-active node in the multiple hop peer-to-peer chain is merely skipped (effectively removed for the channel being streamed) and re-connected after the streaming is complete.

The design focuses on the aspects of building a stable and reliable network while being simple to set up, operate and manage. Essentially, the design caters for a single broadcaster scenario where the media source is always known and is always the same. An alternative simple design adaptation would allow any peer to supply video information to the media server including allowing the media server to function as a regular peer in a separate network.

5.6 SUMMARY

This chapter provided a conceptual overview of the proposed peer-to-peer video streaming implementation. The building blocks of the reference network were introduced and briefly described along with some pertinent design decisions. Distinguishing characteristics and important contributions of this work were highlighted. This coverage serves as an introduction to the detailed implementation design specification in the following chapter. Implementation analysis, simulation results and conclusions are provided in a subsequent chapter.

CHAPTER SIX
IMPLEMENTATION

6.1 INTRODUCTION

The resultant design is constructed based on the outcomes of the literature survey where several related implementations were examined. Aspects of the design were gleaned from these implementations as well as from several publications and resources in the respective fields. The complete systems-level design is introduced in the previous chapter. This chapter details the development process for the application and provides a complete reference implementation with detailed design specifications.

The design of the peer-to-peer component of the system is largely based on findings of the survey of peer-to-peer carried out in Chapter 3. Reference networks (such as Bittorrent) were also used as inspiration. The overall structure of the peer-to-peer network is unique and is designed to offer potential solutions to common streaming network problems. The video compression component is designed to allow for simple replacement of the video codec if required and is thus not tied to a single encoding platform or technology. Design choices and their ramifications during development are thoroughly explained in the following sections.

6.2 DESIGN ELEMENTS

The following design elements are the main focus areas for the implementation. The purpose of this chapter is to completely specify the design elements as well as the reference implementation in such a way that actual implementation in a production environment is non-challenging and a simple conversion of the specification into functional programming source code.

Critical design elements include:

- Video handler and encoder

- Communication modules
- Data and communication protocol and packet formats
- Feedback modules
- Peer controller
- Incentive mechanisms

6.3 OPERATIONAL FLOW DIAGRAMS

Flow diagrams are provided to clarify the different operations within the entire implementation. The overall flow of data within the system is shown in the data flow diagram in Figure 6.1 where the path of video data is shown from the original source down to the viewer (or peer). Interactions with the tracking server are shown. The end user (viewer) redistributes received data to connected peers. Figure 6.2 shows the communication message flow for the

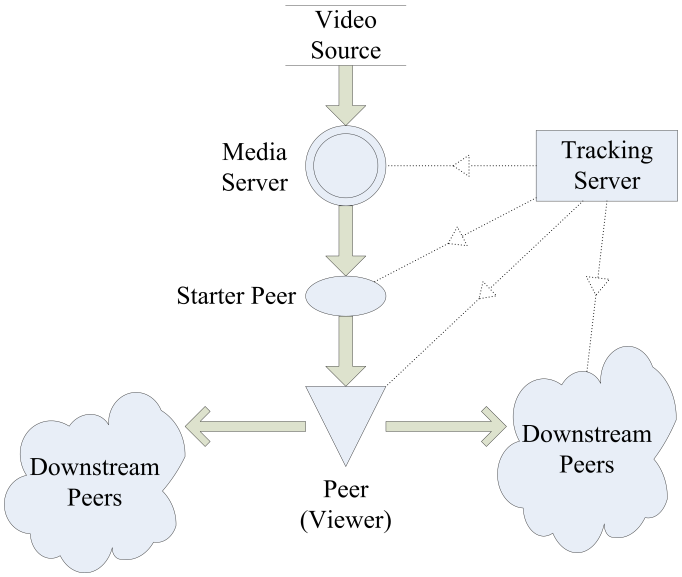


FIGURE 6.1: Data flow diagram for a single peer in the peer-to-peer video streaming network

tracking server. This diagram shows all the different messaging relationships between the tracking server and the other entities in the network. Starter peer selection is used to determine which starter peers will serve initial regular peers. Another starter peer selection module supplies the media server with the addresses of the starter peers. Peer location is used to determine where to place new peers within the peer-to-peer network. The relationship between regular peers and other components of the network is shown in Figure 6.3. For simplicity, a

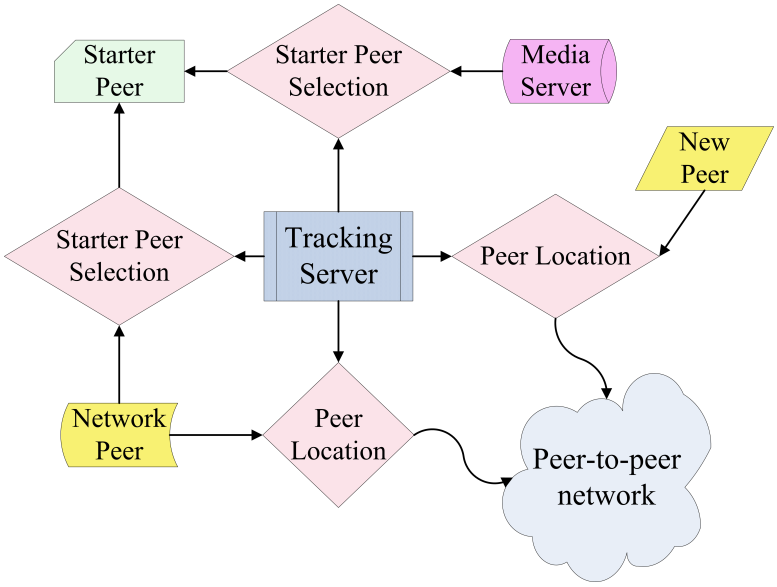


FIGURE 6.2: Control flow diagram for the tracking server

single feedback path is shown from downstream peers to the tracking server. A regular peer views a starter peer as any other upstream peer. Essentially, each regular peer maintains only two types of upstream connection: one to the tracking server and another to the source peer(s).

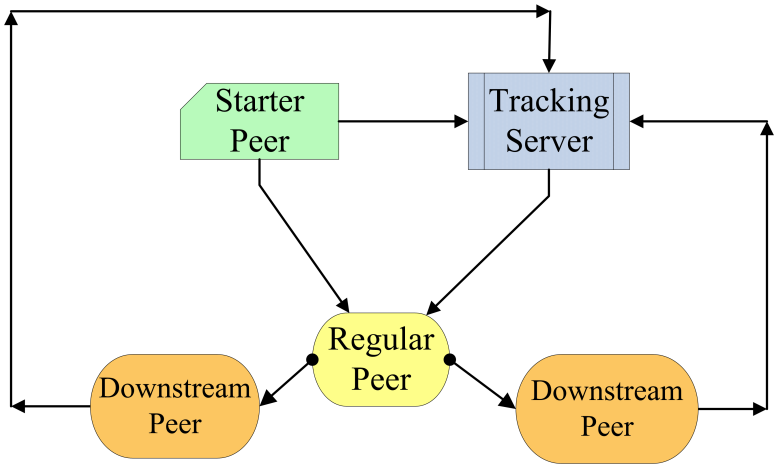


FIGURE 6.3: The relationship between a regular peer and other elements within the network

The following sub-sections indicate the sequence of steps for the various actions within the network.

6.3.1 Streaming system initialisation

The streaming system is initialised by the tracking server (the controlling entity). An index of connected peers is maintained but peer connections are not directly controlled by the

centralised server. For this reason, the hybrid network is of a partially centralised nature.

The tracking server listens for connections from the media server, starter peers and regular peers. The media server initialises the streaming system by connecting to the tracking server and announcing that it is ready to stream data. Starter peers initiate the reception of the data stream by connecting to the tracking server and subsequently sending a connection request to the media server. The media server listens for these requests from starter peers. At this point, the system is ready to stream data. When a sufficient number of starter peers have been connected, the media server starts streaming data to these starter peers.

6.3.2 Initialising video stream

The initialisation of the video stream involves the connection of the media server to the video source. Source video is prepared, encoded and packaged for distribution. The tracking server connects starter peers to the media server by providing the address of the media server when the starter peers announce their presence.

The process is as follows:

1. The media server announces its presence to the tracking server at run time
2. The connection between the media server and the video source is established by the tracking server
3. The streaming address and port is sent to starter peers (in response to connection requests)
4. Connections are established and starter peers begin to receive data directly from the media server

6.3.3 Unicast streaming layer

Each starter peer connects directly to the media server. This allows the formation of a unicast streaming layer where each node receives a full copy of the stream (*byte-by-byte*) from the media server. The benefits of utilising a unicast layer include:

- Higher reliability due to stable downstream peers
- Makes optimal use of the outgoing bandwidth of the server

- The distribution of bandwidth is more balanced resulting in fewer proportional leaf nodes
- Higher QoS as initial peer-to-peer nodes receive streaming data from highly reliable source nodes

The inclusion of the unicast streaming layer makes the proposed implementation more efficient and more adaptable. With no stable unicast peers, the media server would be sending copies of the stream to unreliable peers requiring excessive and expensive stream duplication. This system aims to exploit the advantages of unicast streaming combined with the vastly improved scalability offered by using peer-to-peer as a distribution mechanism.

A notable advantage of the unicast layer is that the media server can specify the extent of the starter peer layer based on how much upload bandwidth is available. For small numbers of starter peers (below four), the operation of the entire network becomes only peer-to-peer (*i.e.* unicast layer is essentially peer-to-peer as well). In order to circumvent difficulties that arise due to conditions where bandwidth supply is asymmetric, a low starter peer count is discouraged as a low number of starter peers has a significant effect on the overall efficiency of the streaming network. This can be mitigated to some degree by placing super peers in the downstream network.

It is assumed that each starter peer has higher than average upload bandwidth availability. In contrast to regular peers (who stream to three or four downstream peers), each starter peer connects to between five and twenty downstream peers in the peer-to-peer network. The higher number of connections reduces the overall effect of asymmetric subscriber bandwidth.

6.3.4 Initial peer connection

The initial connection of a peer takes place when the peer joins the network. The act of joining is technically separate to requesting a stream but these two actions typically occur at the same time as nodes typically join the network with the sole intent of viewing a video stream. Separating the initial connection from the stream request allows consideration for the case where peers join the network and then subsequently browse the available video streams and select one to view (as an example). This and an optional file based stream discovery system discussed below are add-ons to the core implementation.

The steps are as follows:

1. Connection request sent to tracking server
2. Tracking server replies with a list of potential upstream peers offering connection
3. Peer attempts connection to each peer in the list until the minimum number of successful connections for fluid streaming are established
4. If all connection slots of the requested peer are full or connection timeouts occur, the peer re-requests a new source list from the tracking server
5. Streaming starts when the peer requests the initial stream data part from the first upstream peer

Figure 6.4 shows the process:

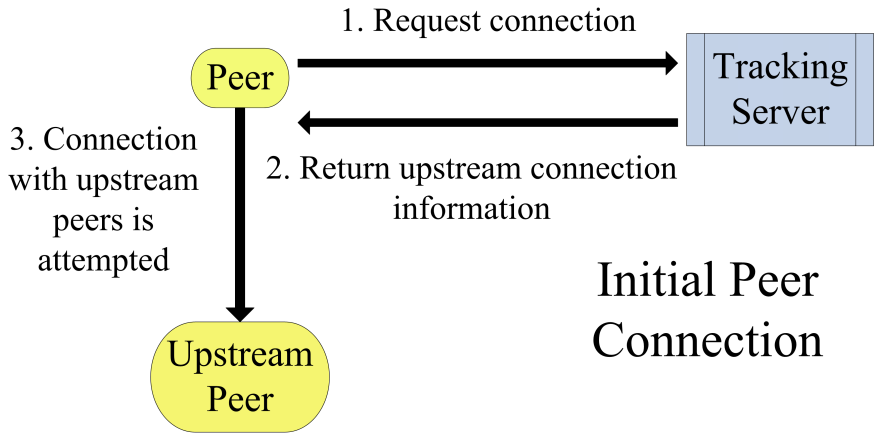


FIGURE 6.4: The intial peer connection process

The initial peer connection includes the connection of peers to secondary peers as a backup method in case of upstream node failure. Due to the lower priority of this process, it is carried out after a successful streaming connection is established. A secondary connection request is sent to the tracking server (with a backup flag to indicate that it is a redundancy request).

The process of connection is identical for secondary backup peers and as with the primary streaming connection setup, it is an iterative process which is carried out until a certain number of successful connections are established depending on the level of redundancy required. This backup connection is maintained using periodic *keepalive* messages. A peer will typically connect to three unique upstream sources and will maintain between one and three backup connections depending on the size of the peer-to-peer network.

6.3.5 Peer stream request and delivery

The peer stream request is sent to the tracking server. Typically, this function is carried out as part of the peer connection process. If a peer is already connected to upstream nodes, the tracking server will attempt to provide the requested stream over the existing network by checking if connected upstream nodes are already receiving the requested stream. If an upstream node is not a viewer of the requested stream, the requesting peer is disconnected from this particular peer and switched to a node which is able to provide the requested stream. This is the process in a multi-channel or multi-stream environment.

In a single stream network, the stream request is a basic action which entails the initialisation of the streaming process. No switching takes place as all peers are viewers of the stream that is being requested. This is the typical deployment scenario for the proposed implementation since in general, the streaming network is constructed *from scratch* each time a stream is set up. The platform is designed to support multiple streams (or channels) by implementing peer switching.

If a peer *is* switched in order to receive a different stream, all connected downstream peers need to be dealt with. These peers are connected to the original upstream node of the switched peer (subject to slot availability). If the number of open slots is not sufficient, these peers are re-classified as initial peers (*i.e.* newly connecting peers) and undergo the initial connection process in order to reconnect them to the network. A notable side effect of this switching process is potential re-buffering where the peers resume the stream at a slightly later position due to the delay between the hierarchical top of the network (their initial position) and the bottom of the network (the final position).

Subsequent to receiving a response (to the stream request) from the tracking server, a peer sends a stream setup request to the addresses supplied by the tracking server. This communication to initiate delivery of data packets takes places only between the peer and connected upstream peers. This is intended to remove networking load off the tracking server and thus improve overall scalability (or allow support for a less expensive, less powerful server).

6.3.6 Node failure recovery

When one of a node's upstream nodes fails, requests for data parts are reorganised temporarily so that the stream is requested from only connected nodes. At the same time, the node attempts connection to peers in the backup list. This is not always possible due to limits on upload slots for the redundancy peers. By design, each and every peer within the network should always maintain sufficient resources to support a single extra peer connection in the case of node loss. However, for efficiency, multiple backup connections are supported with the result being a possibility of zero backup slot availability for short periods of time.

If no backup slots are available, the processes of initial peer connection and stream request are carried out. Due to the high priority of this specific request, other new connection requests are immediately interrupted and consequently delayed. With the effective utilisation of a stream buffer, re-organised requests sent to connected upstream peers and timeous re-connection to a replacement upstream peer, the downstream peers should experience minimal or no stream loss whatsoever.

6.3.7 Performance feedback and network adjustment

Each peer within the network will send periodic updates to the tracking server if sufficient network capacity is available (this feature can be turned off to improve overall efficiency). These updates include the following:

- Counters of successfully received and decoded video packets
- Number of video packets delivered to downstream peers
- Periodic packet latency to upstream and downstream peers
- Potential disconnection warnings or outage notifications
- Any changes in the peer connection table (including change of redundancy peers)

6.4 SYSTEM DESIGN

The overall system is made up of a number of important functionally separate parts (or applications). These parts as well as their characteristics and interactions are described in this section.

6.4.1 Component Design

Each component within the overall system has a particular function. The four main components are unique in their structure, implementation and operation. The starter peer and regular peer components do share some functionality but are two separate components. The four components within the proposed implementation are documented below.

6.4.1.1 Tracking Server

The tracking server is designed to carry out the following functions:

- Process the ANNOUNCE packet from the media server and store the media server address
- Receive the starter peer CONNECT packet and reply with a SSETUP packet which contains the media server address information
- Listen for peer REQUEST packets and respond with appropriate source addresses for the peer (in a SETUP packet)
- Wait for a START packet sent by the media server when the stream is initiated
- Process performance feedback and network update packets if clients are configured to send these packets

The nature of the packets implies that the media server and starter peer packets are used for establishment and thus are only expected to be received once. In contrast, peer stream request and feedback packets will be expected to arrive on an ongoing basis. The arrival rate of these requests can be modelled as a Poisson process.

The structure of the tracking server is very simple. The block diagram in Figure 6.5 shows the different modules. The main functions are based on information reception (via packets) as well as the processing and generation of instructions (outgoing packets). The tracking server runs independently and waits to receive packets from the other network components of the system.

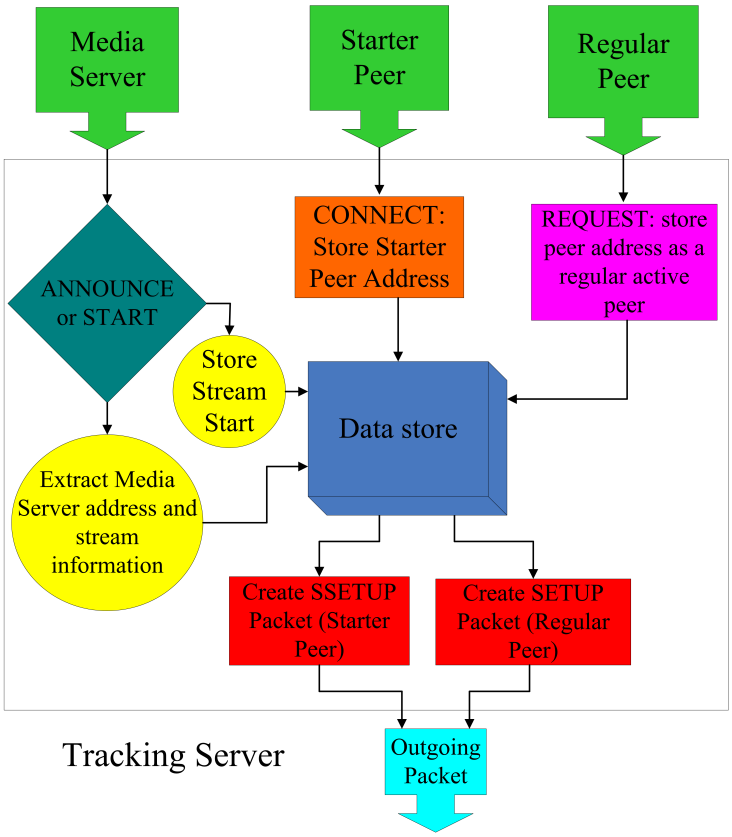


FIGURE 6.5: Functional block diagram for Tracking Server

6.4.1.2 Media Server

The media server has one primary function which is the conversion of input video to an encoded output video format for streaming. The media server has the necessary hardware for converting any form of input video (for example: capture cards, analog input, RCA, DV, HDMI *etc*) as well as the necessary software for decoding input video and/or conversion of already compressed or encoded video supplied from a digitally connected source or stored file. Video is encoded and delivered directly to starter peers.

The block diagram in Figure 6.6 summarises the functions of the media server.

6.4.1.3 Starter Peer

The starter peer is the interface between the unicast network (the starter peers and media server form a unicast layer) and the peer-to-peer network. Essentially the data is received via unicast and distributed using peer-to-peer. The block diagram in Figure 6.7 shows a simple overview of the two halves of the starter peer.

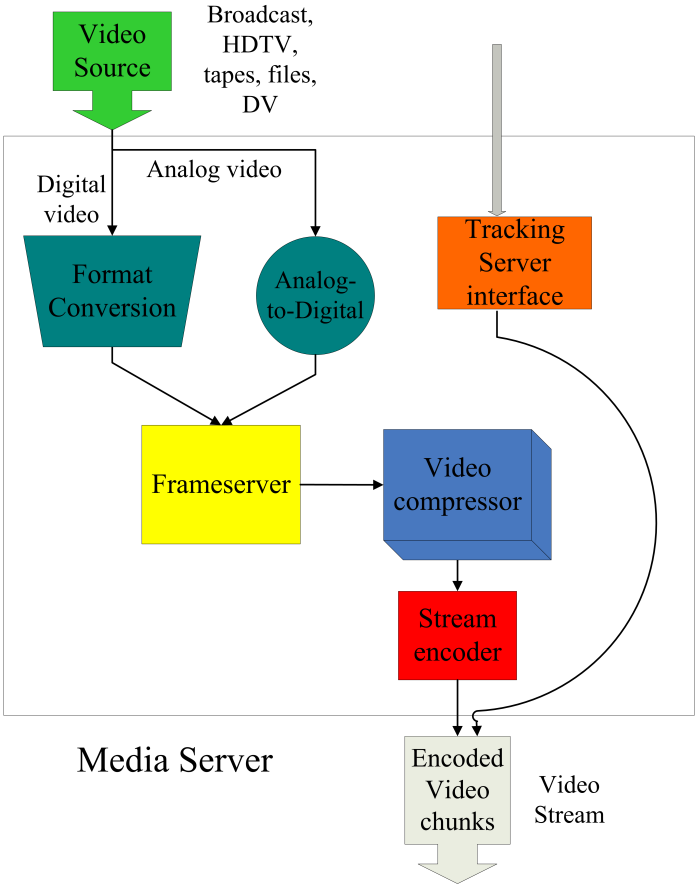


FIGURE 6.6: Functional block diagram for Media Server

6.4.1.4 Regular Peer

Regular peers operate as full peer-to-peer nodes requesting parts of the stream from other nodes. These requested parts are used to construct the incoming data stream in real time and supplying requested parts if they have already been received. The peer node includes sub-components for dealing with part request packets as well as for actual data packets. These sub-components as well as their interactions are shown in Figure 6.8.

The design encompasses a sub-component that continually checks the part register for received parts and generates PARTREQ packets for the parts that have not been received (yet). The part register is updated as DATA packets are received. For simplicity, the block diagram only indicates a single source (typically there could be three sources). The part register is a simple array (zero filled on initialisation) where indexed cells are mapped (by index reference) to the sequence of parts (by part ID) within the stream. As parts are received, their respective cells are updated to hold a non-zero value.

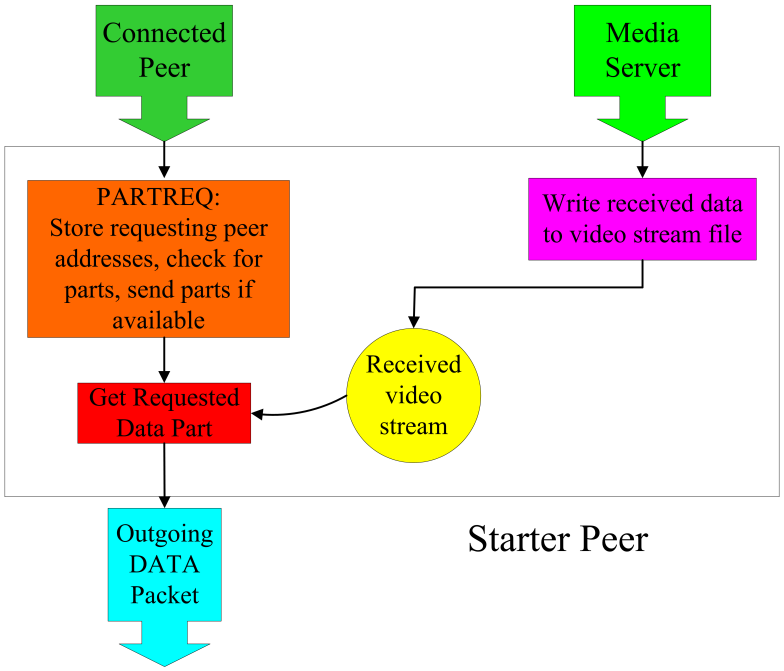


FIGURE 6.7: Functional block diagram for Starter Peer showing the unicast and peer-to-peer sections

6.4.2 Communication protocol

Within a system such as that of the proposed implementation, networking is a very important factor. The components within the system need to communicate and interact in order to perform the functions that they were designed to perform. The communication protocols between components are designed to have a low overhead and low complexity. The minimum number of messages (or packets) are sent in order to set up connections and initiate the streaming of video data to end users. For reliability, TCP is used for protocol messages. If required, UDP can be used for protocol messages but this will have an effect on reliability of message transmission. The various communication protocols and packet formats are documented in the following sub-sections. Detailed packet formats are provided in Appendix A.

6.4.2.1 Tracking server-Media server communication

The main communication involves the setting up of the starter peer connections to the media server. The media server sends an ANNOUNCE packet to the tracking server in order to announce its presence and willingness to begin streaming. The tracking server does not send a reply to this message but does record the media server address and stream information.

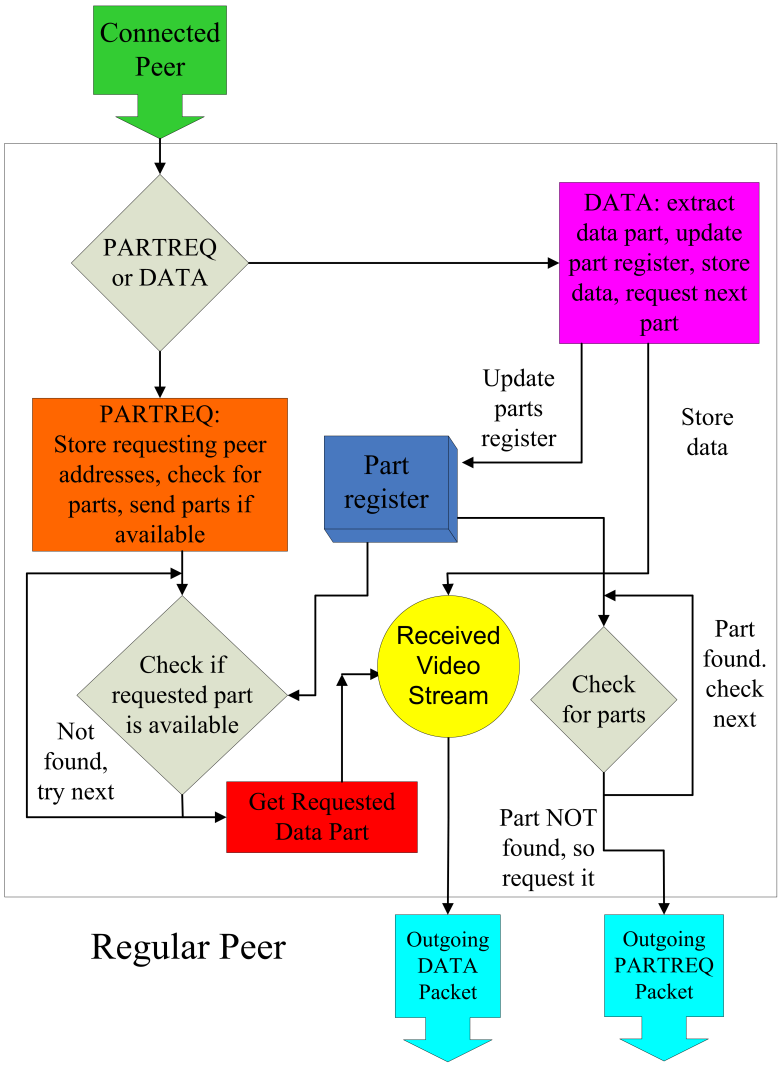


FIGURE 6.8: Functional block diagram for Regular Peer

Once the media server has received a sufficient number of starter peer connections, the stream is started. At this point, the media server sends a START packet to the tracking server such that the tracking server knows the exact point in time when the stream started. This information is used to provide a base reference for part requests by future viewers.

6.4.2.2 Tracking server-starter peer communication

Starter peers connect to the network by contacting the tracking server in order to receive the address of the media server. Subsequent communication occurs directly between the starter peers and the media server. This initial packet is the CONNECT packet which only contains

the address of the starter peer.

The tracking server responds with a SSETUP packet. The main purpose of this packet is to supply a unique ID for the requesting starter peer, the streamID for the current stream as well as the address of the media server.

6.4.2.3 Tracking server-regular peer communication

When regular nodes want to receive the stream, they send a connection REQUEST packet to the tracking server. This packet is very basic and only contains the address of the requesting node. The tracking server responds to the request with a SETUP packet.

All peers within the network send periodic updates to the tracking server. These updates are separate from and in addition to the optional performance feedback. Periodic updates are sent more frequently (adjustable) than performance feedback and are used mainly to update the tracking server on the status of all connected peers while at the same time announcing to the tracking server that the peer sending the update is still live. Reception of the packet (by the tracking server) implies that the sending peer is live.

6.4.2.4 Peer-to-peer communication

Initially, peers receive source addresses from the tracking server. These addresses are the addresses of regular nodes that are offering to send parts of the stream (on request) that they have already received. Starter nodes are seen as regular peers if they happen to be one of the sources sent by the tracking server. Once a regular peer has received the source addresses, the part request process is initiated.

PARTREQ packets are created to request parts of the stream starting at the starting point received from the tracking server in the SETUP packet. Parts are requested from upstream peers by part ID in sequential order using a *round robin* scheme. These PARTREQ packets are sent to source (upstream) nodes. Each packet contains a unique part ID for the requested part. Upon receipt of the PARTREQ packet, the part register is checked to determine whether or not the requested parts are available to be sent.

If a requested part is available, it is encapsulated in a PART packet and sent to the requesting peer. If the part is not found, the PART packet is not sent. In this case, the part

selection process and subsequent creation of PARTREQ packets will result in a second request for the same part at a later point (after a predefined timeout has elapsed). This request could be sent to a different upstream peer. If sent to the same original upstream peer, the probability is high that parts that had not been received at the time of the original request will be available for secondary requests.

6.4.3 Protocol Flow

The protocol described above is summarised in the form of an overview of the protocol flow (including the normal ordering of protocol messages) as presented in Figure 6.9.

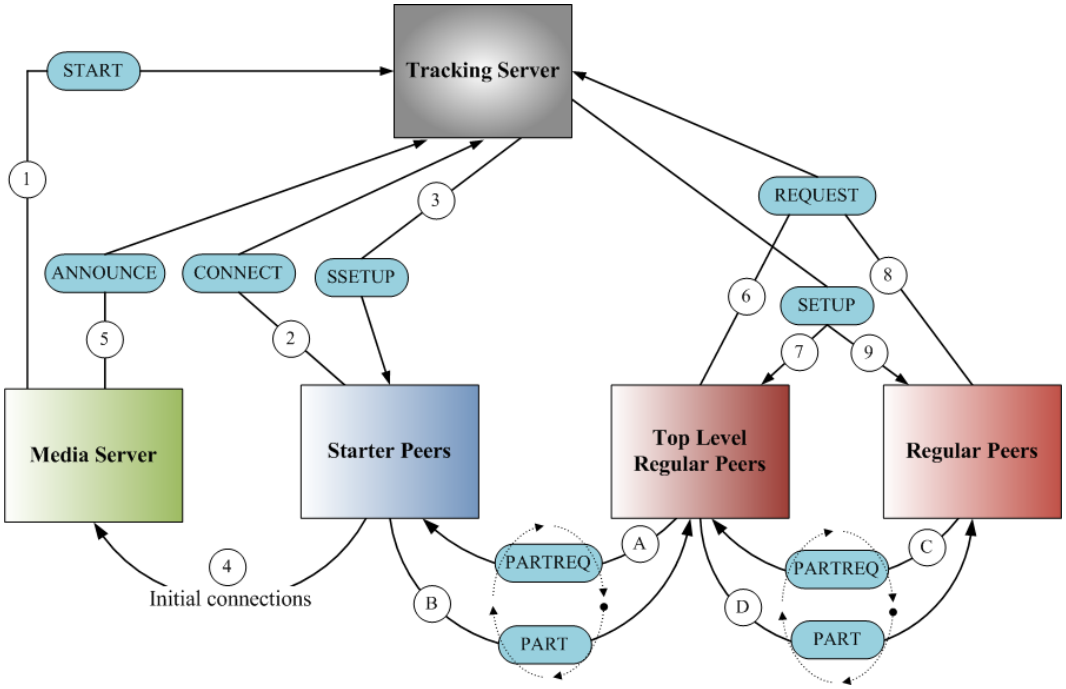


FIGURE 6.9: Overview of the protocol flow with the typical ordering of messages under normal operating conditions

6.4.4 Parameters of operation

Operational efficiency for a complex system (of which the proposed implementation is an example) is highly desirable. In order to provide efficiency under varying operational conditions and in a variety of scenarios, the implementation needs to be adaptable. An adaptable system lends itself to experimental optimisation.

Typical operating parameters are specified in Table 6.1.

TABLE 6.1: Typical operating parameters

Parameter	Description
Part size	16384 bytes
Minimum starter peers	3
Streaming container	ASF
Video codec	MPEG-4 Part 2
Video profile	Advanced Simple Profile
Video bitrate	800Kbps
Audio codec	MP3
Audio bitrate	96Kbps

6.4.4.1 Format support

The media server is designed to receive video from a number of sources. Frameserving is used to provide uniform input to the encoder. Frameserving is a technique whereby frames are delivered to the encoder (instead of rendered for display) irrespective of the source format. This enables support for digital sources (files, DV/HDV, DTV/HDTV/DVB) or analog input sources (cameras, tapes, film).

Streaming video codec and container format selection are important parameters. The codec used for video compression can easily be changed for a more efficient or more suitable codec. A simple configuration change allows drop-in replacement of both the video codec and encoding container format. Different codecs offer different suitability (for example, codecs optimised for streaming) and different container formats specify extensions and adaptations which impact streaming and allow additional streams, features and subtitles.

6.4.4.2 Performance parameters

The most important performance parameter is the size of data parts. Smaller part sizes result in a significant increase in part requests effectively lowering the network efficiency due to the higher volume of request packets being transmitted. Greater part size raises the susceptibility to stream loss when packets are delayed or dropped.

The encoding bit rate is a parameter that has a very large impact on the performance of the streaming network as a whole. This parameter is closely related to the overall or average upstream bandwidth. These factors should be taken into account when choosing a bit rate for

the stream.

A possible scheme for adapting the bit rate could be incorporated as follows. Each peer sends their available upstream bandwidth in update messages to the tracking server (or this can be calculated dynamically). Using the received data, the average upstream bandwidth can be calculated to determine what the streaming bit rate should be. If the bit rate is set within this average limit, then the resultant stream QoE should be maintainable. This is a parameter that can be dynamically adjusted.

6.4.4.3 Limits

A number of limits can be adjusted to manage and control performance of the various components in the overall system. The most important adjustable limit is the number of starter peers that can be connected. This limit is dependent on the allocated outgoing bandwidth made available by the media server and is set accordingly.

Other important limits include the number of sources that a peer requests data parts from. More sources means more requests and thus more overhead. However, more sources also results in a more reliable and smoother video stream. The number of backup sources can also be limited to a small number to reduce network bandwidth consumption.

The number of outgoing connections for a regular peer is a limit dictated by available upstream bandwidth. This limit will be different for each peer within the network. The amount of upstream bandwidth made available is typically related to the effect of incentive mechanisms which are discussed below.

6.4.4.4 Expansion

The proposed implementation is designed to cater for dynamic expansion. As the streaming network grows, the network scaling factor is dependent on the peer-to-peer algorithm. This is the main advantage of peer-to-peer since peer-to-peer networks (theoretically and by practical experimentation) scale well. The proposed implementation allows for additional scalability by incorporating more starter peers and by maximising the available resources of connected peers (*i.e.* by utilising super peers).

In addition, a multi-tier starter peer network can be created using additional starter

peers. Similarly, a multicast system can be implemented within the starter peer layer or alongside the peer-to-peer network. This type of expansion is beyond the scope of this implementation in its current state but exhibits excellent potential for greater *horizontal* scale and more efficient network distribution.

6.4.4.5 Incentive mechanisms

Typical broadband Internet connections are asymmetric resulting in a low amount of available upstream bandwidth per node. This leads to users actively restricting outgoing connections in order to prevent sacrificing overall bandwidth performance (for the benefit of others). For this reason, incentive mechanisms may need to be incorporated. Since these mechanisms are an *add-on* to the network, they are viewed as a performance parameter.

Highly effective incentive mechanisms can result in greater overall performance for the peer-to-peer streaming network. Example incentive mechanisms include the following rewards for greater contributed upload bandwidth: movement upwards in the streaming hierarchy (for greater QoE), early access to future streams, lower subscription costs (if applicable), access to additional sub-channels (for example, layered encoding enables provision of higher definition audio or video streams as an incentive) or additional privacy and anonymity (if desired).

6.4.4.6 Deployment environment

Several parameters can be adjusted to suit the deployment environment. These parameters will typically alter the system operation to make allowances for effects that the deployment scenario introduces. A typical example is the latency introduced over a wireless network. Deployment in a high speed, high bandwidth network (such as a large LAN or high bandwidth broadband network) will allow for higher limits on peer connections, a greater number of starter peers and higher video encoding bit rates (HDTV support is an example).

6.4.4.7 Other applications

The proposed implementation can be adjusted for deployment in other streaming scenarios. The video codec can easily be changed for an audio-only codec to enable audio streaming. The actual encoder can be seen as a parameter of operation and is merely a processing block. The design caters for replacement of the video compressor with a data or file compression application. This would enable distributed file dissemination. Some adaptations that will need

to be made include revised part definition or an adjustment to the part request system to enable dissemination of the compressed data (or file) from the beginning to the end of the outgoing data stream.

6.5 SYSTEM SPECIFICATIONS

This section defines the design specifications for the system. Implemented technologies and specific aspects of the design configuration are covered.

6.5.1 Tracking Server

The tracking server is designed to accept packets from all other entities within the system on a configurable port number (the default port is 7979). The tracking server is purely reactive and only responds to incoming packets. Embedded information is processed and used to create outgoing packets which are sent to the respective entities. Other than setup, feedback processing and control - the tracking server has no other function(s).

The tracking server receives the stream start time from the media server when the stream is started. This start time is used to create a zero point for the stream. Using the partsize, the current stream location is calculated based on the starting point. This current stream position is supplied to peers requesting the stream. Equation 6.1 is used to calculate the current streaming point in parts (based on part identifiers) since the zero point.

$$\mathrm{CSP} = \left(\frac{\text{bit rate}}{\text{partsize}}\right) * \left(\mathrm{t}_{\text{current}} - \mathrm{t}_{\text{start}}\right) \tag{6.1}$$

Each entity in the network is assigned a unique numerical identifier. These identifiers are used for various purposes. The identifiers are assigned by the tracking server according to the convention outlined in Table 6.2. The tracking server is designed to run on Linux and is

TABLE 6.2: Unique Numerical Identifier assignment convention (x represents identifier)

Normal Peer	$0 < x < 999999$
Starter Peer	$1000000 < x < 1999999$
Media Server	$2000000 < x < 2999999$

developed under the open source model in the sense that open source components were used wherever possible and that the source code is not intended to be commercialised.

6.5.2 Media server and video encoder

The media server includes a video compressor that is based on ffmpeg [51]. Replacement of the chosen video codec implies a simple configuration change in the media server.

Connection to any number of starter peers is supported subject to third-party restrictions such as available upstream bandwidth as well as processing power to support encoding and streaming to *n* starter peers simultaneously.

The media server is designed to run on Linux and supports any form of video or audio input. The maximum number of supported starter node connections (specified as *spmax*) can be set at runtime and can be dynamically adjusted. The media server typically listens for packets on port 7978 (this can be set at runtime if required). When the video stream is initiated, the current system time is read and sent to the tracking server.

Source inputs to the video server include files, hardware inputs from capture devices and input received from other streams (such as live broadcasts). Essentially, the media server can encode video from any video source. A built-in frame server allows for encoding of source frames. Using this configuration, the media server can be used as a stream repeater or convertor (for example, converting and streaming a high definition source over a network with relatively low average bandwidth).

Encoded data is streamed to starter nodes using unicast streaming. Each encoded data block is cloned and sent to each starter node. The media server assigns a numerical identifier to the stream. This identifier is compatible with the identifiers of all the entities within the system as the identifiers are designed to be unique within their assigned ranges. The identifier follows the global naming convention (as shown in Table 6.3).

Table 6.3: Unique Numerical Identifier assignment convention for the stream (x represents identifier)

Stream Identifier	$3000000 < x < 3999999$

6.5.3 Starter peer specifications

Starter nodes form part of a unicast streaming network. They receive streamed data from the media server. All received video data is written to file. This file acts as the data buffer for the stream. Incoming part requests (by regular peers) are served from this buffer. The file is operated as a FIFO buffer for long streams as only the oldest data is discarded as new data is written (unless sufficient storage space is available to store the entire received stream).

The stream is served to regular peers over the peer-to-peer interface. This interface is constructed in such a way that regular peers view starter nodes as typical regular peers and thus cannot distinguish these peers from any others. The starter node listens for part requests and delivers requested parts to requesting peers. The key difference is that the starter node does not need to check a parts register for the existence of requested parts as it can be assumed (implicitly) that the starter node will always have every part of the stream (up to the point when the request was initiated).

By design, starter nodes use two distinct port ranges (for the unicast and peer-to-peer sections respectively). These port ranges are dynamically assigned at runtime. The typical ranges used are 7950-7960 (usually 7950) for the unicast listener and 8001-8020 for the peer-to-peer interface.

6.5.4 Regular Peer

Regular peers receive the stream over the peer-to-peer network exclusively. An initial connection to the tracking server leads to connection to source peers. The initial response from the tracking server includes the current stream point identifier. This identifier is used as the first part requested and is mapped to the zero point in the output playback buffer.

Stream data parts are requested from provided sources. As each part is received, it is used to construct the output streaming file. Parts are written to this file in the correct locations calculated using their part identifiers. The stream output file is filled pseudo-randomly (as opposed to FIFO or LIFO) and read linearly from the beginning during stream playback. Playback only begins after a pre-selected number of parts (sufficient to pre-load 30 seconds of the stream) have been successfully received.

6.6 PEER-TO-PEER ALGORITHM

An algorithm for peer-to-peer file transfer forms the core for the proposed implementation. The implemented algorithm is relatively simple and is designed to be deployable with low overhead. The algorithm is sporadically described above. This section provides a summary of the relatively simple peer-to-peer algorithm.

The source stream is split into parts of a pre-set size. These parts are indexed from zero (which is the start of the stream). Peers request parts from the point at which they joined the stream. These requests can be chained in two ways. The first way is grouping of requests where groups of up to five parts are requested (this is the *super block* request method) from a source peer as shown in Figure 6.10.

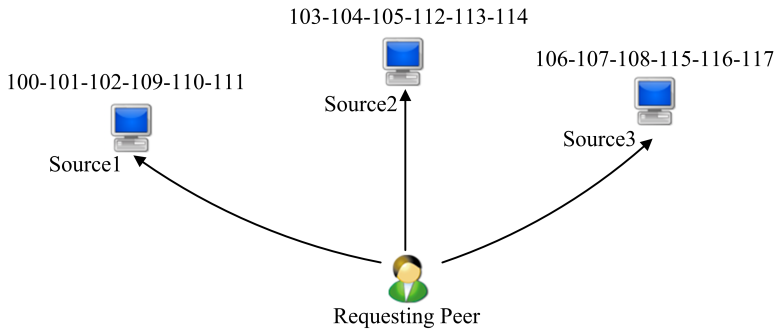


FIGURE 6.10: Example of the *super block* request method

The second method uses a round robin approach (Figure 6.11) where every n-th part is requested from each of the n source peers. In both request formats, any number of parts can be requested in a single packet. The number of parts requested is dependent on the partsize for optimal efficiency.

As a peer receives data parts, these parts are stored in a buffer. The parts are stored in ordered locations indexed from a zero point created using the received starting index for the stream. This index is the point at which the peer requested the stream. This point is mapped to the beginning of the buffer. As can be seen in Figure 6.12, parts do not necessarily arrive in order. This creates *holes* in the buffer. To compensate for this, the peer waits for a pre-determined time (typically 30 seconds) before starting playback. This system relies on the fact that parts can be retrieved slightly faster than the playback rate. Using the example in Figure 6.12, the combined packet delivery (or download) rate should be roughly 1024Kbps which resembles

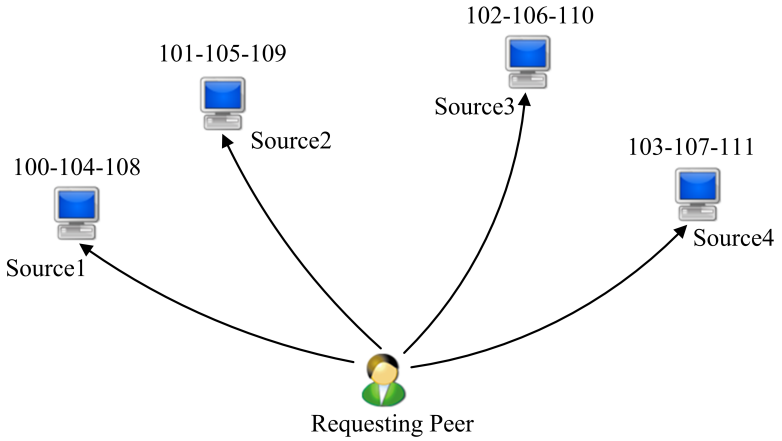
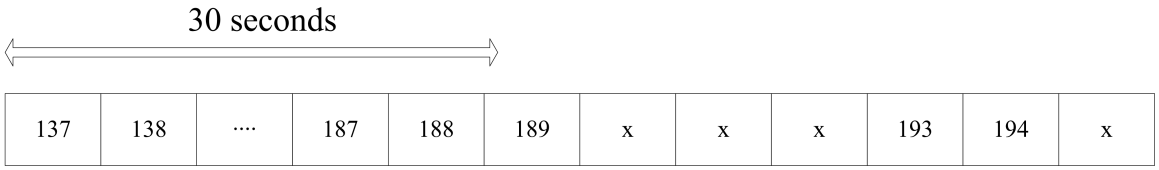


FIGURE 6.11: Round robin part request example

typical ADSL broadband downlink throughput.



Encoding Bitrate = 900Kbps
Partsize = 65536 bytes
Part length = 581ms

FIGURE 6.12: An example playback buffer that is filled as parts are received. The buffer is based on the parameters indicated.

6.7 SUMMARY

This chapter builds on the design introduced in the previous chapter. Design details and specifications for the proposed implementation are provided. These include operational descriptions, interactions, component design, design choices and specifications for components and the system as a whole.

The proposed implementation is designed as a reference application of peer-to-peer video streaming. The main objective of the development process is to determine and present any beneficial outcomes of implementing the proposed implementation when compared to traditional streaming systems. By definition, this implies demonstrating a higher limit on scalability and more efficient bandwidth usage in the server side.

A secondary aim of the design is to provide a reference framework for peer-to-peer based video streaming applications. The proposed design includes elements that address the issues of redundancy to failure, modularity to allow additional channel (or stream) support, adaptable video compression and support for various forms of multimedia apart from video.

The implementation was designed and specified using a text-based packet format for readability and simplified debugging. Optimisation would result in a field based binary format for protocol packets. The aim of the implementation is proof of concept and not full optimisation and robust design.

Results derived from testing by analysis and simulation are presented in the following chapter.

CHAPTER SEVEN
EVALUATION OF PROPOSED IMPLEMENTATION

7.1 INTRODUCTION

The design and implementation of the proposed peer-to-peer video streaming solution concludes with performance analysis and functional evaluation. This chapter provides basic analytical models for estimating performance. The testing environment is described as well as the methods used to test the various components (and their interactions). The performance of the system as a whole is evaluated based on a typical deployment scenario with a suitable viewer audience size. Finally, full analysis and experimental results are provided along with a discussion of the observed performance characteristics.

7.2 EXPERIMENTAL SETUP

This section describes the environment in which analysis and testing of the implementation was performed. The construction of the network as well as how it models a typical network is defined. Operating parameters are defined and explained with reference to how they affect the evaluation environment.

7.2.1 Simulation environment

The simulated network environment is constructed to model a real network in which a peer-to-peer video streaming service would be deployed. The system parameters are used to adapt the environment to ascertain how the proposed implementation would perform in the *real world*.

7.2.1.1 Network architecture

The simulation environment resembles a typical network with multiple connected nodes. This network is in reality a small LAN with high speed interconnections and composed of relatively

powerful blade servers. Real network performance is simulated using trickle [88] (a user space bandwidth shaper) for each application instance. The network bandwidth available to each peer is set at typical speeds for a 1Mbps ADSL broadband service (1024Kbps downlink, 512Kbps uplink bandwidth).

By definition, starter and super peers have greater bandwidth availability and are thus configured as 4Mbps subscribers (4096Kbps downlink, 2048Kbps uplink bandwidth for FTTC, Metro Ethernet or Annex M xDSL variants). As the system is reliant on upload capacity, performance would be equivalent for any variant with the same upload capacity (*i.e.* 8096Kbps/2048Kbps up to 24288Kbps/2048Kbps). It is assumed that the media server is a typical commercial media streaming server and is thus set to have full symmetric 100Mbps bandwidth. This means that the media server can theoretically support up to 25 starter peers.

7.2.1.2 System parameters

Given that the proposed implementation is designed for a typical 1Mbps service, the overall video streaming bit rate is set at roughly 90% of the downlink bandwidth. Video data is encoded for streaming at 912Kbps which comprises the following: 784Kbps for the video stream, 112Kbps allocated to the audio stream and approximately 16Kbps for container packaging overhead. The encoding rate selection is discussed and analysed in a later section.

The video format used in testing was MPEG-4 Part 2 (ASP) with MP3 for audio in an ASF container. The resolution (frame size) used is 720x576 (DVD standard).

7.3 MATHEMATICAL ANALYSIS

This section presents a mathematical analysis of the proposed implementation. The aim is to derive performance characteristics based on known analysis methods. The analysis gives rise to a set of equations that can be used to predict the performance of all elements within the streaming system.

The nature of the implementation results in exponentially increasing difficulty when evaluating performance on anything but a relatively small scale. The use of mathematical analysis for design verification seeks to overcome this challenge. The output of numerical analysis can be combined with experimentation results such as to show that expected

performance characteristics of the peer-to-peer streaming system are attainable in reality. Since there is no clear cut procedure for analysing a system of this nature, a number of approaches are followed to develop a set of viewpoints from which analytical models are developed. These sub-equations can then be seen as components of a global system evaluation and can be applied to similar or alternative implementations (if required).

7.3.1 Video encoder

The video encoder is responsible for preparing the source video stream. Two main factors are investigated: encoding speed and quality of encoding. Encoding rate is a function of processing speed, frame resolution, chosen frame playback rate (in frames per second) and codec selection. In order to realise a reliable live video stream, the minimal condition for encoding (or the minimal encoding rate r_e in Kbps) is that $r_e > r_p$ where r_p is defined as the video playback rate in Kbps (Equation 7.1). The frame size f_{size} is specified as the average number of bytes for each individual frame while f_{rate} is the number of frames played each second (*i.e.* frames per second).

$$r_p = f_{\text{size}} * f_{\text{rate}} \tag{7.1}$$

r_e is typically determined experimentally based on the selected codec. This rate is dependent on the parameters of encoding and the processing speed of the encoding system (*i.e.* CPU speed). Based on experimentally determined performance characteristics for a given system and codec, it is possible to determine whether this encoding system is able to deliver a fully compressed output stream encoded in real time.

7.3.2 Communication protocol

Performance of the communication protocol is not verifiable mathematically. However, an expression for overhead per unit time can be derived. Inputs include message size m_{size} which specifies the actual size in bytes of each protocol message and messaging rate m_{rate} (the sending regularity for each specific message). Overhead can be calculated and specified in *bytes/second* and subsequently represented as an average percentage (o_{ratio}) of the overall available transmission bandwidth (bw_{total}). This portion is determined using Equation 7.2.

$$o_{\text{ratio}} = \frac{m_{\text{size}} * m_{\text{rate}}}{\text{bw}_{\text{total}}} \tag{7.2}$$

The ratio of overhead to total bandwidth (o_{ratio}) specifies the *average* overhead. The instantaneous nature of overhead messages means that the average bandwidth consumption is

not the most accurate measure. However, due to the small size of messages compared to the overall bandwidth (typically, $m_{\text{size}} < 120$ bytes for a 65536 bytes/second bandwidth) and the fact that messages are not bandwith-sensitive, this is not a factor. The size of these message *spikes* (in terms of bandwith consumption) is negligible in relation to the total bandwidth (below 0.2% if the message is sent over one second and if the message is sent at line rate, it will only take approximately 2ms to transfer the message).

m_{rate} is calculated based on m_{delay} (from Equation 7.3) using Equation 7.4. In Equation 7.3, p_n indicates the number of packets sent/received at time t_n. The adjustment factor is used to account for dynamic behaviour and is typically set between 0.90 and 1.05 implying that packets can be transmitted marginally faster than expected or that packets can be slightly delayed. This factor is dependent on historical known network performance and while being flexible is static once determined.

$$m_{\text{delay}} = \frac{p_2 - p_1}{t_2 - t_1} * \text{adj}_{\text{factor}} \tag{7.3}$$

$$m_{\text{rate}} = \frac{1}{m_{\text{delay}}} \tag{7.4}$$

Equation 7.5 is the complete equation for calculating o_{ratio}:

$$o_{\text{ratio}} = \frac{\frac{m_{\text{size}}}{\text{adj}_{\text{factor}}} * \frac{t_2 - t_1}{p_2 - p_1}}{\text{bw}_{\text{total}}} \tag{7.5}$$

7.3.3 Peer-to-peer streaming

The entire implementation is designed around the peer-to-peer algorithm. This algorithm is the most important and influential element in the system. Mathematical analysis is carried out as a means to predict performance and to expose problems when scaling up user numbers as the streaming network grows. This is only a theoretical prediction that can be paired with testing results to give an indication of expected performance characteristics.

7.3.3.1 Assumptions and methodology

This analysis largely follows the model introduced in [89] with some modifications to suit the proposed implementation. For clarity, the original notation is maintained. This model uses stochastic fluid theory and as the implementation is generally *churnless*, the system is assumed to have a fixed set of peers during streaming.

The reference configuration is analysed to ascertain whether or not full streaming to each and every viewer is possible. This part of the analysis will rely on the assumption that the streaming system architecture is fully balanced and that resources and peers are evenly distributed.

7.3.3.2 Peer differentiation

Video data originates at the media server and is specified as u_s (in bits/second) which is the total streaming upload rate across all connections. u_n specifies the upload capacity for a class-n streaming node. The playback or encoding bit rate of the video is denoted by r (also in bits/second). In keeping with the naming convention of [89], starter peers are referred to as *super peers* (in contrast, super peers are *not* necessarily starter peers) and regular peers are *ordinary peers*.

The analysis follows the experimental setup (where a typical deployment scenario is established). All super peers (both starter peers and converted peers) have identical upload capacity u_1 (class-1 peers) while ordinary peers (class-2 peers) all upload at u_2 with $u_2 < u_1$. This assumption is made to simplify the analysis and is in line with expected characteristics of a typical network whereby the *average* upload capacity will be roughly equivalent for all peers in a given class.

By assumption, $u_2 < r$ since if $u_2 \geq r$ then the peers can simply be connected in a serial streaming system where each viewer simply relays the stream.

7.3.3.3 Universal streaming

Super peers join the system with a Poisson distribution $P_1(t)$ and ordinary peers join the system with a Poisson distribution $P_2(t)$ with $P_n(t)$ representing the number of class-n peers in the system at time t. $P_1(t), P_2(t)$ are $M/G/\infty$ processes meaning that arrival time is exponentially distributed, viewing time is subject to any general probability distribution and that within the queuing system, there are an infinite number of servers.

For a bufferless system, *universal streaming* implies that for a system where peers receive data at a rate r they can play back this video data at the same rate r (essentially, viewers receive and decode all frames with no delay). The ability to achieve universal streaming is sensitive to the number of nodes in each class in the system and is thus a function of $P_1(t)$ and $P_2(t)$.

7.3.3.4 Streaming capacity

The maximum delivery rate at which the system can deliver fresh content to each of the peers is denoted by $\Phi(P_1(t), P_2(t))$. The $\Phi(\bullet,\bullet)$ fluid function depends on the distribution efficiency of the designed peer-to-peer pull-driven streaming system. At a time t, universal streaming is possible if and only if $\Phi(P_1(t), P_2(t)) \geq r$.

By definition, the aggregate bit rate out of the media server cannot exceed u_s while the aggregate outgoing bit rate for a peer cannot exceed u_i for $i = 1, ..., n$ peers.

The *maximum achievable rate* r_{max} is the maximum bitrate encoding value r for a universal streaming system based on u_s and u_i. A value for r_{max} is determined using Equation 7.6 [89]:

$$r_{\text{max}} = \min\{u_s, \frac{u_s + \sum_{i=1}^{n} u_i}{n}\} \text{ and } r_{\text{max}} \leq u_s \tag{7.6}$$

The maximum streaming rate to an individual peer is bounded by $[u_s + u(\rho)]/n$ since the maximum aggregate bit rate for data flowing out of the media server and the entire swarm of n peers is limited to $[u_s + u(\rho)]$ distributed amongst the n peers. $u(\rho)$ is a measure of the total upload rate for *all* peers (Equation 7.7):

$$u(\rho) = \sum_{i=1}^{n} u_i \tag{7.7}$$

For a finite download rate d_i for peer i, $d_{\text{min}} = \min\{d_i : i = 1, ..., n\}$. In this case, r_{max} is determined using Equation 7.8. Thus, r_{max} is the minimum of: the media server streaming rate, the maximum individual streaming rate to each peer or the minimum download rate for any peer on the network.

$$r_{\text{max}} = \min\{u_s, \frac{u_s + u(\rho)}{n}, d_{\text{min}}\} \tag{7.8}$$

The $\Phi(\bullet,\bullet)$ fluid function for a two class system (where n_1 is the number of super peers and n_2 the number of regular peers) is described below. For $u_2 < r < u_1$, universal streaming is possible for a fluid distribution scheme that meets the condition in Equation 7.9:

$$r \leq \Phi(n_1, n_2) \tag{7.9}$$

This is based on the $\Phi(\bullet,\bullet)$ fluid function in Equation 7.10 [89]:

$$\Phi(n_1, n_2) = \min\{u_s, \frac{u_s + n_1 u_1 + n_2 u_2}{n_1 + n_2}\} \tag{7.10}$$

The probability of universal streaming (in the steady state) can be calculated as follows. P_i

denotes a random variable that represents the number of active class-i peers in the steady state. For P_i having a Poisson distribution with mean $E[P_i] = \rho_i$:

$$P(\text{universal streaming}) = P(P_i \geq cP_2 - u_s^{'})$$
$$\text{where} \tag{7.11}$$
$$c = \frac{r - u_2}{u_1 - r} \text{ and } u_s^{'} = \frac{u_s}{u_1 - r}$$

7.3.3.5 Critical region

Using the results of [89], the peer-to-peer streaming system operates in the *critical region* when $\rho_1/\rho_2 \approx c$. It is important to note that as the values for ρ_1 and ρ_2 become larger, the acceptable range for the *critical region* becomes wider. A peer-to-peer streaming system can operate in either the *critical region* or either of the under-capacity region ($\rho_1/\rho_2 < c + \epsilon$) or the over-capacity region ($\rho_1/\rho_2 > c + \epsilon$) where ϵ is dependent on the size of the system (ϵ is smaller for a larger system).

System performance is optimal in the over-capacity region whereas universal streaming is unlikely in the under-capacity region. Performance is good in the critical region subject to implementation of buffering and a suitable delay between receiving stream packets and playback of the stream.

A simple equation can be developed to determine how many peers are required in the *super* peer class to fulfil the overall bandwidth needs of the streaming system as a whole. The total required bandwidth to support streaming of the full video bit rate (r) to each and every peer (universal streaming) comprises regular peer upload bandwidth (p_{uprate}) combined with the upload bandwidth for *super* peers (s_{uprate}). The objective of this equation (Equation 7.12) is to determine the minimum number of super peers (s_{total}) required (for p_{total} regular peers) to provide universal streaming. Derivation of this equation can be found in Appendix B.

$$s_{\text{total}} \geq \frac{(p_{\text{total}} * r) - (p_{\text{total}} * p_{\text{uprate}})}{s_{\text{uprate}}} \tag{7.12}$$

In order to analyse the performance of the peer-to-peer part distribution algorithm, the overall probability of receiving the full stream needs to be calculated. Effectively, this implies multiplying the probabilities of receiving 100% of requested parts (before the part is required

for playback) from all the source peers (Equation 7.13):

$$P(\text{stream}) = P(s_1) * P(s_2) * ... * P(s_n)$$

where (7.13)

$P(s_i) =$ probability of receiving 100% of requested parts before they are required for playback and $P \in [0:1]$

7.3.3.6 Minimum distribution time

The minimum distribution time is a concept that is normally relevant in analysing a peer-to-peer file sharing system as this is the minimum amount of time required to transfer a full copy of a specific file. In the context of peer-assisted streaming, minimum distribution time can be used to verify that chunks can be transferred within a specific time window such as to avoid lost segments of video. For real time streaming, the minimum distribution time (T_{min}) should be equal to or lower than the playback time (P_{min}) for a specific chunk size (C_{size}).

The ordered nature of part delivery in peer-to-peer streaming implies that minimum distribution time needs to be lower than a prior known threshold established by the streaming or encoding rate for the video stream. The minimum distribution time is determined based on a generalised equation from [90] and is reworked here (Equation 7.14) to provide a deterministic value based on the distribution time for a chunk C_y during streaming. The time period used for verification will be determined as $P_{\text{min}} = \frac{8*C_{\text{size}}}{r}$ for C_y a video chunk with size C_{size} bytes and r is the rate at which video is encoded in bits/sec.

$$T_{\text{min}} = \frac{C_{\text{size}}}{\min\{d_{\text{min}}, \frac{u(A)}{L}, u(S)\}}$$

where

$d_{\text{min}} =$ lowest download rate amongst all peer classes

$u(S) =$ highest rate that a set of peers can upload fresh bits (7.14)

$L =$ number of viewers in the network

$u(A) =$ total aggregate upload bandwidth of all peers in the network

7.3.3.7 Tree architecture

The implementation design is based on a variant of the $PTree^k$ peer-to-peer distribution architecture. The results of [91] show that when compared to Linear or $Tree^k$ architectures,

$PTree^k$ shows better performance as it is robust to node failures and makes the most optimal use of available upload bandwidth. In the $PTree^k$ architecture, a node concurrently receives k parts from k different peers in parallel at a rate $\frac{p_b}{k}$ (p_b = peer's total download bandwidth). Essentially, the chunk transfer speed is the minimum of $\frac{p_b}{k}$ and the lowest upload speed (s_b) from any of the source peers.

In this analytical model, every node is (in general) an interior node in one or more trees and a leaf node in another tree. Each tree includes all N peers. The number of levels in a tree is determined as $m = 1 + \lfloor log_k N \rfloor$ for a tree with outdegree k and the total *height* of the tree is determined as $\lfloor log_k N \rfloor$.

Each chunk C_y is distributed on a different tree T^k. A leaf peer P_x in tree T^k located $\lfloor log_k N \rfloor$ levels away from the root of the T^k tree starts receiving chunk C_y after a time period of $[\lfloor log_k N \rfloor * t_{\text{delay}}]$ seconds where $t_{\text{delay}} = \min\{\frac{C_{\text{size}} * k}{p_b}, \frac{C_{\text{size}}}{s_b}\}$. This allows for calculation of the lag time from part insertion into the network and reception by a peer at the far end of the streaming network.

7.3.4 Analysis of Peer-to-Peer concepts

This section briefly describes the various functions that are required or form part of a peer-to-peer streaming system. Basic equations are developed as a means to give rise to a mathematical model that describes how each aspect of the system functions.

7.3.4.1 Congestion Control Mechanism

Congestion control is not directly addressed by the developed peer-to-peer streaming system. However, the various elements discussed below have some form of congestion control as a side effect. For example, the part requesting system is designed to be sensitive to available bandwidth and will not flood the upstream bandwidth with part requests. Typically, a super peer will not send a full copy of the stream to any one peer.

The system is designed to work on the basis of multi-source multi-part streaming (*i.e.* using multiple trees in a $PTree^k$ model) such that congestion is managed in an $\frac{r}{k}$ way by spreading the bandwidth *load* across multiple sources for a stream with a bit rate of r being streamed from k source peers.

7.3.4.2 Packet Scheduling System

The packet scheduling algorithm is very important in a packet-based queuing system. Essentially, the packet scheduling rules manage packet flow, scheduling and ordering. The performance of the streaming system as a whole is highly dependent on the efficiency of packet scheduling. For this specific implementation, the packet scheduling system is based on a dynamic request-and-deliver model.

The number of packets in each request (q) is calculated using a known segment size a where $a = \{1, 2..5\}$ seconds and p_{size} represents the preselected data chunk size in bytes (Equation 7.15). The encoding rate is specified as r and the transfer bandwidth as s (Kbps). The time delay (t_{delay}) between requests for a certain peer can be calculated using Equation 7.16.

$$q = \lceil a * \frac{r}{8 * p_{\text{size}}} \rceil \tag{7.15}$$

$$t_{\text{delay}} = 0.9 * \frac{a * p_{\text{size}}}{s * 128} \tag{7.16}$$

7.3.4.3 Rate Allocation Algorithm

The rate allocation algorithm is static in the sense that video is encoded at a constant rate and that the encoding rate is not varied at any point in the streaming session. The encoding rate is calculated based on the performance characteristics of a typical 1024Kbps broadband Internet service. The rate is adjustable and in the system configuration described above, the rate is adjusted and set so that the lowest class peer is able to download the stream at the full streaming rate. The simple rate allocation system can be modelled as follows (Equation 7.17):

$$r_{\text{enc}} = 0.90 * \min\{db_{c1}\}\{..\}\{db_{cn}\}$$

where (7.17)

db_{cn} = download bandwidth for a class n peer

The value for r_{enc} is calculated and set in accordance with the conditions defined in Equation 7.8. Other factors also play a role in selecting the encoding rate. These factors include the minimum distribution time for the reference system as well as the performance characteristics of the tree architecture in the peer-to-peer streaming system.

7.3.4.4 Peer classification

As previously discussed, the system is designed around the concept of peer classification. This classification is crucial in determining the performance characteristics and viability of a streaming system. Peers are typically classified by the bandwidth that they are able to provide for sharing of received data. The most critical factor is the number of peers that are able to stream at a higher transfer rate than the overall stream playback rate. These peers contribute to the ability of the system as a whole to stream data to each and every peer or viewer.

When classifying peers, it is important to monitor the ratio of peers and available bandwidth such as to predict scenarios where adding peers could run the system into a degraded performance mode. This ratio is closely linked to the b_a quantity (total available bandwidth for the entire network for new peers as determined in Equation 7.19) and is calculated using Equation 7.18 as follows:

$$\text{for class } l \text{ peers, } L_{\text{ratio}} = \frac{c_l}{k_{\text{total}}}$$

where

$$\sum_{l=1}^{n} c_l = k_{\text{total}} \text{ for n classes} \tag{7.18}$$

with c_l = total number of peers in class l

and k_{total} = the total number of peers in the streaming network

7.3.4.5 Peer Selection

The criteria for peer selection is very simple and can be modelled quite easily. Peers are selected as sources based on a *first come, first serve and become a server* basis. In other words, each peer makes a certain number of slots available (c slots of x Kbps upload capacity). These slots are then handed out in a round robin way to all new peers. Thus, when a peer joins the network, the contribution of resources is roughly equal to $c * x$ Kbps. Slots are filled up sequentially and there is no unique selection mechanism for new peers.

7.3.4.6 Admission Control

The current design for the streaming system does not include any admission control apart from a basic check that sufficient resources exist to support a new peer. Based on known parameters for peer profiles, available resources can be calculated as using Equation 7.19:

$$b_a = \{\sum_{l=1}^{n} k_l(c_l * x_l)\} - \{\sum_{l=1}^{n} k_l\} * r$$

where:

b_a = total available bandwidth in Kbps for new streaming peers

l = peer class

k_l = total number of peers in this class (7.19)

n = total number of classes

c_l = number of slots for this peer class

x_l = total contributed upload bandwidth per upload slot for this peer class

r = video bit rate

If the available resources do not exceed the video stream's bitrate (*i.e.* the condition in Equation 7.20 is not met), all potential admission requests are denied by the tracking server until such a time that sufficient resources are available.

$$\text{if } b_a > (r - c_l * x_l) \quad (7.20)$$

then: *admit new peer in class* l

7.3.4.7 Buffering

For efficient streaming with no dropping of frames, two important conditions need to be adhered to. Firstly, the total cumulative download bandwidth capacity needs to be at least 10% higher than the video streaming bit rate and secondly there should be a small effective delay between joining the stream and initial playback. This allows for creation of a small buffer which mitigates the effect of delayed packets due to network congestion, delayed requests, slowly responding peers *etc*.

The total time that should elapse before playback begins determines the size of the buffer. Based on the characteristics of the design of the streaming system, a buffer of three requesting rounds should be sufficient to offset the effect of delayed packets and/or packet losses. The buffer size (in bytes) can be calculated as follows (Equation 7.21):

$$b_{\text{size}} = 3 * p_{\text{req}} * n_{\text{srcs}} * p_{\text{size}}$$

where

b_{size} = total buffer size

p_{req} = number of parts requested in each request round from each peer (7.21)

n_{srcs} = number of source peers

p_{size} = partsize

The buffer size is used in Equation 7.22 to calculate the total delay before playback begins:

$$b_{\text{delay}} = \frac{b_{\text{size}}}{d_b}$$

where (7.22)

d_b = total download bandwidth

The buffer efficiency (how much streaming data is buffered during b_{delay}) can be determined using Equations 7.23 and 7.24 or by multiplying the number of parts in the buffer by the number of viewable seconds per part.

$$b_{\text{play}} = \frac{8 * b_{\text{size}}}{r}$$

where (7.23)

b_{play} = length of stream stored in the buffer

$$\text{b}_{\text{eff}} = \frac{b_{\text{play}}}{b_{\text{delay}}} \tag{7.24}$$

In a typical streaming system such as that of the implementation, buffering can be represented as two disparate traffic queues: a server upload queue $[q_s(t)]$ and a peer playback queue $[q_p(t)]$ [89]. The server queue is a queue of data that is ready to be delivered to the peer-to-peer network. The peer queue represents incoming data that is buffered by a peer prior to playback.

If queueing methodology is used to represent the buffering action of a single peer, that peer's source peers are represented as the server. In this case (and to reduce complexity), the server will be defined as the actual media server and the peer will be any peer that is logically receiving data from the server as well as other peers. Thus, the server queue is filled at a rate r (the server's video encoding rate) for a stream encoded in real time. For a two class system, the total bandwidth available to a peer can be calculated using Equation 7.25:

$$\phi(t) = \min\{u_s, \frac{u_s + u_1 P_1(t) + u_2 P_2(t)}{P_1(t) + P_2(t)}\} \tag{7.25}$$

When the outgoing bandwidth from the server (*i.e.* available bandwidth to a peer) $\phi(t)$ is less than r, the server buffer volume increases by $[r - \phi(t)]$. $R_s(t)$ represents the rate at which new content is provided to the peer-to-peer network. Typically, $R_s(t) = r$. The peer buffer volume then also decreases at this rate. Content is transferred from the server's buffer to the peer's buffer when $\phi(t)$ becomes greater than r.

The playback buffer length at each peer is represented by b_{play}. When playback begins (and at any subsequent time), it is always true that $q_s(t) + q_p(t) = r * b_{\text{play}}$. The peer playback buffer $q_p(t)$ is always drained at a rate r which is the stream playback rate. The peer will experience skipping or break ups in the stream when $q_p(t) = 0$ or when $R_s(t) < r$.

7.3.4.8 Incentive mechanism

The design presented for the peer-to-peer streaming system does not allow a peer to choose how much upload bandwidth to contribute. However, as with any networking application or system, this can be controlled using a third party method on the user's side. In these instances, an incentive mechanism is necessary to prevent restrictive throttling. In line with the assumption that users will restrict bandwidth to some degree, two separate values are used to indicate upload bandwidth.

The first value (u) is a representation of the configured or theoretical maximum upload capacity. This value is static and is tied to the type of service that the subscriber subscribes to. The second value is a dynamic representation for upload bandwidth (w). This measure is calculated and updated periodically. The more frequently that it changes, the more frequently it is checked. w is compared to u on a regular basis to determine whether performance indicators need to be adjusted when calculating peer selection, buffering and admission control. By implication, this is a passive incentive mechanism whereby users are silently encouraged to participate.

If $w < 0.9 * u$ for any significant length of time, performance will be degraded to discourage reducing available upload bandwidth. The comparison is based on a running average such as to mitigate the effects of normal traffic variation. The values for w and u are normalised to typical values for high end subscriber bandwidth figures so that very high

bandwidth connections (*i.e.* universities, corporates *etc*) need not be penalised if they choose to not contribute a large amount of bandwidth. There is no reward system for contributing upload bandwidth as there is no added value beyond receiving a fluid stream.

7.3.4.9 Bandwidth availability

An overlay model for modelling of peer-to-peer network structures for mesh-based streaming is provided in [27]. Based on this overlay model where the source can be considered as level 0 and peers are organised into levels where level m consists of all peers that are exactly m hops away from the source, the bandwidth availability can be determined using Equation 7.26. By adjusting the calculation to account for known system characteristics, the following representations are derived for available bandwidth:

$$\text{inbw}_j \approx \frac{\sum_{h=0}^{k} \text{ib}_h}{k} \text{ for } k \text{ source peers in level } i \tag{7.26}$$

$$\text{outbw}_i \approx \frac{\sum_{g=0}^{h} \text{ob}_g}{h} \text{ for } h \text{ connected downstream level } j \text{ peers} \tag{7.27}$$

The values for inbw_j (Equation 7.26) and outbw_i (Equation 7.27) refer to inter-peer bandwidth and can be compared to determine if sufficient bandwidth is available on level i to serve a peer on level j and to determine whether bandwidth bottlenecks occur at the outgoing interfaces of source peers or at the incoming interface of a destination peer. For example, if $\text{inbw}_j > \text{outbw}_i$ then this implies that for this specific peer-to-peer connection, the peer on level i is not able to provide an appropriate stream to the peer on level j based on average values (*i.e.* level i is congested).

7.3.5 Numerical analysis

The numerical analysis is carried out using the series of equations described above. The media server upload rate is assumed to be independent and constant. The number of super peers (n_1) is constant at the start of streaming and all super peers connected at the start of streaming are called starter peers. A new variable is introduced to represent the maximum download rate for a node - d_i for a node in class i.

7.3.5.1 Encoding rate and video bit rate

Using typical numerical input, r_p is calculated as shown below using Equation 7.1:

$$r_p = f_{\text{size}} * f_{\text{rate}} = 4500 \text{ bytes/frame} * 25 \text{ frames/second} = 112500 \text{ bytes/second} \tag{7.28}$$

TABLE 7.1: Typical numerical values for peer count(s) and related upload speeds

Name	Min	Max
r	912 Kbps	
n_1	10	250
u_1	2048 Kbps	
d_1	4096 Kbps	
n_2	50	750
u_2	512 Kbps	
u_s	n_1 * d_1	
N	1	1000

r_e was determined by running a sample encode with the target encoding parameters and chosen codec. For typical input, an average value for r_e was observed as 876453 bytes/second (determined using the codec selected for testing). This particular encoding experiment was carried out using an MPEG-4 codec with default parameters on a Intel Xeon 2.8GHz dual core CPU. Thus, it can be seen that the video encoder meets the $r_e > r_p$ requirement for implementation on this high end processor. If a typical desktop processor was able to perform roughly 13% as fast, the requirement would still be met.

The video bit rate (r_p where $r_p \leq r_{\text{max}}$) is set using the criteria in Equation 7.8. r_{max} is determined in Equation 7.29 based on typical data from Table 7.1:

$$r_{\text{max}} = \min\{40960, \frac{40960 + (10)(2048) + (50)(512)}{(50 + 10)}, 1024\} = 1024 \tag{7.29}$$

Thus, $d_{\text{min}} = 1024$ is selected. This value is then downscaled to account for slight packet loss and overhead. Thus, $r_{\text{max}} = 0.90 * 1024 = 921$ which is rounded down to the nearest multiple of 16 ($r_{\text{max}} = 912$Kbps) to determine the final encoding bit rate.

The fluid function $\Phi(n_1, n_2)$ can be used to determine the conditions under which the encoding rate r_p allows universal streaming *i.e.* when $r_p \leq \Phi(n_1, n_2)$ (from Equation 7.9). With typical input applied to Equation 7.10:

$$\Phi(n_1, n_2) = \min\{40960, \frac{40960 + (10)(2048) + (50)(512)}{(50 + 10)}\} = 1450\text{Kbps} \tag{7.30}$$

7.3.5.2 Overhead bandwidth consumption

A typical analysis of the effect of overhead can be completed using known message sizes as inputs along with the known frequency of sending these messages. As there are a number of types of message in the designed protocol, only one variant is studied here as an example. Due to the regularity of part request messages, these messages were analysed.

Based on the designed protocol characteristics, m_{size} = 52 bytes/packet with sending rate m_{rate} = 15 packets/second and bw_{total} = 256Kbps for this regular peer. This ratio is calculated (in this example) for upstream bandwidth were a node is sending PARTREQ packets to source peers. Using Equation 7.2, o_{ratio} = 2.4% (Equation 7.31) of upstream bandwith for PARTREQ packets. This result indicates a relatively high percentage overhead as a direct result of the text-based protocol. Optimisation from text to binary would result in an order of magnitude reduction in bandwidth consumed by protocol messages.

$$o_{\text{ratio}} = \frac{52\,\text{bytes/packet} * 15\,\text{packets/second}}{32\,\text{KB/second}} = 0.024375 \tag{7.31}$$

7.3.5.3 Universal streaming

For simplicity, u_1 is constant as all starter/super peers are assumed to be homogeneous. The system is designed to support a typical regular peer and thus, regular peer upload rate u_2 is also specified as constant. Using typical values as in Table 7.1, numerical analysis (by adjusting n_2) shows that the size of first layer (or level) of regular peers can reach 50 before the condition for universal streaming is not met (Equation 7.9). This result can be seen in the graph in Figure 7.1. This result means that for ten starter peers, the maximum horizontal size of the uppermost layer of regular peers should ideally not exceed 50 in order to guarantee universal streaming (based on the selected system parameters).

Equation 7.11 can be used to calculate the probability of universal streaming based on numerical input:

$$c = \frac{912 - 512}{2048 - 912} = 0.352\,,\text{and}\,,u_s^{'} = \frac{9120}{2048 - 912} = 8.028 \tag{7.32}$$

Thus,

$$P(\text{universal streaming}) = P(P_1 \geq 0.352P_2 - 8.028) \tag{7.33}$$

Based on the configuration of the proposed system where P_1 is known and constant, the

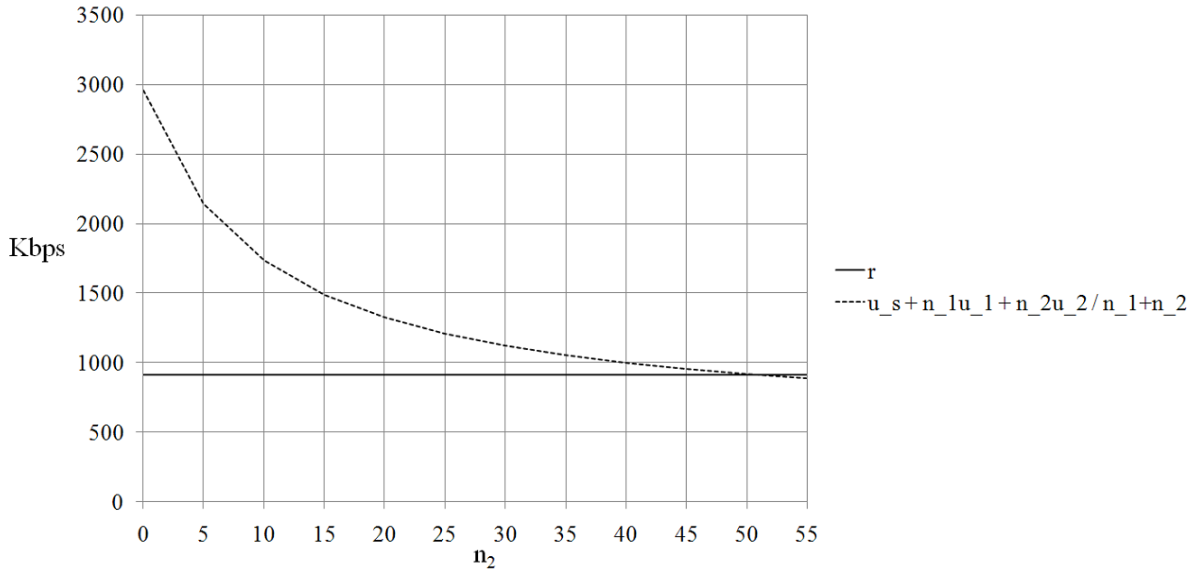


FIGURE 7.1: Visual representation of the point at which universal streaming becomes impossible (where n_2=50)

calculation is adapted as follows:

$$P(\text{universal streaming}) = P(P_2 \leq 2.841P_1 + 22.807) \tag{7.34}$$

With $P_1 = 10$, the probability of universal streaming using a single level of *ordinary class* peers is determined as $P(P_2 \leq 51.217)$ which implies that this result confirms the result determined above *i.e.* $n_2 = 50$.

If the same process is carried out for a network with 150 super peers:

$$P(\text{universal streaming}) = P(P_2 \leq (2.841 * 150) + 22.807) = P(P_2 \leq 448.957) \tag{7.35}$$

Thus, if universal streaming was to be guaranteed under all conditions, only 449 regular peers can be supported by 150 super peers. Figure 7.2 shows the required relationship between super peers and regular peers (for universal streaming) in making up the total number of peers.

The critical region is the region around the value for the critical threshold c. Given that $c = 0.352$, the critical region occurs where $\rho_1/\rho_2 \approx 0.352$. With $\rho_1 = E[P_1] = 10$, $\rho_2 = 28.4$. The calculation for the critical region gives rise to further criteria for determining the likelihood of universal streaming for the system. Universal streaming is likely if in general $\rho_1/\rho_2 > c + \epsilon$ *i.e.* $\rho_1/\rho_2 > 1.05c = 0.37$. Thus, there need to be at least $\rho_1 = 0.37\rho_2$ super peers (for ρ_2 regular peers) to achieve universal streaming within the system parameters described above.

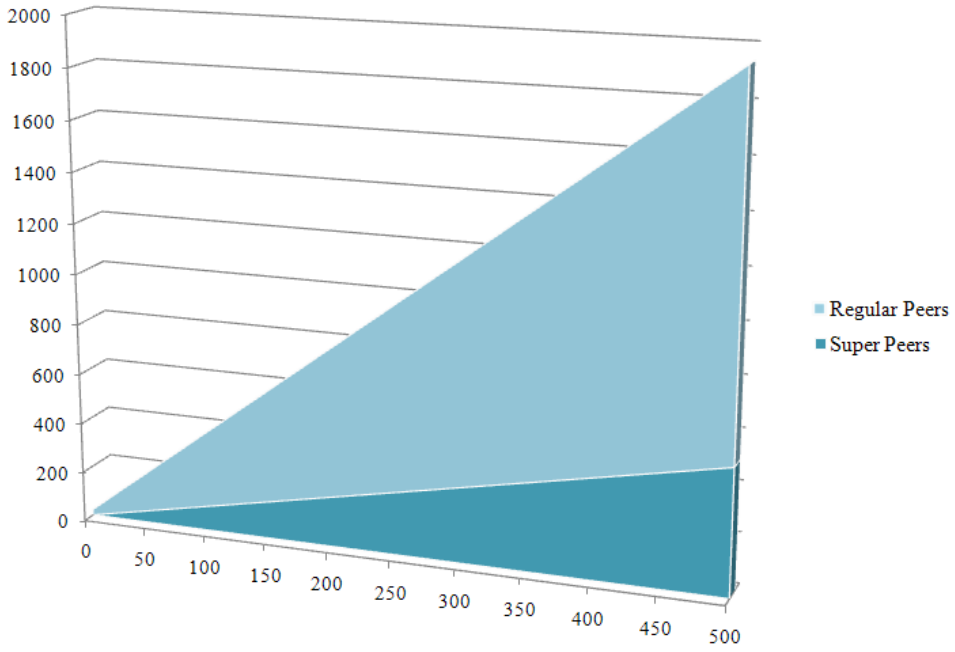


FIGURE 7.2: Stacked representation indicating the proportions of super peers and regular peers in composing the total number of peers (to maintain universal streaming)

This result implies that roughly 27% of all peers either need to be super peers or need to be able to be converted such as to act as a super peer (which is in line with the result depicted in Figure 7.2 where the super peer ratio tends to 26.03%). Based on this result and the system configuration, it can be stated that the average peer in the system needs to be able to stream at 116 KBps (926Kbps) in order to achieve universal streaming. Slightly reducing the encoding bit rate can result in the requirement dropping below 900Kbps.

Thus, to maintain a good streaming service, the network should be constructed of as many peers as possible capable of uploading at (or above) 900Kbps. Essentially, this implies that roughly one quarter of peers in the network need to be able to upload at a rate that is higher than the encoding bit rate (ideally significantly higher).

Equation 7.12 was developed as a second method of determining the minimum number of super peers required for a certain number of ordinary peers to achieve full universal streaming. Using typical values from Table 7.2, the minimum value for s_{total} is calculated as follows:

$$s_{\text{total}} \geq \frac{(p_{\text{total}} * 912) - (p_{\text{total}} * 512)}{2048} = 0.195 * p_{\text{total}} \tag{7.36}$$

TABLE 7.2: Typical numerical values for peer count(s) and related upload speeds

Name	Value
s_{uprate}	2048 Kbps
n_{total}	1..100000
v_{bitrate}	912 Kbps
p_{uprate}	512 Kbps

This implies a ratio of approximately 4:1 for regular:super peers and can be interpreted as every fifth peer needs to be a super peer in order to maintain universal streaming. This is roughly in line with the conclusion that 27% of all peers need to be super peers.

Equation 7.13 provides a simple deterministic calculation of the probability of receiving 100% of parts in a stream. As an example, if it is known that for four source peers, two source peers will be unreliable (based on past measurements) and only deliver 98.3% and 99.2% of packets in time, then the overall probability of receiving all packets on time can be calculated as:

$$P(\text{stream}) = 1 * 1 * 0.983 * 0.992 = 0.975 \tag{7.37}$$

7.3.5.4 Minimum Distribution Time and Tree performance

The minimum distribution time (MDT or T_{min}) can be determined using numeric input to Equation 7.14 as follows. As an example, the MDT is determined for a chunk size C_{size}=64KB on a network of 1000 viewers with a 25:75 ratio for super:regular peers and four source peers.

$$T_{\text{min}} = \frac{64\text{KB}}{\min\{128\text{ KBps}, \frac{750*64\text{KBps}+250*256\text{KBps}}{1000}, 1*256\text{KBps} + 3*64\text{KBps}\}} = \frac{64}{112} = 0.571\text{s} \tag{7.38}$$

In order to determine whether $T_{\text{min}} < P_{\text{min}}$, P_{min} is determined as follows:

$$P_{\text{min}} = \frac{8 * C_{\text{size}}}{r} = \frac{8 * 64\text{KB}}{912\text{Kbps}} = 0.561\text{s} \tag{7.39}$$

Based on this result it can be seen that in order to achieve real time streaming, adjustments are required. Typically, this implies increasing the number of super peers in relation to the number of regular peers. Alternatively, by decreasing the stream encoding rate, P_{min} is increased. If the encoding rate is decreased from 912Kbps to 896Kbps, the real time requirement ($T_{\text{min}} = P_{\text{min}}$) is met. Ideally, minimum distribution time needs to be lower than playback time (for a fixed chunk size) to ensure optimal real time performance.

Tree performance (or more specifically, $PTree^k$ performance) is analysed as follows. Each regular peer receives data chunks from k source peers (typically, k=4) at the maximum download rate p_b. Thus, the download rate from each source peer (or tree) is:

$$\frac{p_b}{k} = \frac{128\text{KBps}}{4} = 32\text{KBps} \tag{7.40}$$

This is the theoretical limit in terms of download rate from each source peer. A peer serving four downstream peers typically shares the upload capacity across these peers. In the scenario defined above, this implies that the upload speed (s_b) to each peer is limited to the total upload bandwidth distributed evenly across these peers. Thus:

$$s_b = \frac{512\text{Kbps}}{4} = 128\text{Kbps}\,(16\text{KB/sec}) \tag{7.41}$$

Based on an analytical model of the implementation network for 1000 peers, the total number of levels are determined as:

$$m = 1 + \lfloor log_4 1000 \rfloor = 5 \tag{7.42}$$

and the height of the tree is calculated as:

$$\lfloor log_4 1000 \rfloor = 4 \tag{7.43}$$

The delay between part insertion (at the source) and reception by a peer P_x can be determined. If P_x is located in a tree T^k and $m = 5$ levels away from the root, this peer starts receiving chunk C after approximately 8 seconds (Equation 7.44):

$$\lfloor log_4 1000 \rfloor * \min\{\frac{64\text{KB} * 4}{128\text{KBps}}, \frac{64\text{KB}}{16\text{KBps}}\} = 4 * 2\,\text{seconds} = 8\,\text{seconds} \tag{7.44}$$

7.3.5.5 Packet/Part scheduling

The packet scheduling characteristics can be examined by applying known quantities as follows (based on Equation 7.15). A number of packets (q) are requested in each round of requests. A certain number of seconds (a) are requested from each source peer. For example, if 1 second is requested from each source, q = 15 (assuming 8192 bytes are requested in each packet - Equation 7.45). Thus, 15 PARTREQ packets are distributed across the source peers per request round.

$$q = \lceil 1 * \frac{933888}{8 * 8192} \rceil = 15 \tag{7.45}$$

The time between request rounds is calculated (Equation 7.16) for a typical standard peer as follows:

$$t_{\text{delay}} = 0.9 * \frac{1 * 8192}{128 * 128} = 0.45\,\text{seconds} \tag{7.46}$$

7.3.5.6 Admission criteria

The level of available resources (b_a) for new joining peers can be determined using Equation 7.19. An example system is proposed with two peer classes (super peers with ten slots of 192Kbps each and regular peers with three slots of 160Kbps per slot). The super peer class is made up of 418 peers whereas the regular peer class has 913 peers.

$$\begin{aligned} b_a &= \{418 * (10 * 192)\} + \{913 * (3 * 160)\} - \{(418 + 913) * 912\} \\ &= 802560 + 438240 - 1213872 \\ &= 26928\text{Kbps} \end{aligned} \tag{7.47}$$

This result means that a total combined bandwidth of 26928 Kbps is available for new joining peers. In theory, if only regular peers were to connect, a further 62 peers could be supported:

$$\frac{26928}{912 - 480} = 62.33 \tag{7.48}$$

7.3.5.7 Buffering

Based on the above system, typical buffer size is calculated using Equation 7.21:

$$\text{b}_{\text{size}} = 3 * 4 * 3 * 8192 = 294912\,\text{bytes} \tag{7.49}$$

where a regular peer requests four 8192-byte parts from three source peers and three request rounds of data are buffered. The time that elapses before playback begins is equal to the transfer time for the above buffer size *i.e.* $\frac{288\text{KB}}{128\text{KBps}}$ = 2.25 seconds. Using Equation 7.24, buffering efficiency is calculated as:

$$\text{b}_{\text{eff}} = (8 * 294912)/(912000 * 2.3) = 1.125 \tag{7.50}$$

implying a 12.5% efficiency. This means that the buffer will on average contain enough data to deal with a 12.5% delay in the stream (dropped or delayed packets) if necessary.

For the system configuration describe above, buffering can be modelled as follows:

$$\begin{aligned} \phi(t) &= \min\{12500, \frac{12500 + (256\text{KBps})(250) + (64\text{KBps})(750)}{250 + 750}\} \\ &= \min\{12500, \frac{124500}{1000}\} \\ &= 124.5\text{KBps}\,(\text{or}\,996\text{Kbps}) \end{aligned} \tag{7.51}$$

The available bandwidth $\phi(t) = 996$ Kbps (Equation 7.51).

This value is compared to the typical value for r ($r \approx 900$Kbps). Thus, on average, the peer's buffer is able to fill at a rate $\phi(t) - r = 996 - 900 = 96$Kbps. Since $\phi(t) > r$, no buffering is required. This is based on the assumption that the peer in question joined the stream after a period of time and not at the start of the streaming session.

However, it cannot be assumed that all peers will receive an equal share *i.e.* the average value for available bandwidth to each peer ($\phi(t)$). Buffering also compensates for periods where a peer's available bandwidth were to drop for short periods (such as when another application makes use of the Internet connection or during periods of congestion). In this case, the peer's buffer will be emptied at a rate $d - r$ where d is the peer's actual download rate and $d < r$. This is assumed to only occur for short periods of time.

7.3.6 Experiments performed

Various implementation experiments were performed while adjusting the number of starter peers and the connection limit for regular peers. Each peer was set to upload at 512Kbps while downloading at 1024Kbps. This implies that each peer can supply data at 50% of the maximum receive rate. Therefore, each peer sources the stream from at least two upstream peers. The peer downstream connection limit was set at three. For the simulations, each peer received streaming parts from three sources. The simulation environment is fairly basic in comparison to a typical deployment scenario. A typical deployment scenario is subject to a large number of variable factors and is thus near impossible to model accurately under all conditions in a simulated testing environment.

In order to determine if the system would be functional as a whole, it is important to test each component individually (if possible). Results for these *standalone* experiments are included below. Video codec selection was initially based on theoretical evaluation. However, practical benchmarking is a more optimal method for selecting the implementation codec as visual encoding quality could be determined. The most suitable codecs were benchmarked as described in the following sub-section.

Subsequent to testing each component and sub-section of the implementation, the entire system was tested in a typical operational scenario. A streaming network was set up and the bandwidth performance was tested. Bandwidth consumption for all components was measured. Comparisons between unicast streaming and the performance of the proposed

solution aim to indicate the level of efficiency achieved using peer-to-peer as an alternative to unicast streaming.

7.3.6.1 Video codec comparison

The streaming system is dependent on a good video codec choice. The metric for a good video codec is typically video quality and compression speed. The video codec comparison was performed using the most popular and most portable video codecs. A high definition sample clip with a relatively high bitrate and resolution was encoded in one pass with each individual codec thereby simulating live stream encoding.

The encoding speed was recorded for each codec by measuring the overall time to encode the clip. The PSNR for each frame was recorded and averaged as a means to compare the quality of each compression scheme.

The test was carried out using a video bitrate of 900Kbps and an audio bitrate of 128Kbps. Output frame size was set as the DVD frame size (720x576). GOP size was set as 150 frames. For accurate comparison, a new quality measure is proposed. This quality ratio is calculated as encoding bitrate divided by average PSNR. This ratio compensates for the improved PSNR obtained by using more bits per frame during compression.

7.3.6.2 Unicast layer bandwidth consumption

The main objective of this simulation was to determine the real throughput or amount of consumed bandwidth for the upstream connection of the media server when streaming data to a certain number of starter nodes. A secondary objective was to estimate (based on reported performance data) the overhead for streaming. The output of these simulations is an accurate bandwidth consumption value which is a critical network design parameter when establishing the streaming network.

The experiment was performed by streaming a typical clip using the media server developed as part of the proposed implementation. The starter nodes connect to the media server and receive the stream using unicast streaming. In order to simulate practical deployment network links, the incoming bandwidth to each starter node was shaped to 4096Kbps as explained earlier. The media server's uplink bandwidth capacity was shaped to 100Mbps.

The test was performed using two, four and six starter peers with bandwidth consumption measured on the outgoing interface of the media server.

7.3.6.3 Peer-to-peer connection

The peer-to-peer architecture relies on an initial setup protocol. The tracking server implements this designed protocol and manages the construction of the peer-to-peer network. The simulation of the establishment of a peer-to-peer network involves all the messages as specified in the communication protocol. As each peer connects, the tracking server provides that peer with the addresses of three other source peers. The network structure is maintained in a table which is updated by the tracking server as each peer connects. The tracking server also maintains the status of each peer (with respect to available upstream connections) as the peer-to-peer network expands.

7.3.6.4 Peer-to-peer network bandwidth consumption

The objective of utilising a peer-to-peer architecture for high volume streaming is to spread the bandwidth distribution over the entire network (in contrast to a single link or point consuming all available bandwidth). During simulations using the proposed implementation, bandwidth consumption graphs for all components of the network were recorded.

The aim of this experiment was to show that the media server outgoing bandwidth remains constant as peers are added to the network. During this simulation, peers were added to a network (similar to the network shown in Figure 7.16) of two starter nodes and ten regular peers. Figure 7.3 shows the outgoing bandwidth of the media server during this streaming session. It can be seen from Figure 7.3 that the outgoing bandwidth is limited to roughly 1MBps (8Mbps) for two starter peers. This is consistent with the experimental setup where each starter node's incoming bandwidth is limited to 4096Kbps (or approximately 500KBps).

7.3.6.5 Experimental observations

The experiments performed led to a number of observations. Due to the speed of the system running the media server, incoming video data can be encoded faster (up to 300%) than real-time. This results in 100% data availability to starter peers. The result of this is that if starter peer download bandwidth is unconstrained, video data is received at the full encoding speed. This was measured and recorded using the reporting features of the simulation environment.

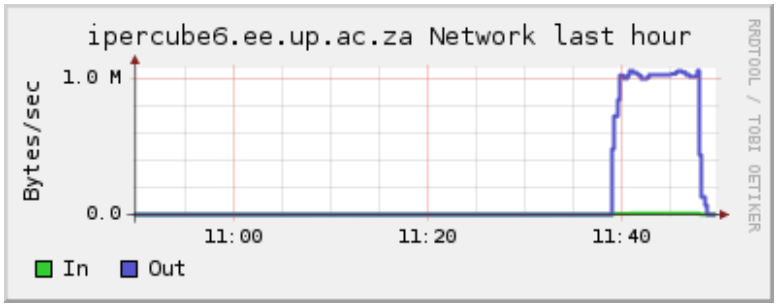


FIGURE 7.3: Media server outgoing bandwidth during a peer-to-peer streaming simulation for two starter peers (regular peers joining the stream do not affect the media server bandwidth)

The bandwidth consumption (downstream) for a single unconstrained starter peer is shown in Figure 7.4. Figure 7.5 shows the upstream overhead during the unicast stream. Comparing these two graphs allows for an approximation of the overhead for a single starter peer.

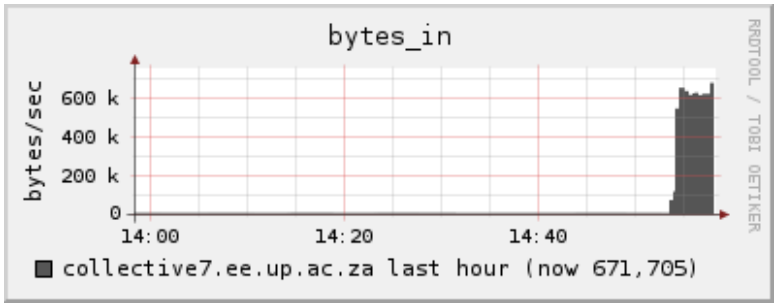


FIGURE 7.4: Unconstrained unicast stream download rate

Based on an 863Kbps stream, the average download rate (and thus, stream encode rate) for an unconstrained starter peer is approximately 610KBps (5.65x the streaming bit rate) and the average upstream overhead is 30KBps for this particular test. This result indicates that overhead bandwidth consumption increases in proportion with the overall transfer rate. Based on the overhead measurement, upstream overhead per starter node can be assumed to be 43Kbps if data is streamed at the encoding bit rate. This value should to be added to the outgoing bandwidth calculation for each starter node when selecting the number of downstream peers that can be supported.

The video codec comparison highlighted the extreme processing complexity of H.264 compared to the other equivalent codecs. The encoding of the sample clip took 208-250% longer when compared to the other codecs in the benchmark. This is offset by a compression quality improvement of at least 7-10% based on average PSNR comparison alone.

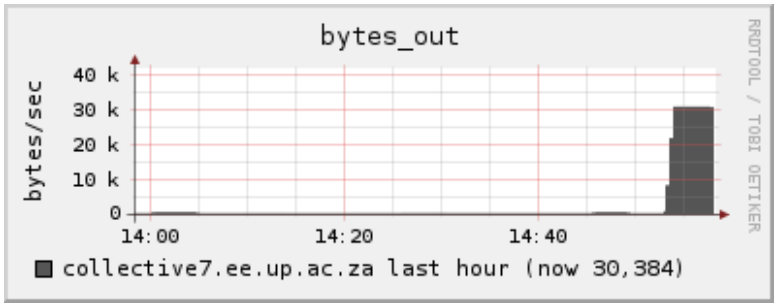


FIGURE 7.5: Unicast receiver upstream overhead

7.3.7 Verification procedures

Verification procedures were used at every step in the development process. These ranged from simple visual output verification to more formal hash (MD5) based data integrity checking.

During development and testing, part delivery was tested by hashing input parts and recording the 128-bit hashes to file. It is trivial to send these hashes along with the data. However, in this proposed implementation MD5 hashes are unnecessary and only increase the total overhead throughput since there is no requirement for integrity checking during streaming (re-requesting corrupted parts is normally meaningless in a live stream). The MD5 hashing outputs can be chained to check blocks of data with a uniform size (*i.e.* 1 minute blocks or every 5MB during streaming) if required. An example of the stored MD5 hashes files is provided in the appendices.

The peer-to-peer algorithm was verified by examining received packets generated by the tracking server (containing source addresses) as well the the network tables maintained by the tracking server. Protocol messages were recorded and packets logged to file. These log files were used to verify communications between entities and to verify that the peer-to-peer interactions were operating correctly.

The bandwidth performance of the network was determined using the bandwidth reporting mechanisms of the simulation environment. Bandwidth consumed by each entity in the system was measured and compared with expectations. The main objective of implementing peer-to-peer (distributing the bandwidth load) was verified by recording and comparing bandwidth consumption graphs for the peer-to-peer and unicast layers.

7.4 SIMULATION RESULTS

Results obtained by simulation and benchmarking are included in this section. Typical operating performance measurements are obtained and summarised.

7.4.1 Video encoding benchmarks

The following codecs were benchmarked: H.263, H.263+ (H.263v2), H.264, MPEG-2, MPEG-4, FLV and WMV2. Results of the encoding tests are summarised in a series of graphs. Figure 7.6 compares the encoding time of the sample clip for the benchmarked video codecs. Default options were used (regarding profiles and levels) and resolution was kept constant (720x576) at the PAL frame rate (25.0 FPS). Compression quality measurements (based on

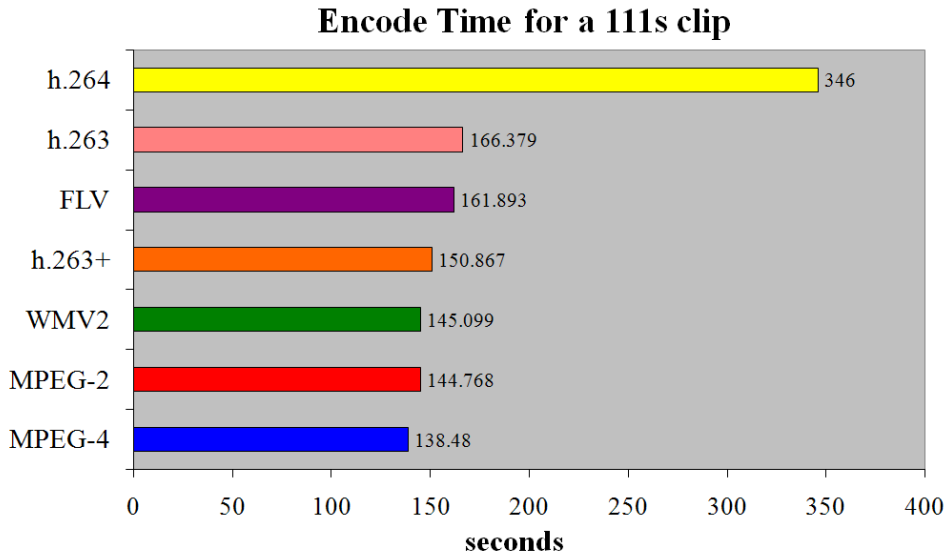


FIGURE 7.6: Encoding speed comparison for benchmarked video codecs (low end workstation with a high definition input video clip)

average PSNR) are contrasted in Figure 7.7. The proposed overall quality measurement as a scheme for directly comparing the actual compression quality of a codec at a specified bit rate was calculated using encoding results (average PSNR and real output bit rate). This quality ratio is used to accurately compare video codecs (Figure 7.8).

7.4.2 Communication protocol

All simulations depend on the communication protocol. Therefore, verification of the functionality and performance of the protocol is a critical step in the verification and simulation

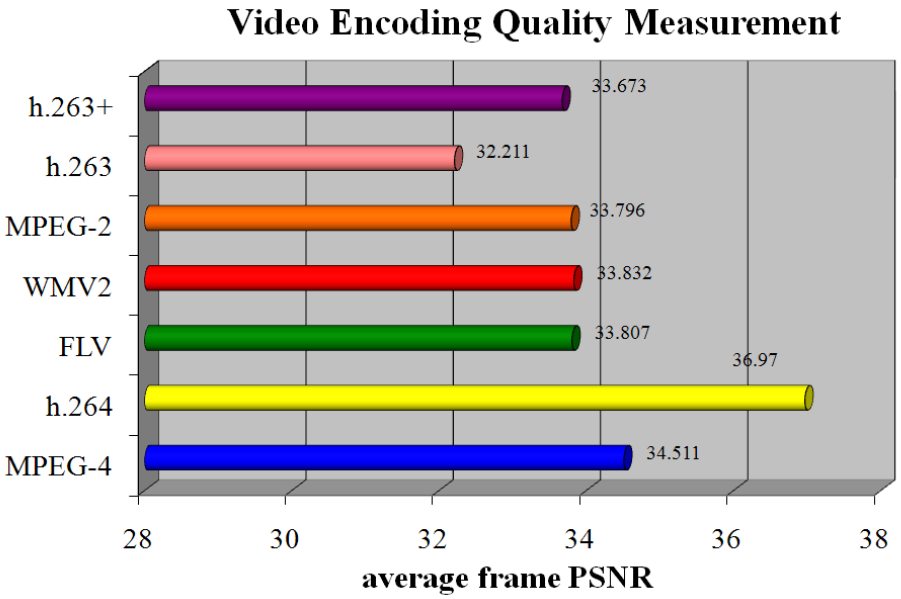


FIGURE 7.7: measurements using average PSNR

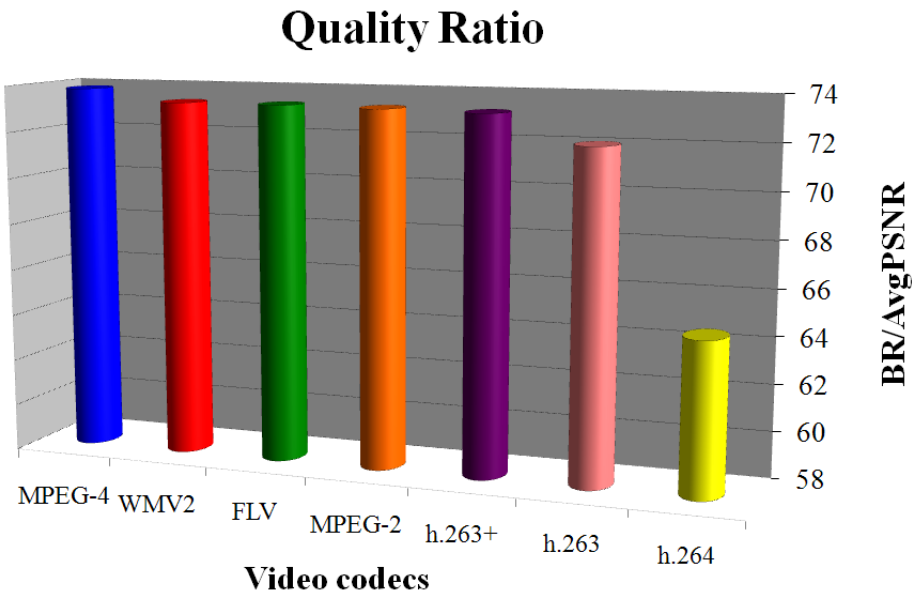


FIGURE 7.8: Quality ratio calculated as $\frac{\text{encoding bitrate}}{\text{avgPSNR}}$ for each codec

process. The proposed implementation includes packet logging within each component. Samples of packet logs are provided separately in the appendices of this dissertation.

7.4.2.1 Starter Peers

The following packet types were recorded: PARTREQ, PART. Each part request was logged and as each part was served, the part's identifier was logged to file. Table C.2 includes a

sample of the *requested.log* file. It can be seen that only every *n*-th packet was requested in this case. For simplicity, this packet sample was recorded for a small downstream network of three downstream peers (with two starter peers) and shows a brief practical overview of the part request and supply process.

In this configuration, only one part is requested and supplied at a time. Table C.3 includes a sampling of the sent packets (matching up with the requests above) from the *sent.log* file. Part requests and sent parts are marked using peer identifiers. To prevent flooding, PARTREQ packets are sent at no more than 10% faster than the playback rate. This part request rate is calculated using the stream bit rate and partsize. The part requesting module waits for k ms between each request where:

$$k = \frac{1000 * \text{Part Size}}{1.1 * \text{download rate}}. \tag{7.52}$$

7.4.2.2 Regular Peers

Corresponding log files for requested and received packets are provided in Table C.4 and Table C.5 respectively. These packets were generated during a simulation session with two source nodes and where only a single part is requested in each PARTREQ packet (thus, part requests are sent to alternate source nodes). These log files are associated with Table C.2 and Table C.3.

The stream is sent to downstream peers using PART packets. These packets contain a short preamble followed by a raw data chunk (sourced from the stream output file which is originally constructed from received PART packets) delimited by BEGIN and END strings. An example packet is provided in Table C.6 where the data is represented in hexadecimal format with each character pair representing a single byte of binary data. In this particular example, the part size is chosen as 256 bytes for demonstration purposes (typical simulation part size was between 8KB and 65KB).

7.4.2.3 Tracking Server

A number of packets recorded in a typical session are shown in Table C.7. In this particular session, there were two starter nodes and three regular peers.

The tracking server is the main entity in the implementation and only transfers control packets and no data whatsoever. For this reason, all data being transferred to and from the

tracking server can be considered a communication message or packet. A snapshot of the traffic graph for the tracking server shows data transfer made up of incoming and outgoing protocol packets exclusively. This bandwidth measurement graph is shown in Figure 7.9 and provides an approximation for the typical tracking server bandwidth usage in a normal streaming session. It can be seen from this graph that the bandwidth consumption is almost negligible as only a limited number of very small packets are transmitted.

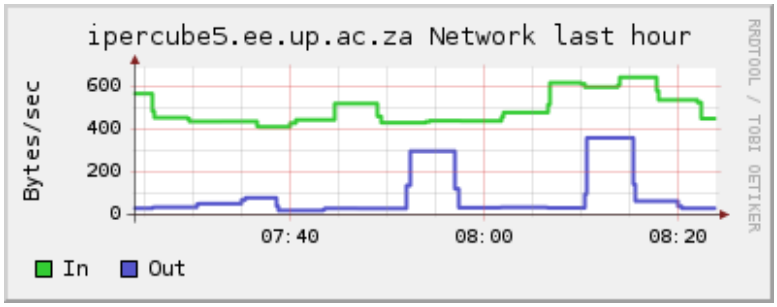


FIGURE 7.9: Incoming and outgoing bandwidth consumed by the Tracking server during a peer-to-peer video streaming simulation

This graph shows packets arriving from the point at which the tracking server was started. Communications from this point include the connection of the media server, starter nodes and regular peers (essentially establishment of both the unicast and peer-to-peer network layers).

7.4.2.4 Media Server

The communication protocol only specifies basic usage of messaging for the media server. Table C.8 includes a sample of the packet log file. The only packets received by the media server are once-off ESTABLISH packets sent by connecting starter peers.

7.4.3 Unicast performance

The simulation results for the unicast streaming section were obtained using the experimental setup described earlier. Graphs for each test are shown in the following sequence of figures. The last graph (Figure 7.14) shows the session stream download rate for a sample single starter node. It can be seen that data is received at approximately 512KBps. Based on the measured bandwidth consumption, a collective graph was generated to plot the growth of the overall consumption in relation to the increased number of starter peers. This graph is shown in Figure 7.15.

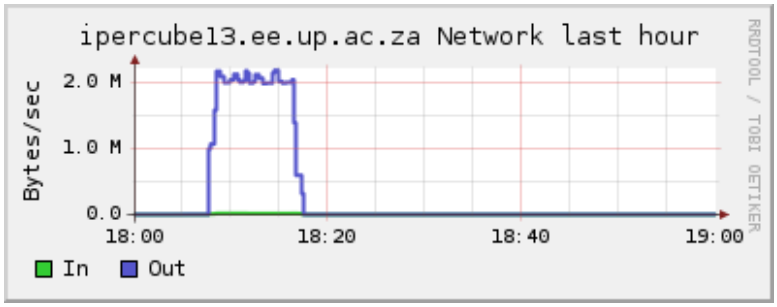


FIGURE 7.10: Media server total uplink bandwidth consumption (testing run at 18:07 for four starter peers)

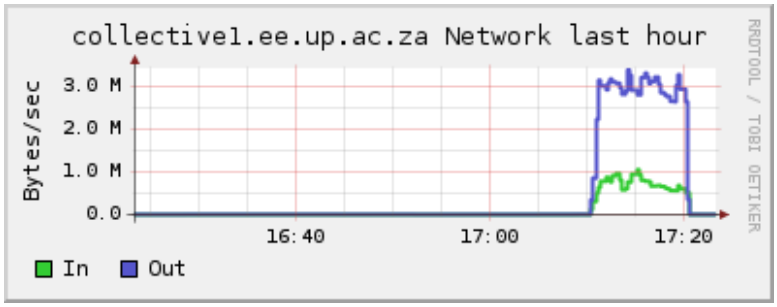


FIGURE 7.11: Media server total uplink bandwidth consumption (testing run at 17:10 for six starter peers)

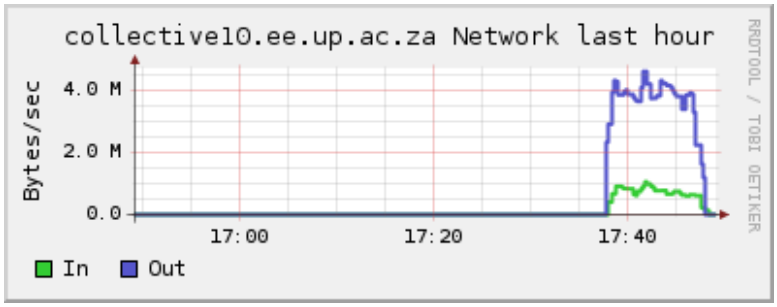


FIGURE 7.12: Media server total uplink bandwidth consumption (testing run at 17:37 for eight starter peers)

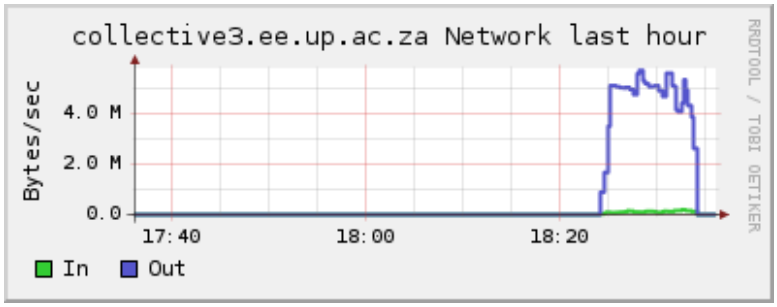


FIGURE 7.13: Media server total uplink bandwidth consumption (testing run at 18:24 for ten starter peers)

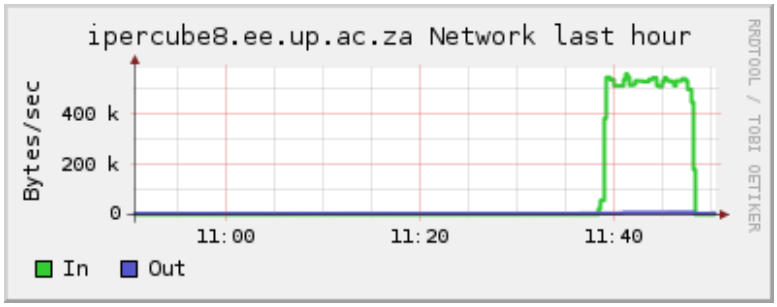


FIGURE 7.14: Sample starter node indicating that the stream is downloaded at the set downlink limit

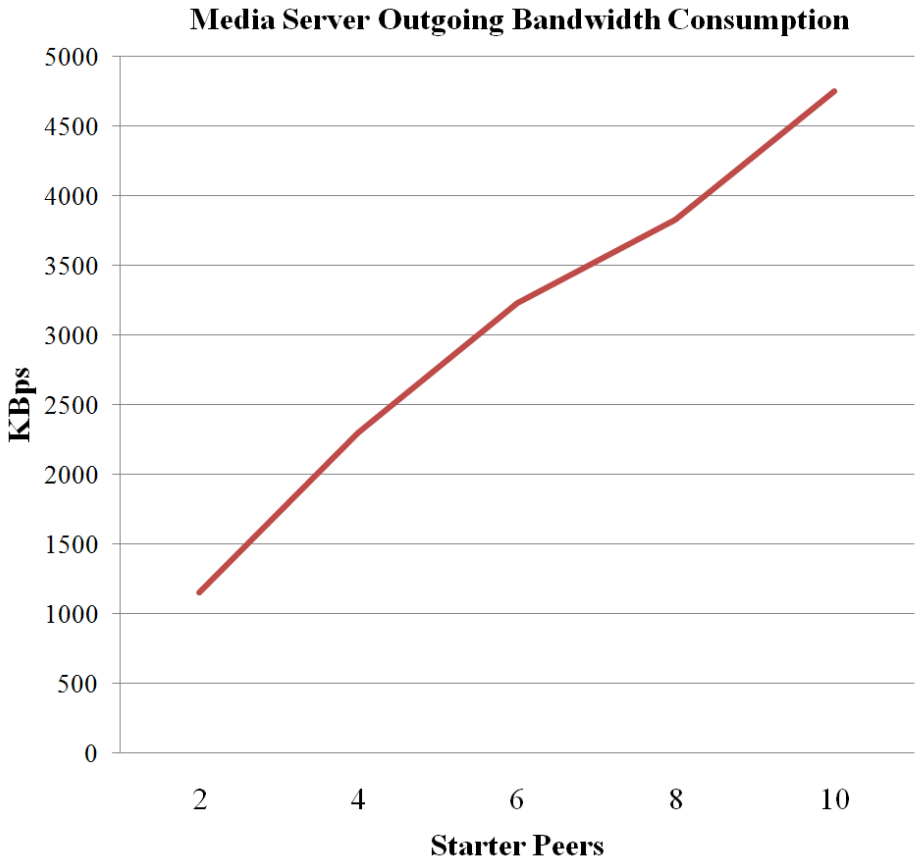


FIGURE 7.15: Media server total uplink bandwidth consumption increase relative to growth in number of connected starter nodes

The outgoing bandwidth consumption begins to taper off as the connected starter node count increases. This suggests that the upper limit for starter node connections is less than the theoretical limit of 25. This limit is not tested as it is not relevant at this stage.

7.4.4 Peer-to-peer performance

Peer-to-peer performance measurement is based on two main requirements. The initial requirement is that the designed peer-to-peer algorithm is adhered to. The second requirement is that the peer-to-peer network supplies data in such a way that the stream can be constructed in real time and is not obviously peer-to-peer (*i.e.* network agnostic) and thus functions as if it were a regular unicast-powered streaming system.

The peer-to-peer algorithm specifies how the network is constructed by the tracking server. The tracking server manages source assignment and builds a network table which is updated as each source is connected. With each new downstream source, the offered (or available) downstream connection counter is decremented from an intial preset limit. If this counter reaches zero, no more new connections are established.

The tracking server maintains a simple overview of the network structure based on the format in Table 7.3. The sources for each peer are recorded and stored in a single place. Using this data, a simple plot of the network layout can be generated (Figure 7.16). It can be seen that regular peers are provided with source addresses on a first-come first-serve basis resulting in the initial peers receiving the starter nodes as sources. The port numbers are chosen for convenience and have no other significance. Ports within the 7950-7960 range typically signify starter nodes whereas 7980-7989 represent regular peers.

The performance of the peer-to-peer network as a regular streaming network is difficult to measure qualitatively. Successful verification by means of packet delivery accounting implies good performance for streaming. Two verifications are used to gauge streaming performance: careful comparison of the received parts (by using the log files and MD5 data for integrity checking) with the parts that were requested as well as a secondary check where the streamed data size (file: *data.output*) can be compared with the source size (*data.output* for a peer source or *data.stream* for a starter node source).

7.4.5 Bandwidth consumption measurements

In addition to measurements made as described above, several measurements of bandwidth consumption were captured. The typical incoming stream for a regular peer/viewer with no downstream connections is shown in Figure 7.17. It can be seen that the incoming stream

Table 7.3: Sample network table for a typical simulation session with eleven regular peers being fed by three starter nodes

PEER	No.	Source1	Source2	Source3
137.215.153.62:7980	1	137.215.153.60:7950	137.215.153.61:7951	137.215.153.54:7952
137.215.153.63:7981	2	137.215.153.60:7950	137.215.153.61:7951	137.215.153.54:7952
137.215.153.64:7982	3	137.215.153.60:7950	137.215.153.61:7951	137.215.153.54:7952
137.215.153.65:7983	4	137.215.153.60:7950	137.215.153.61:7951	137.215.153.54:7952
137.215.153.66:7985	5	137.215.153.60:7950	137.215.153.61:7951	137.215.153.54:7952
137.215.153.67:7986	6	137.215.153.62:7980	137.215.153.63:7981	137.215.153.64:7982
137.215.153.68:7987	7	137.215.153.62:7980	137.215.153.63:7981	137.215.153.64:7982
137.215.153.55:7988	8	137.215.153.62:7980	137.215.153.63:7981	137.215.153.64:7982
137.215.153.56:7989	9	137.215.153.65:7983	137.215.153.66:7985	137.215.153.67:7986
137.215.153.57:7984	10	137.215.153.65:7983	137.215.153.66:7985	137.215.153.67:7986
137.215.153.58:7980	11	137.215.153.65:7983	137.215.153.66:7985	137.215.153.67:7986

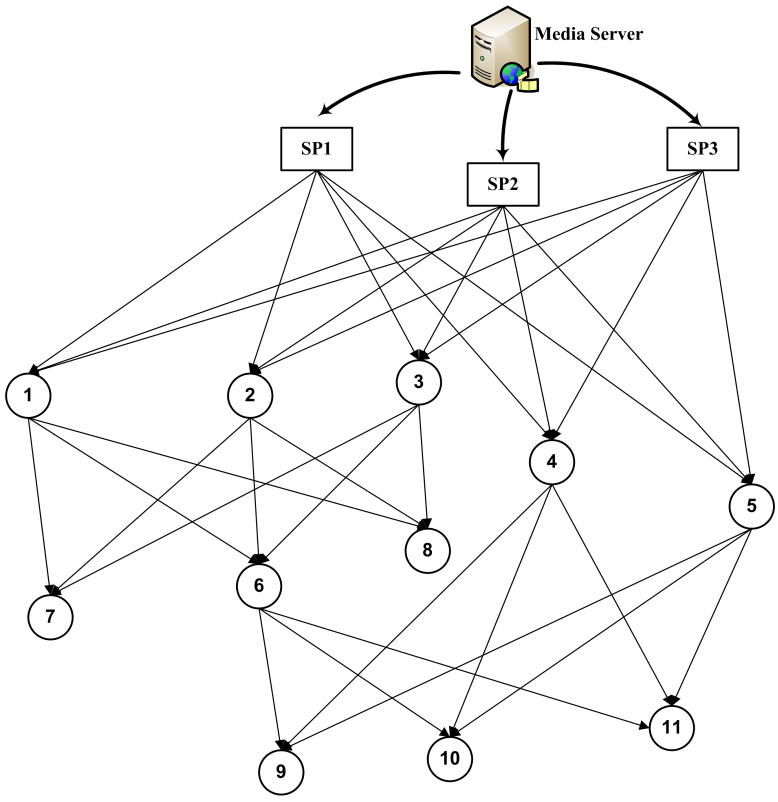


FIGURE 7.16: A sample simulation network layout (based on the data from Table 7.3)

consumes the full available bandwidth (1024Kbps or 128KBps). The typical configuration for a regular peer results in availability of roughly half of the incoming stream for outgoing traffic. Figure 7.18 shows the incoming stream (from upstream peers) along with the outgoing

streamed parts delivered to connected downstream peers.

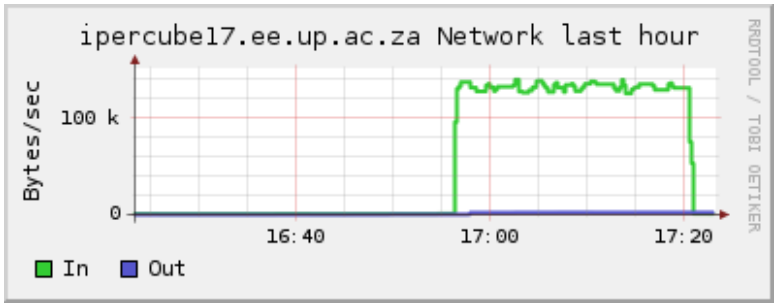


FIGURE 7.17: Typical incoming streaming bandwidth consumption for a regular peer with no downstream peer connections

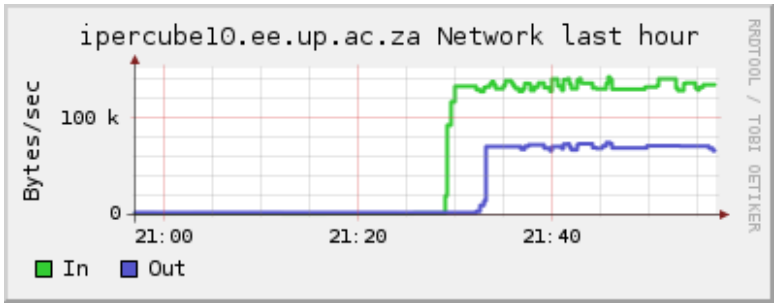


FIGURE 7.18: Typical bandwidth consumption for a regular peer that relays the stream to downstream peers

During normal streaming, a starter peer receives unicast data from the media server. This data is then supplied to connected downstream peers using the designed peer-to-peer protocol. Typical bandwidth performance for a normal starter peer during a streaming session is shown in Figure 7.19 where the received stream is being sent to 3 connected downstream peers.

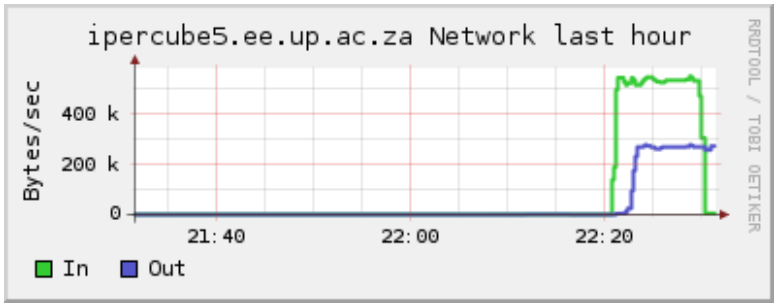


FIGURE 7.19: Incoming and outgoing bandwidth for a starter peer receiving the stream from the media server and relaying data to downstream regular peers

7.5 DISCUSSION OF RESULTS

This section compares obtained results with initial expectations. The methodology and rationale for the testing approach followed is discussed. A variety of problems were encountered during the development phases as well as during simulations and testing. These problems are briefly mentioned along with implemented solutions. Results obtained during video codec benchmarking, unicast streaming and peer-to-peer are independently summarised. Suitability as well as the reliability and repeatability of the obtained results is briefly discussed. The results obtained in simulation are briefly compared to outcomes derived from analysing the implementation in detail.

7.5.1 Expectations

The proposed design was developed with the sole purpose of determining whether peer-to-peer can be an appropriate platform on which to offer video streaming and that it can be advantageous to a media source server to do so. Based on the application of peer-to-peer principles along with a unicast layer (for horizontal scalability) the design is expected to offer bandwidth savings for the server side. Pre-development expectations included the following shortlist:

- Video data should be streamed reliably to every single node within the network
- The distribution of data happens in real time *i.e.* faster than playback rate
- A peer can join the network at any point and receive the stream from that point onwards
- The video encoding module should allow for replacement of the video codec if desired
- The system should be extensible and adaptable

7.5.2 Methodology of testing

The unicast layer was tested independently to verify that regular streaming was possible. This layer provides for a more stable and balanced distribution network. The bandwidth consumption used by the stream was measured at the outgoing interface of the media server (while varying the number of connected starter nodes).

As a means to verify that the peer-to-peer section does indeed offer benefit to the server in

the form of bandwidth savings, performance of the network with ten nodes (two being starter nodes, eight peer-to-peer) was directly compared with the bandwidth performance of the server for ten connected starter nodes. This can be seen as a direct comparison between the proposed implementation's peer-to-peer performance and the performance of a regular unicast steaming system. The comparison showed that on average, server-side bandwidth consumed was reduced by roughly 70-85% as expected (based on a 900Kbps video stream).

7.5.3 Developmental challenges and problematic aspects

A number of pertinent problems were encountered. A small selection of these problems within the related applications is briefly mentioned below. The method used to solve the problem is briefly mentioned.

Network blocking and file access issues resulted in a re-designed networking interface which lowered CPU usage (in waiting mode). In order to simplify and guarantee access to data, a buffering system (for packets and raw data) was implemented.

Connection limits and open file descriptor limits were reached quickly by the initial design. This required a reworking of the design of many parts of the system and addition of open connection reuse within the streaming network. Due to the complexity and high number of interactive parts, numerous software bugs were encountered. This required continual optimisation of the implementation source code. The eventual *proof-of-concept* implementation is made up of four separate applications and approximately 5500 lines of C code.

Transfer of data over the network was problematic due to TCP data transfer scheduling algorithms resulting in random numbers of bytes being transferred during each $send()$ or $recv()$ call. The complex nature of data transfer (due to the peer-to-peer part sharing algorithm) introduced significant design complexity in the implementation further delaying eventual simulation and testing.

7.5.4 Video encoding

The video encoding benchmarks indicate that while H.264 offers the highest encoding quality, it is not yet sufficiently optimised (using basic encoding parameters) for general real time

encoding (a requirement for streaming) on typical hardware. MPEG-4 Part 2 produces the highest quality at an acceptable encoding speed (when compared to its counterpart codecs).

7.5.5 Unicast streaming

The unicast streaming tests indicate that the media server will stream data to starter nodes at their maximum downlink speed. The simulations were performed using real world downlink and uplink limits for the starter nodes. Video data was streamed to starter nodes at 512KBps (for a 4096Kbps downlink connection). The media server typically uploaded approximately *n* x 512KBps for *n* connected starter nodes.

During a number of the tests, a significantly high amount of data was transferred in the opposite direction to the streamed data (*i.e.* in the downlink direction) such as in Figure 7.10. This data is a simulation anomaly due to a combination of factors including the effect of the traffic shaping algorithm and the fact that logging was active and log files were written on a network drive.

When the simulations were performed with ten starter nodes, the total overhead data (incoming to the media server) was approximately 130KBps (Figure 7.20). Effectively, each starter node contributes roughly 13KBps (overhead) to the incoming bandwidth for the media server.

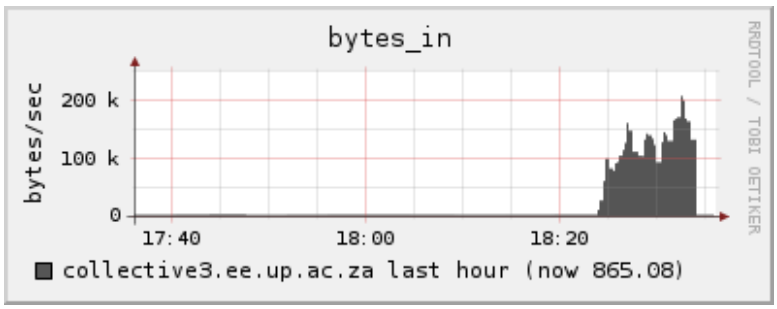


FIGURE 7.20: Total incoming data overhead for the media server with ten connected starter nodes (from the testing run as indicated by Figure 7.13)

7.5.6 Peer-to-peer

During development, each module was tested as it was developed. However, it was difficult to ascertain the performance of the peer-to-peer system as a whole until final simulations. While seemingly simple, the peer-to-peer architecture has a high implementation complexity.

Simulations showed that bandwidth was indeed saved in the server side (*i.e.* constant outgoing bandwidth for a growing connected peer count or audience).

7.5.7 Peer-to-peer video streaming

The simulations performed indicate that peer-to-peer video streaming in real time is possible. However, the challenging development, analysis and testing processes show why peer-to-peer networks for streaming have not been implemented on a large scale. The development of individual components as well as the management of interactions between entities within the network proved to be a challenge. The analysis and simulations show acceptable performance for a small to medium sized network.

As the network scales, performance can only be predicted by mathematical analysis. This analysis showed that the performance for a large network will meet expectations subject to some basic conditions. However, it must be noted that as the network grows, it becomes ever more susceptible to external influences and non-ideal interactions and occurrences within the network itself. These effects must be taken into account and tend to jeopardise the accuracy of predicted performance characteristics derived from numerical analysis.

The finite time required to add each peer can not be neglected. In addition, fairly expansive and disparate computing resources are required for effective testing (a high number of unique nodes). Automation of the peer additions adds additional complexity and was not part of the original design (as it was not considered as a relevant design consideration). For this reason, simulations were limited to a relatively small set of streaming peers.

7.5.8 Suitability

The suitability of the results is relatively high since the simulation environment models a typical deployment scenario fairly accurately. The achieved results do indeed show that implementing peer-to-peer can be significantly beneficial in terms of bandwidth savings. This is true when the network size (or scale) is relatively small. By extension, these results will apply to a large network and this can be confirmed by analysing the implementation using mathematical modelling and numerical prediction.

It was shown that a stream can be sent to at least ten starter peers with no inherent

problems. As the peer-to-peer network reaches capacity (when universal streaming becomes impossible), regular peers will need to be converted to super peers (if they have the necessary resources) or more starter peers connected to the media server in order to maintain universal streaming. When regular peers are converted to super peers, they will open more available slots to downstream peers effectively making more overall bandwidth available to the network.

By showing that a small set of starter peers can support a multi-layer network of peer-to-peer nodes, the simulation results imply that larger streaming networks can be constructed. The analysis of the implementation gives rise to criteria that can be used to determine the point where more super peers are required and also shows that if the necessary peer classification ratio is maintained, fluid streaming is possible for very large peer-to-peer networks.

Basic equations were developed to determine whether fundamental requirements for real time streaming were met. These equations were based on the minimum distribution and tree performance of the streaming system.

7.5.9 Reliability and repeatability

The unicast streaming performance of starter nodes was shown to be reliable. This was shown by establishing streaming networks based on two up to ten starter nodes. Essentially, adding more starter nodes would be possible but would subject the unicast network layer to the media server's outgoing bandwidth limit. With a unicast network size of ten nodes and a bandwidth limit of 100Mbps, it can be seen that (for the typical operating parameters of the stream) the performance limit would be reached at approximately 15-20 starter nodes. Therefore, reliability can be guaranteed for up to 15 starter nodes (100Mbps). If the server had full unconstrained bandwidth of 1Gbps (such as on a campus LAN), approximately 60 full speed super peers could be supported.

The repeatability and reliability of the experiment was confirmed by running the experiment under differing conditions with different network sizes. The network size was tested from a single basic starter node receiving the stream right up to a full ten starter nodes receiving the unicast stream. When testing the proposed implementation as a whole, the network was tested starting with a single regular peer-to-peer node and concluded with a test of the network architecture comprising multiple levels of peer-to-peer nodes. The wide variety of the tests improves repeatability of results and shows good reliability under the assumption that peers

are reliable. Simulation (and analysis to a large extent) does not account for peers failing or leaving the network.

7.5.10 Real world application

The proposed implementation is designed with the sole intent to act as a proof-of-concept for video streaming using peer-to-peer data transfer for distributing video. The implementation was developed to show that it is both possible and practical to use peer-to-peer data sharing for video streaming within the real time processing constraint.

Essentially, the results show that the implementation is appropriate for a real world environment. Some design adaptations are required to offer wider network support and support for variable bandwidth per peer (and per session). New performance data will need to be generated as the current simulation data would be invalidated if the purpose of simulation was to determine performance in a production environment. The proposed implementation is not a consumer-friendly application but can be adapted to become a *real world* application (subject to showing acceptable performance in a simulated environment with typical characteristics such as packet loss, network segment failures, random effects on latency, congestion *etc*).

7.5.11 Relation to analytical outcomes

The mathematical analysis of the system in parts and as a whole revealed some shortcomings of the initial design. This process resulted in some basic adaptations to the initial implementation design. The analytical approach followed introduced a measure for determining the likelihood of achieving universal streaming which implies maintaining a full video stream to each and every peer throughout the life of the streaming network (while the network grows).

Performance measures for each element were developed. These measures and equations can be used to estimate the performance characteristics for the implementation at any point in time or during any phase of growth. The analytical methods developed are intended to be used for general performance analysis and not specifically performance verification. The analysis is carried out as an addition to simulation.

Numerical analysis provides an insight into the configuration required to support streaming as

the network scales. Theoretical predictions can be made and compared with the simulation results (overhead bandwidth consumption is an example) as a method of verification.

7.6 SUMMARY

This chapter aggregated the results and outcomes of all analyses, simulations, experiments and tests performed. Basic analytical modelling produced a series of equations that are general enough to be used to evaluate specific characteristics of any peer-to-peer streaming system. In order to verify functionality, each component was thoroughly tested before incorporation into the final proposed system. The unicast layer was independently simulated. The video encoding system was benchmarked and the resultant deployment codec decision was made using data obtained from benchmarking. Finally, the peer-to-peer system was tested and showed good performance as a streaming platform. Simulation and analytic results show that the system performs as expected and meets design expectations. These results also give rise to pre-conditions and criteria for adjustments that are required to ensure full universal real time streaming.

CHAPTER EIGHT
CONCLUSION

8.1 SUMMARY OF THE RESEARCH

This body of research involved thoroughly investigating and studying the fields of video streaming and peer-to-peer networking leading up to a solution that harnesses both. The goal was to understand and evaluate each method of multimedia streaming such as to be able to determine how the eventual streaming system can be optimized within the peer-to-peer streaming paradigm.

Existing peercasting implementations were examined and contrasted based on operational capabilities, functionality, availability and other characteristics. The output of this process further defined the reference implementation.

A reference implementation was designed, developed and tested with the eventual goal being to show that peer-to-peer is a suitable platform for video streaming and that using this platform as a streaming deployment environment results in significant bandwidth and resource savings in the server side of a streaming system with no discernable impact on overall streaming performance from the viewer's perspective.

The implementation was developed and used in a simulation environment to derive performance characteristics based on a typical streaming scenario. Several suitable video codecs were benchmarked in order to determine the most optimal video codec for streaming (based on the real time encoding requirement).

Simulation results have shown that peer-to-peer is a highly suitable platform for large scale video streaming. A server can stream to a very large network of end users while maintaining near-constant upstream bandwidth and at the same time allow the network to scale.

8.2 RESEARCH OUTPUTS

The main outputs of the research are the following:

- conclusive investigation into available video encoding and streaming technologies
- detailed summary of the different peer-to-peer methods for constructing peer-centric networks
- summary of the available peercasting systems that are free to use
- detailed design for a proposed peercasting implementation (a proof-of-concept model)
- a basic performance evaluation environment and model
- analytical, simulation and testing results for the designed implementation

8.3 CONCLUSIONS

Streaming live video to a large number of endpoints is typically an expensive process in terms of bandwidth consumption. A system where any number of viewers can be supported is thus highly desirable. A peer-to-peer based video streaming system is an ideal solution to the problem.

The proposed implementation was designed to take advantage of the reliability of a unicast streaming system by making use of a layer of initial or starter peers. Beyond this layer, video data is streamed to end users via a peer-to-peer part sharing algorithm. These end users contribute their unused upload bandwidth to the network by redistributing video parts (or chunks) that they have received.

Simulations have shown that peer-to-peer video streaming is both possible and highly effective in distributing the bandwidth load across the broadcast network. This has been shown to be achievable in software with no hardware changes (such as with IP multicast).

8.4 SUGGESTIONS FOR FUTURE WORK

This implementation lends itself to intense optimisation in a number of areas. Topics related to the work include optimal partsize derivation using experimental analysis, performance of the

network in very low and very high bandwidth scenarios and adaptations for good performance in specific networks (for example, adapting to high packet loss).

Most related topics tend to address one of the above-mentioned areas. Several other ideas can also be explored such as implementation of the peer-to-peer algorithm for file transfer and dissemination. Future work can examine issues such as supporting a video-on-demand streaming system (by using non-viewing *transit* peers) using a peer-to-peer network.

Some specific areas for future extensions include:

- optimised part delivery scheduling and fairness schemes for honouring part requests
- comparison and improvement of the peer addition algorithm in order to achieve continuous network balance
- robust container and video data packaging design to support heavy data loss and playback (using skipping or FEC techniques) for undelivered video parts
- implementation of communications protocol optimisations and compression of protocol packets
- optimisation of the data chunk size specifically for video streaming
- additional support for sub-channels (using the same or separate distribution paths) including layer-encoded video, subtitles and audio streams for alternative languages

REFERENCES

[1] S. Alstrup and T. Rauhe, "Introducing Octoshape - a new technology for large-scale streaming over the Internet," European Broadcasting Union, Tech. Rep. 303, July 2005. [Online]. Available: http://www.ebu.ch/trev_303-octoshape.pdf (Last accessed: February 2007).

[2] R. Schollmeier and G. Schollmeier, "Why peer-to-peer (P2P) does scale: an analysis of P2P traffic patterns," in *Second IEEE International Conference on Peer-to-Peer Computing (P2P 2002)*, Linkoping, Sweden, September 2002.

[3] M. Singh, "Peering at peer-to-peer computing," *IEEE Internet Computing*, vol. 5, no. 6, pp. 4–5, November 2001.

[4] K. Jack, *Video Demystified: A Handbook for the Digital Engineer*, 4th ed. Elsevier, Burlington, MA, USA, 2005.

[5] *MPEG-4 Part-10 Standard specifications - Advanced Video Coding: A codec for video signals*, ISO/IEC Std. 14 496-10, identical to the ITU-T H.264 standard. December 2005.

[6] P. N. Tudor, "MPEG-2 video compression," *IEE Electronics & Communication Engineering Journal*, vol. 9, no. 6, pp. 257–264, December 1995. [Online]. Available: http://www.bbc.co.uk/rd/pubs/papers/paper_14/paper_14.shtml (Last accessed: November 2007).

[7] J. Bier. (2006, April) Introduction to video compression. Berkeley Design Technology, Inc. [Online]. Available: http://www.videsignline.com/howto/showArticle.jhtml?articleId=185301351&pgno=1 (Last accessed: January 2007).

[8] R. A. Quinnell. (2004, September) Introduction to MPEG-4 Video Compression. TechOnLine. [Online]. Available: http://www.techonline.com/community/ed_resource/feature_article/37054 (Last accessed: February 2007).

[9] R. Schäfer, H. Schwarz, and T. Wiegand, "The emerging H.264/AVC standard," Heinrich Hertz Institute, Berlin, Germany, Tech. Rep. 293, January 2003. [Online]. Available: http://www.ebu.ch/trev_293-schaefer.pdf (Last accessed: February 2007).

[10] S. Berson, R. Braden, S. Herzog, S. Jamin, and L. Zhang, "Resource ReSerVation Protocol (RSVP)," RFC2205, Network Working Group, September 1997. [Online]. Available: http://tools.ietf.org/html/2205 (Last accessed: February 2007).

[11] S. Casner, R. Frederick, V. Jacobson, and H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications," RFC1889, Network Working Group, January 1996. [Online]. Available: http://tools.ietf.org/html/1889 (Last accessed: December 2006).

[12] R. Lanphier, A. Rao, and H. Schulzrinne, "Real Time Streaming Protocol (RTSP)," RFC2326, Network Working Group, April 1998. [Online]. Available: http://tools.ietf.org/html/2326 (Last accessed: January 2007).

[13] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC2327, Network Working Group, April 1998. [Online]. Available: http://tools.ietf.org/html/2327 (Last accessed: December 2006).

[14] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC4566, Network Working Group, July 2006. [Online]. Available: http://tools.ietf.org/html/rfc4566 (Last accessed: December 2006).

[15] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, P. Leach, L. Masinter, and J. Mogul, "HyperText Transfer Protocol – HTTP/1.1," RFC2616, Network Working Group, June 1999. [Online]. Available: http://tools.ietf.org/html/2616 (Last accessed: November 2006).

[16] J. Postel, "Transmission Control Protocol," RFC793, DARPA Internet Program, September 1981. [Online]. Available: http://tools.ietf.org/html/793 (Last accessed: January 2007).

[17] J. Postel, "User Datagram Protocol," RFC768, August 1980. [Online]. Available: http://tools.ietf.org/html/768 (Last accessed: February 2007).

[18] F. Kozamernik, "Media Streaming over the Internet - an overview of delivery technologies," European Broadcasting Union, Tech. Rep. 292, October 2002. [Online]. Available: http://www.ebu.ch/en/technical/trev/trev_292-kozamernik.pdf (Last accessed: February 2007).

[19] T. Higgins. (2006, July) Video Streaming Need To Know: Part 1 - Encoding, Bit Rates and Errors. Tom's Guide Publishing (TGPublishing) - Tom's Networking. [Online]. Available: http://www.tomsnetworking.com/2006/07/13/video_streaming_need_to_know_part_1/page8.html (Last accessed: August 2006).

[20] GridNetworks, "GridNetworks Streaming Internet Video Platform Offers Alternative to YouTube, BitTorrent and the Venice Project (Joost)," November 2006. [Online]. Available: http://mediaserver.prweb.com/pdfdownload/479014/pr.pdf (Last accessed: January 2007). Unpublished.

[21] Y.-H. Chu, T. S. E. Ng, A. Ganjam, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early experience with an Internet broadcast system based on overlay multicast," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-CS-03-214, 2003. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/2003/CMU-CS-03-214.pdf (Last accessed: December 2006).

[22] (2006, June) AllCast: Webcasting made simple. [Online]. Available: http://www.allcast.com (Last accessed: September 2006).

[23] (2006, June) The Octoshape Live Streaming solution. Octoshape APS. [Online]. Available: http://www.octoshape.com (Last accessed: September 2006).

[24] (2006, November) End System Multicast: community through technology. ESM Group, Carnegie Mellon University. Pittsburgh, PA, USA. [Online]. Available: http://esm.cs.cmu.edu (Last accessed: February 2007).

[25] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, USA, October 2003. [Online]. Available: http://research.microsoft.com/~antr/SplitStream/ (Last accessed: December 2007).

[26] J. Chuang and A. Habib, "Incentive Mechanism for Peer-to-Peer Media Streaming." [Online]. Available: http://p2pecon.berkeley.edu/pub/HC-IWQOS04.pdf (Last accessed: December 2006).

[27] N. Magharei and R. Rejaie, "Understanding Mesh-based Peer-to-Peer Streaming," in *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2006)*, Newport, USA, May 2006, pp. 56–61. [Online]. Available: http://mirage.cs.uoregon.edu/pub/nossdav06.pdf (Last accessed: June 2007).

[28] B. Li and C. Wang, "Peer-to-Peer Overlay Networks: A Survey," April 2003, unpublished survey. (Last accessed: December 2007). [Online]. Available: http://comp.uark.edu/~cgwang/Papers/TR-P2P.pdf

[29] K. Bertels, B. Pourebrahimi, and S. Vassiliadis, "A survey of peer-to-peer networks," in *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC)*, Veldhoven, Netherlands, November 2005.

[30] T. Asatani, T. Hama, H. Nakazato, and H. Tominaga, "P2P Live Streaming System with Low Signal Interruption," in *The 18th International Conference on Advanced Information Networking and Applications (AINA 2004)*, vol. 1, Graduate School of Science and Engineering, Waseda University, Tokyo, Japan. Fukuoka, Japan: IEEE Computer Society, March 2004, pp. 605–610.

[31] J. Villasenor, J. Wen, and Y. Zhang, "Robust video coding algorithms and systems," in *Proceedings of the IEEE*, vol. 87, October 1999, pp. 1724–1733. [Online]. Available: http://research.microsoft.com/china/papers/Robust_Video_Coding_Algorithms_Systems.pdf (Last accessed: January 2007).

[32] T. Enokido, S. Itaya, M. Takizawa, and A. Yamada, "A scalable multimedia streaming model based-on multi-source streaming concept," in *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS 2005)*, vol. 1, Fukuoka, Japan, July 2005, pp. 15–21.

[33] T. Ahmed, D.-E. Meddour, and M. Mushtaq, "Open Issues in P2P Multimedia Streaming," in *First Multimedia Communications Workshop (Multicomm): State of the Art and Future Directions*, June 2006, pp. 43–48. [Online]. Available: http://www.multicomm.org/proc_multicomm06_8.pdf (Last accessed: December 2006).

[34] S. Casner, R. Frederick, V. Jacobson, and H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications," RFC3550, Network Working Group, July 2003. [Online]. Available: http://tools.ietf.org/html/3550 (Last accessed: January 2007).

[35] S. Robinson, "RDT Feature Level 3.0: Design Specification," RDT transport protocol specifications, RealNetworks, Inc., 2005. [Online]. Available: https://protocol.helixcommunity.org/2005/devdocs/RDT_Feature_Level_30.txt (Last accessed: February 2007).

[36] "Flash Media Server TechNote: HTTP Tunneling Protocols," RTMP Technical Description, Adobe/Macromedia, October 2002. [Online]. Available: http://www.adobe.com/cfusion/knowledgebase/index.cfm?id=tn_16631 (Last accessed: February 2007).

[37] H. M. Alnuweiri, K. Asrar-Haghighi, and Y. Pourmohammadi, "Delivery of MPEG-4 object based multimedia in a multicast environment," in *Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2002)*, Las Vegas, USA, April 2002, pp. 446–451.

[38] S.-S. Yu, X.-W. Zheng, and J.-L. Zhou, "A p2p scheme for live media stream multicast," in *Proceedings of the 12th IEEE International Conference on Multi Media Modeling (MMM 2006),*, Beijing, China, January 2006, p. 4.

[39] R.-I. Chang, J.-M. Ho, S.-C. Hsu, and C.-H. Wang, "Rate-sensitive ARQ for real-time video streaming," in *IEEE Global Telecommunications Conference (GLOBECOM 2003)*, vol. 6, San Francisco, USA, December 2003, pp. 3361 – 3365.

[40] A. H. Li. (2006, June) RTP Payload Format for Generic Forward Error Correction. IETF Internet Draft. IETF Audio/Video Transport Working Group. [Online]. Available: http://tools.ietf.org/html/draft-ietf-avt-fec-07 (Last accessed: February 2007).

[41] J. Cai and C. W. Chen, "Video streaming: an FEC-based novel approach," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE 2001)*, vol. 1, Toronto, Canada, May 2001, pp. 613–618.

[42] S. S. Panwar, K. W. Ross, Z. Liu, Y. Shen, and Y. Wang, "Streaming Layered Encoded Video using Peers," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2005)*. Amsterdam, Netherlands: Center for Advanced Technology in Telecommunications and Distributed Information Systems, Polytechnic University, June 2005.

[43] V. Bharghavan, K.-W. Lee, R. Puri, and K. Ramchandran, "An Integrated Source Transcoding and Congestion Control Paradigm for Video Streaming in the Internet," *IEEE Transactions on Multimedia*, vol. 3, no. 1, pp. 18–32, 2001. [Online]. Available: http://timely.crhc.uiuc.edu/Papers/transcom01.pdf (Last accessed: October 2006).

[44] J. Cai and C. W. Chen, "Use of pre-interleaving for video streaming over wireless access networks," in *Proceedings of the IEEE International Conference on Image Processing (ICIP 2001)*, vol. 1, Greece, October 2001, pp. 934–937.

[45] Y. Shibata, K. Takahata, and N. Uchida, "Optimal Video Stream Transmission Control over Wireless Network," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2004)*, vol. 3, Taipei, Taiwan, June 2004, pp. 1791–1794.

[46] (2003) Nullsoft Streaming Video. AOL/Nullsoft. [Online]. Available: http://www.nullsoft.com/nsv/ (Last accessed: September 2006).

[47] Y. Altunbasak and N. Kamaci, "Performance comparison of the emerging H.264 video coding standard with the existing standards," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2003)*, vol. 1. Baltimore, USA: Georgia Institute of Technology, July 2003, pp. 345–348. [Online]. Available: http://www.ece.gatech.edu/research/labs/MCCL/pubs/dwnlds/h26l_analysis.pdf (Last accessed: January 2007).

[48] M. J. Mirza and G. Raja, "Performance Comparison of Advanced Video Coding H.264 Standard with Baseline H.263 and H.263+ Standards," in *International Symposium on Communications and Information Technologies (ISCIT 2004)*, vol. 2. Sapporo, Japan: Department of Electrical Engineering, University of Engineering and Technology, Taxila, Pakistan, October 2004, pp. 743–746.

[49] S. Wenger, "H.264/AVC Over IP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 645–656, July 2003.

[50] (2006, April) MPlayer: The Open source media player. Made available under GNU General Public License v2. [Online]. Available: http://www.mplayerhq.hu (Last accessed: September 2006).

[51] (2006, November) FFmpeg - complete solution to record, convert, encode, decode and stream both audio and video in multiple formats and containers. Made available under GNU Lesser General Public License (LGPL). [Online]. Available: http://ffmpeg.mplayerhq.hu (Last accessed: September 2006).

[52] (2006, November) VideoLAN - free cross platform multiple format media player (VLC) and streaming server (VLS). Free software released under the GNU General Public License. [Online]. Available: http://www.videolan.org (Last accessed: February 2007).

[53] A. Gupta, B. Liskov, and R. Rodrigues, "Efficient routing for peer-to-peer overlays," in *First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, San Francisco, USA, March 2004, pp. 113–126. [Online]. Available: http://www.pmg.csail.mit.edu/~anjali/nsdi-onehop.pdf (Last accessed: December 2006).

[54] H. Balakrishnan, R. H. Katz, I. Stoica, and L. Subramanian, "OverQoS: An Overlay Based Architecture for Enhancing Internet QoS," in *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI 2004)*, San Francisco, USA, March 2004, pp. 71–84. [Online]. Available: http://nms.lcs.mit.edu/papers/overqos-nsdi04.pdf (Last accessed: February 2006).

[55] D. Doval and D. O'Mahony, "Overlay Networks - A Scalable Alternative for P2P," *IEEE Internet Computing*, pp. 2–4, August 2003.

[56] P. Druschel and A. Rowstron, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[57] S. Birrer and F. E. Bustamante, "The feasibility of DHT-based streaming multicast," in *Proceedings of the 13th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (ASCOTS*

2005), Atlanta, USA, September 2005, pp. 288–296. [Online]. Available: http://www.aqualab.cs.northwestern.edu/publications/SBirrer05FDSM.pdf (Last accessed: October 2006).

[58] "Bittorrent Protocol Specification v1.0," Theory.org, April 2006. [Online]. Available: http://wiki.theory.org/BitTorrentSpecification (Last accessed: February 2007).

[59] (2006, June) The Coral Content Distribution Network. Secure Computer Systems group, New York University. [Online]. Available: http://www.coralcdn.org/ (Last accessed: February 2006).

[60] E. Anceaume, M. Gradinariu, and A. Ravoaja, "Incentive for P2P Fair Resource Sharing," in *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P)*, Konstanz, Germany, September 2005.

[61] D. Chiu, W. Hawe, and R. Jain, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," Digital Equipment Corporation, Tech. Rep. 301, September 1984. [Online]. Available: http://www.cs.wustl.edu/~jain/papers/fairness.htm (Last accessed: December 2006).

[62] A. Singh and M. Haahr, "A Peer-to-Peer Reference Architecture," in *Proceedings of the First International Conference on Communication System Software and Middleware (COMSWARE 2006)*, New Delhi, India, January 2006, pp. 1–10.

[63] L. Melville, I. Sommerville, and J. Walkerdine, "Reference Architectures for P2P Applications," Computing Department, Lancaster University, Tech. Rep. COMP-009-2005, 2005. [Online]. Available: http://www.comp.lancs.ac.uk/computing/users/walkerdi/Papers/ReferenceArch.pdf (Last accessed: December 2006).

[64] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth, "The essence of P2P: A reference architecture for overlay networks," in *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, Konstanz, Germany, August 2005. [Online]. Available: http://eprints.sics.se/246/01/refarch.pdf (Last accessed: February 2007).

[65] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, San Diego, USA, August 2001, pp. 149–160. [Online]. Available: http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf (Last accessed: September 2006).

[66] F. Bjurefors, R. Gold, and L. A. Larzon, "Performance of Pastry in a heterogeneous system," in *Fourth IEEE International Conference on Peer-to-Peer Computing (P2P 2004)*, Zurich, Switzerland, August 2004, pp. 278–279.

[67] L. Huang, A. D. Joseph, J. D. Kubiatowicz, S. C. Rhea, J. Stribling, and B. Y. Zhao, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, January 2004.

[68] L. Huang, A. D. Joseph, J. D. Kubiatowicz, J. Stribling, and B. Y. Zhao, "Exploiting Routing Redundancy via Structured Peer-to-Peer Overlays," in *Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP 2003)*, Atlanta, USA, November 2003, pp. 246–257. [Online]. Available: http://www.project-iris.net/irisbib/papers/routeroute:icnp/paper.pdf (Last accessed: December 2006).

[69] P. Francis, M. Handley, R. Karp, S. Ratnasamy, and S. Shenker, "A Scalable Content Addressable Network," in *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2001. [Online]. Available: http://berkeley.intel-research.net/sylvia/cans.pdf (Last accessed: December 2007).

[70] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, USA, March 2002. [Online]. Available: http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf (Last accessed: January 2007).

[71] J. S. A. Bridgewater, J. S. Kong, and V. P. Roychowdhury, "A General Framework for Scalability and Performance Analysis of DHT Routing Systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2006)*, Philadelphia, USA, June 2006, pp. 343–354.

[72] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data," in *Proceedings of the ACM SIGCOMM 1998 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Vancouver, Canada, August 1998, pp. 56–67. [Online]. Available: http://www.ecse.rpi.edu/Homepages/shivkuma/teaching/sp2001/readings/digital-fountain.pdf (Last accessed: November 2006).

[73] (2006, June) The First Internet Television Peercasting Network. Veoh Networks. [Online]. Available: http://www.veoh.com (Last accessed: September 2006).

[74] (2006, March) Peercast: P2P broadcasting for everyone. Made available under GNU General Public License. [Online]. Available: http://www.peercast.org (Last accessed: September 2006).

[75] H. D. Karatza and K. G. Zerfiridis, "Large scale dissemination using a peer-to-peer network," in *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003, pp. 421–427.

[76] J. Cotton, J. Lopez, and B. Wiley. (2006, June) Alluvium. Foundation for Decentralization Research. [Online]. Available: http://tristero.sourceforge.net/alluvium/techoverview.html (Last accessed: October 2006).

[77] (2006, June) Open Content Network. Onion Networks Inc. [Online]. Available: http://open-content.net/index.html (Last accessed: June 2007).

[78] (2006, June) ACTLab TV: P2P TV Streaming. University of Texas, Austin, TX. [Online]. Available: http://actlab.tv (Last accessed: September 2006).

[79] (2006, June) Freecast: Peer-to-peer streaming. Made available under GNU General Public License. [Online]. Available: http://www.freecast.org (Last accessed: September 2006).

[80] S. Annapureddy and A. Nicolosi, "P2PCAST: A Peer-to-Peer Multicast Scheme for Streaming data," in *First IRIS Student Workshop*, Cambridge, USA, August 2003. [Online]. Available: http://www.cs.nyu.edu/~nicolosi/P2PCast/P2PCast-PR.pdf (Last accessed: December 2006).

[81] Joost. Joost N. V. [Online]. Available: http://www.joost.com (Last accessed: February 2008).

[82] Babelgum - tv just got personal. Babel Networks. [Online]. Available: http://www.babelgum.com (Last accessed: February 2008).

[83] I. Clarke. (2006, June) Dijjer. [Online]. Available: http://www.dijjer.org (Last accessed: September 2006).

[84] B. Bhargava, Y. Dong, X. Jiang, and D. Xu, "GnuStream: a P2P Media Streaming Prototype," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2003)*, vol. 2, Baltimore, USA, July 2003, pp. 325–328. [Online]. Available: http://www.cs.huji.ac.il/labs/danss/p2p/resources/gnustream.pdf (Last accessed: December 2007).

[85] A. Nandan, G. Pau, and P. Salomoni, "Ghostshare - Reliable and Anonymous P2P Video Distribution," in *IEEE Global Telecommunications Conference Workshops (GLOBECOM 2004)*, Dallas, USA, November 2004, pp. 200–210.

[86] Y. Cai and J. Zhou, "An Overlay Subscription Network for Live Internet TV Broadcast," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 12, pp. 1711–1720, December 2006. [Online]. Available: http://www.cs.iastate.edu/~yingcai/osn.pdf (Last accessed: February 2007).

[87] J. Zhou and Y. Cai, "Streaming over subscription overlay networks," in *Proceedings of the 14th International Conference on Computer Communications and Networks (ICCCN 2005)*, San Diego, USA, October 2005, pp. 577–582.

[88] M. A. Eriksen, "Trickle: A Userland Bandwidth Shaper for Unix-like Systems," in *Proceedings of FREENIX Track: USENIX Annual Technical Conference (USENIX 2005)*. Anaheim, USA: Google Inc., April 2005, pp. 61–70.

[89] R. Kumar, Y. Liu, and K. Ross, "Stochastic Fluid Theory for P2P Streaming Systems," in *Proceedings of the 26th Annual Conference of the IEEE Communications and Computer Societies (INFOCOM 2007)*, Anchorage, USA, May 2007. [Online]. Available: http://eeweb.poly.edu/faculty/yongliu/docs/streaming.pdf (Last accessed: May 2007).

[90] K. Kumar and K. W. Ross, "Peer-Assisted File Distribution: The Minimum Distribution Time," in *Proceedings of the First IEEE Workshop on Hot Topics in Web Systems and Technologies (HOTWEB 2006)*, Boston, USA, November 2006, pp. 1–11.

[91] P. F. E. W. Biersack and P. Rodriguez, "Performance Analysis of Peer-to-Peer Networks for File Distribution," in *Proceedings of the 5th International Workshop on Quality of future Internet Services (QofIS 2004)*, Barcelona, Spain, September 2004, pp. 1–10.

APPENDIX A
PACKET FORMATS

TABLE A.1: Media server ANNOUNCE packet format (sent to tracking server)

MSERV	MS-IP	ANNOUNCE	StreamID	Title	Partsize	Metadata
6	22	9	9	64	6	255

Where:

MSERV identifies the packet as a media server originating packet

MS-IP is the originating IP address of the packet (*i.e.* the IP address of the media sever)

ANNOUNCE identifies the packet as an ANNOUNCE message

StreamID is a unique identifier generated for the stream by the media server

Title specifies the title of the video stream

Partsize is the pre-selected size of video chunks (used to keep the partsize constant across all entities) in bytes

Metadata includes additional information about the video stream such as video bitrate, stream length and audio bitrate as well as format/unit specifications for the part size (typically bytes)

TABLE A.2: Media server START packet format (sent to tracking server)

MSERV	MS-IP	START	StreamID	StreamStartTime
6	22	6	9	5

Where:

MSERV identifies the packet as a media server originating packet

MS-IP is the originating IP address of the packet (*i.e.* the IP address of the media server)

START identifies the packet as a stream start update

StreamID is the identifier assigned to the current stream

StreamStartTime specifies the exact time at which the stream started

TABLE A.3: Starter peer CONNECT packet format (sent to tracking server)

SPEER	SP-IP	CONNECT
6	22	8

Where:

SPEER identifies the packet as a starter peer originating packet

SP-IP is the originating IP address of the packet (*i.e.* the IP address of the starter peer)

CONNECT identifies the packet as a stream connection request

TABLE A.4: Tracking server SSETUP packet format (sent to starter peer)

TS	SSETUP	YourID	StreamID	MediaServerIP:Port
3	7	9	9	22

Where:

TS identifies the packet as a tracking server originating packet

SSETUP identifies the packet as a stream setup packet (for the unicast stream)

YourID is the identifier assigned to the requesting starter peer (generated by the tracking server)

StreamID is the identifier assigned to the current stream

MediaServerIP:port specifies the IP address and port on which the media server is listening for packets

TABLE A.5: Regular peer REQUEST packet format (sent to tracking server)

PEER	PEER-IP	REQUEST
5	22	7

Where:

PEER identifies the packet as a regular peer originating packet

PEER-IP is the originating IP address of the packet (*i.e.* the IP address of the regular peer)

REQUEST identifies the packet as a stream request

Table A.6: Tracking server SETUP answer packet to regular peer requesting the video stream

TS	SETUP	YourID	StreamID	StreamStartPoint	Partsize	SrcCount	SourceIP:Port	END
3	6	9	9	6	6	3	22	4

Where:

TS indicates that the packet as originating from the tracking server

SETUP identifies the packet as a SETUP response message to a peer stream REQUEST packet

YourID is the identifier assigned to the requesting peer (generated by the tracking server)

StreamID is the identifier assigned to the current stream

StreamStartPoint specifies the location of the stream pointer at the time of the request such as to allow the requesting peer to request stream parts from this point onwards

Partsize used to keep the partsize constant across all entities for this particular stream

SrcCount is used to specify how many source addresses are included in this response packet

SourceIP:Port is the first of *n* source addresses sent to the requesting peer (*n* = SrcCount)

TABLE A.7: Periodic Peer update packet format

PEER	My-ID	UPeerCount	UpAddr1	...	DPeerCount	DownAddr1	...
5	9	3	22	22	3	22	22

PEER identifies this packet as a regular peer originating packet

My-ID is the server-assigned identifier used to update the peer information in the tracking server database

UPeerCount is the total number of connected source peers

UpAddr is the address or Peer ID for the source upstream peer

DPeerCount is the total number of downstream *client* peers

DownAddr specifies the address or Peer ID for downstream peers

TABLE A.8: Regular peer PARTREQ packet format (sent to upstream peer)

PEER	PeerIP:Port	PARTREQ	MyID	NoOfParts	Partsize	PartID	END
5	22	8	9	3	6	9	4

Where:

PEER identifies the packet as a regular peer packet

PeerIP:Port is the originating address of the packet and is used for sending the reply PART packet(s)

PARTREQ identifies the packet as a streaming data part request (for the peer-to-peer stream)

MyID is the identifier of the requesting peer (originally received from the tracking server)

NoOfParts specifies the number of parts requested in this packet

Partsize is used to maintain a constant partsize across all peers in the system for this particular stream

PartID specifies the actual part identifier of the requested part(s) in this packet (repeated *n* times for *n* parts where *n* = NoOfParts)

Table A.9: Regular peer PART packet format (sent to peer requesting parts with PARTREQ packet) for a partsize of 8192 bytes

PEER	PeerIP	PART	PartID	DataHash	BEGIN	BinaryData	END
5	22	5	9	33	6	8193	4

Where:

PEER identifies the packet as a regular peer packet

PeerIP specifies the originating address of the supplied data part

PART identifies this packet as a data part packet containing a requested data part

DataHash is used for data integrity verification (based on the assumption that the data length is significantly larger than the packet preamble)

PartID is the identifier of the requested data part which is encapsulated in this packet

BEGIN is used along with END to delimit the actual raw data which forms the requested part of the stream

BinaryData is the raw encoded data chunk which is extracted and used to construct the stream *on-the-fly*

APPENDIX B
DERIVATION OF EQUATION 7.12

This equation is used to determine if there is sufficient bandwidth available in the entire streaming network to stream data at the predefined streaming rate r to each and every regular peer. It is assumed that starter peers will always receive the full stream from their respective unicast source (*i.e.* the media server).

In a stable streaming system (based on the architecture described in this dissertation), there will be s_{total} starter peers which are each able to upload data at a total rate specified as s_{uprate}. $s_{\text{total}} * s_{\text{uprate}}$ provides the first half of the total streaming bandwidth requirement.

The second half of the bandwidth requirement is made up of the upstream bandwidth of all regular peers summed together. This is determined by multiplying the number of regular peers p_{total} by the average upload rate for each regular peer p_{uprate}.

The result is the following series of equations (where REQ specifies the total bandwidth requirement):

$$REQ = p_{\text{total}} * r \tag{B.1}$$

$$s_{\text{total}} * s_{\text{uprate}} + (p_{\text{total}} * p_{\text{uprate}}) \geq REQ \tag{B.2}$$

$$s_{\text{total}} * s_{\text{uprate}} \geq REQ - (p_{\text{total}} * p_{\text{uprate}}) \tag{B.3}$$

$$s_{\text{total}} \geq \frac{(p_{\text{total}} * r) - (p_{\text{total}} * p_{\text{uprate}})}{s_{\text{uprate}}} \tag{B.4}$$

APPENDIX C
PACKET CAPTURES

Table C.1: A small segment of the *md5hashes.db* file showing md5sums for parts 2144-2152

MD5SUM (Received Data Part)	Source Address	Part ID
e2003de3c21506f3f389e6cb8de1109b	137.215.153.66	2145
01bfa7d36eac526c275d7db2a6fb4d44	137.215.153.68	2144
e2003de3c21506f3f389e6cb8de1109b	137.215.153.67	2146
b3da1679825bcebf791051fd66df4810	137.215.153.68	2147
e2003de3c21506f3f389e6cb8de1109b	137.215.153.66	2148
c40bf3914b7d6973d38255118bc787be	137.215.153.67	2149
533bc99b0cee0b1e450a04c0f163185c	137.215.153.68	2150
da0a1ca65a2400077aba65caaa2abea0	137.215.153.67	2152
296fc6c1fc145ee4931d10fc9cd9cf0a	137.215.153.66	2151

Table C.2: Segment of the starter peer log file for part requests (for three downstream peers with IDs: 515317, 617153, 732434)

•••	225 (515317)	339 (732434)	227 (515317)	316 (617153)	341 (732434)
229 (515317)	318 (617153)	231 (515317)	343 (732434)	320 (617153)	233 (515317)
322 (617153)	235 (515317)	345 (732434)	324 (617153)	347 (732434)	237 (515317)
326 (617153)	239 (515317)	349 (732434)	328 (617153)	239 (515317)	•••

Table C.3: Segment of the starter peer log file for sent parts (peer IDs: 617153, 515317, 732434)

•••	314 (617153)	226 (515317)	227 (515317)	339 (732434)	316 (617153)
229 (515317)	341 (732434)	318 (617153)	231 (515317)	343 (732434)	320 (617153)
233 (515317)	322 (617153)	235 (515317)	324 (617153)	237 (515317)	345 (732434)
326 (617153)	239 (515317)	328 (617153)	347 (732434)	239 (515317)	•••

TABLE C.4: Segment of a typical part request log file for a regular peer

73 (137.215.153.35:7953)	74 (137.215.153.66:7957)	75 (137.215.153.35:7953)
76 (137.215.153.66:7957)	77 (137.215.153.35:7953)	78 (137.215.153.66:7957)
79 (137.215.153.35:7953)	80 (137.215.153.66:7957)	81 (137.215.153.35:7953)
82 (137.215.153.66:7957)	83 (137.215.153.35:7953)	84 (137.215.153.66:7957)
85 (137.215.153.35:7953)	86 (137.215.153.66:7957)	87 (137.215.153.35:7953)

Table C.5: Segment of a typical log file for successfully received parts (recorded by each regular peer)

121 (137.215.153.35)	122 (137.215.153.66)	123 (137.215.153.35)	124 (137.215.153.66)
125 (137.215.153.35)	126 (137.215.153.66)	127 (137.215.153.35)	128 (137.215.153.66)
129 (137.215.153.35)	130 (137.215.153.66)	131 (137.215.153.35)	132 (137.215.153.66)
133 (137.215.153.35)	134 (137.215.153.66)	135 (137.215.153.35)	136 (137.215.153.66)

TABLE C.6: An example data packet containing part 211

PEER|137.215.153.31|PART|211|BEGIN|421f34ae163de1ff15aba66786aa83365ff635cd2ba40b860dd3fa3a b48c66054f979883c6e8f1c5f493c3fa714cf81b54107e6ab691b2ac53df1274843a5ed488e3cfb23a6ea751eea 8b61e1a2a1ed8b68a440f8f058332a064182129f89341b8e5418205f77ef5625174f4cc3c0836031ca68880b3d 3cc377a8e92de38483423d9b6f5b6cabce349534cf2f4adf4c8ffcd1534be8a8609d3b7f4996426a38ec31978e3 cde7f749810ebd1415fca75511939752fffafe2bacd7a5cd3846b6fdee2471e061d8fa8fe32b27745a92e2ec11a d51135f1b44ddfd176bc0c1f4c2d45c80bd51b55a5128cc4b70cc14cc52165e584d8ffcd16f5bef568a860|END

TABLE C.7: A segment of the tracking server's log file for received packets

MSERV|137.215.153.30:7978|ANNOUNCE|3219528|Test_Stream|2550|16384|null|
SPEER|137.215.153.36|CONNECT|7950|
SPEER|137.215.153.38|CONNECT|7950|
MSERV|137.215.153.30|START|3219528|65306783|
PEER|137.215.153.65:7980|REQUEST|
PEER|137.215.153.63:7980|REQUEST|
PEER|137.215.153.60:7982|REQUEST|

Table C.8: Segment of the log file for a media server's incoming packets (for 6 starter peers)

SPEER|137.215.153.33|ESTABLISH|7005|END|
SPEER|137.215.153.35|ESTABLISH|7005|END|
SPEER|137.215.153.33|ESTABLISH|7006|END|
SPEER|137.215.153.65|ESTABLISH|7005|END|
SPEER|137.215.153.68|ESTABLISH|7007|END|
SPEER|137.215.153.40|ESTABLISH|7005|END|