OPTIMISING NETWORKS FOR ULTRA-HIGH DEFINITION VIDEO

Paul W. Farrow

A thesis submitted for the degree of Doctor of Philosophy

School of Computer Science and Electronic Engineering

University of Essex

April 2016

I would like to thank my family for the help and support they have given me throughout my studies. I especially want to express my special gratitude to my parents, Sharon and Ron Farrow, whose continued support towards my life goals have led me to this position.

I also want to thank my many friends who have supported me throughout my progress and helped maintain a healthy work and personal life balance; I will always remember and appreciate their support.

Acknowledgements

I want especially thank my supervisor Dr Martin Reed for his continuous support and guidance throughout my Ph.D study; without his support, I would surely have stumbled at the many roadblocks encountered throughout my research. His expertise and knowledge proved invaluable, in both my research and other aspects of my Ph.D.

Furthermore, I would also like to thank the many staff and research students in my department, that have provided help and support with my research.

Throughout my Ph.D, there have been several projects that have enabled me to complete valuable research and collaboration. Firstly, I want to thank the VISIONAIR project and the team at the Poznan Supercomputing and Networking Center (PSNC), for hosting myself and my research during a short research project; the research project allowed results to be gathered for use in both a Journal publication and this thesis. I would also like to express special thanks to the team at the International Center for Advanced Internet Research (iCAIR), part of Northwestern University in Chicago, for both hosting me during my stay, as well as providing the resources and support required to complete several testbed scenarios; this collaboration was possible, thanks to the Slice Around the World Initiative. The collaboration with iCAIR enabled collection of significant results crucial to the presented research, as well as providing the opportunity to collaborate on journal publications. Additionally, I would also like to thank the people at Janet, for configuring and providing the network connectivity between sites, during the various testing and project demonstrations; without this connectivity, some of the collaboration projects would not have been possible.

I would also like to thank the EVANS project and Tsinghua University in Beijing, who hosted me during a research trip to their facilities; the support they provided, also gave me the opportunity to present my work at an IEEE conference.

I would also like to express special thanks to the Engineering and Physical Sciences Research Council (EPSRC) for funding my study, without their financial support I would most likely not have been able to complete my research.

Abstract

The increase in real-time ultra-high definition video services is a challenging issue for current network infrastructures. The high bitrate traffic generated by ultra-high definition content reduces the effectiveness of current live video distribution systems. Transcoders and application layer multicasting (ALM) can reduce traffic in a video delivery system, but they are limited due to the static nature of their implementations. To overcome the restrictions of current static video delivery systems, an OpenFlow based migration system is proposed. This system enables an almost seamless migration of a transcoder or ALM node, while delivering real-time ultra-high definition content. Further to this, a novel heuristic algorithm is presented to optimise control of the migration events and destination. The combination of the migration system and heuristic algorithm provides an improved video delivery system, capable of migrating resources during operation with minimal disruption to clients.

With the rise in popularity of consumer based live streaming, it is necessary to develop and improve architectures that can support these new types of applications. Current architectures introduce a large delay to video streams, which presents issues for certain applications. In order to overcome this, an improved infrastructure for delivering real-time streams is also presented. The proposed system uses OpenFlow within a content delivery network (CDN) architecture, in order to improve several aspects of current CDNs. Aside from the reduction in stream delay, other improvements include switch level multicasting to reduce duplicate traffic and smart load balancing for server resources. Furthermore, a novel max-flow algorithm is also presented. This algorithm aims to optimise traffic within a system such as the proposed OpenFlow CDN, with the focus on distributing traffic across the network, in order to reduce the probability of blocking.

Contents

List of Figures

List of Tables

Acronyms

ALM Application Layer Multicasting

API Application Program Interface

ARP Address Resolution Protocol

BGP Border Gateway Protocol

CDN Content Delivery Network

CLI Command Line Interface

CPU Central Processing Unit

DASH Dynamic Adaptive Streaming over HTTP

DHT Distributed Hash Table

DNS Domain Name System

DPI Deep Packet Inspection

EIGRP Enhanced Interior Gateway Routing Protocol

ESM End System Multicast

GA Genetic Algorithm

GRE Generic Routing Encapsulation

HD High Definition

HLS HTTP Live Streaming

HTTP Hypertext Transfer Protocol

ICN Information Centric Network

IMFMC Integer Multicommodity Flow with Minimum Congestion

IP Internet Protocol

ISP Internet Service Provider

MAC Media Access Control

MCF Maximum Concurrent Flow

MFMC Multicommodity Flow with Minimum Congestion

MPD Media Presentation Description

MSS Microsoft Smooth Streaming

NAT Network Address Translation

OS Operating System

OVS Open vSwitch

P2P Peer-to-Peer

QoS Quality of Service

RAM Random Access Memory

RIP Routing Information Protocol

RTP Real-time Transport Protocol

SD Standard Definition

SDN Software Defined Networking

SLA Service-Level Agreement

SSH Secure Shell

TCP Transmission Control Protocol

UDP User Datagram Protocol

UHD Ultra-High Definition

VLC VideoLAN Client

VM Virtual Machine

Mathematical Symbols

F	Video frame
F_w	Frame width
F_h	Frame height
F_r	Frame rate
C	Colour aspects of the video
C_d	Colour bit depth
C_m	Colour model
B	Bitrate of video in bits/s
V_i	Video 'use value'
V_p	Video popularity rating
V_s	Video segments already cached
L	Total network load
D	Set of all demands
d	Video demand
s	Source
t	Destination
$d_{s,t}$	bitrate of d between s and t
$E(d_{s,t})$	Set of edges that d is transmitted over
V	All network nodes
E	All edges
G	A network graph
$G(E, V)$	Network graph comprising of E and V
S	Source nodes for content
T	Destinations for content
A	Compute resources in the network
a	A compute resource
e	A network edge
$u(e)$	Capacity along e
q	Video codec/quality
b_q	Video bitrate using codec q
$R(s, t, b_q)$	Traffic through network from s to t for b_q
D'	Successful demands
$P(D)$	All paths through network for D
p	A path through the network
$x(e_p)$	Traffic flow on e for a specific path p
$P(D')$	All paths for D'
$x(p_d)$	Traffic flow along a path meeting d
λ	Separation constant
N	Number of transcoders
∞	Infinity
X	Solution flows to meet all D

Ω_L	Proportion of spare capacity on a given link
$x(p)$	The flow along a path p
P_a	All possible paths through the network
P_d	Set of all possible paths associated with demand d
p_d	Single path for d
P_I	Set of all integer paths
θd	Objective to meet proportion of d
Y	Fractional flows to fully meet d
y	Fractional or complete flow to meet d
$y(p)$	Flow over p
P_e	Paths traversing e
θ'	Optimum proportion for all y
Ω'_L	Optimum spare capacity for all x
θ_ϵ	ϵ-optimal solution
β	Dual variable
l	Length value
$l(e)$	Length across e
n	Step of algorithm operation
$p_m(d)$	Minimum cost path for d
$F(l)$	Shortest path using lengths
$C(p_m(d))$	Cost of $p_m(d)$
$f(y)$	Flow across y
$p(y)$	Path of y
β'	Optimal dual value
t	Number of phases in algorithm operation
Γ'	Optimised flow data

Chapter 1

Introduction

Research into improving video delivery systems is essential as, although the Internet was not specifically designed to support this type of content, video is now the predominant traffic. This is becoming a more prevalent issue, as the demand for streaming Ultra-High Definition (UHD) video to large numbers of clients increases [1, 2]. One of the main aims of the presented work, is to investigate the practical solutions and improvements for transporting UHD content across a network infrastructure. A simple example scenario, where clients in the UK are being sent a live real-time UHD video stream from a server in the USA, is shown in Figure 1.1. The stream is transcoded using a transcoder initially located in the USA, which then transmits the stream across one of the limited capacity transatlantic cables. In order to reduce the replicated traffic along the transatlantic link, it is desirable to migrate the transcoder from the USA to the UK. However, existing systems that allow migration during streaming, produce a significant disruption to the client viewing experience; these disruptions can include distortion to the video content,

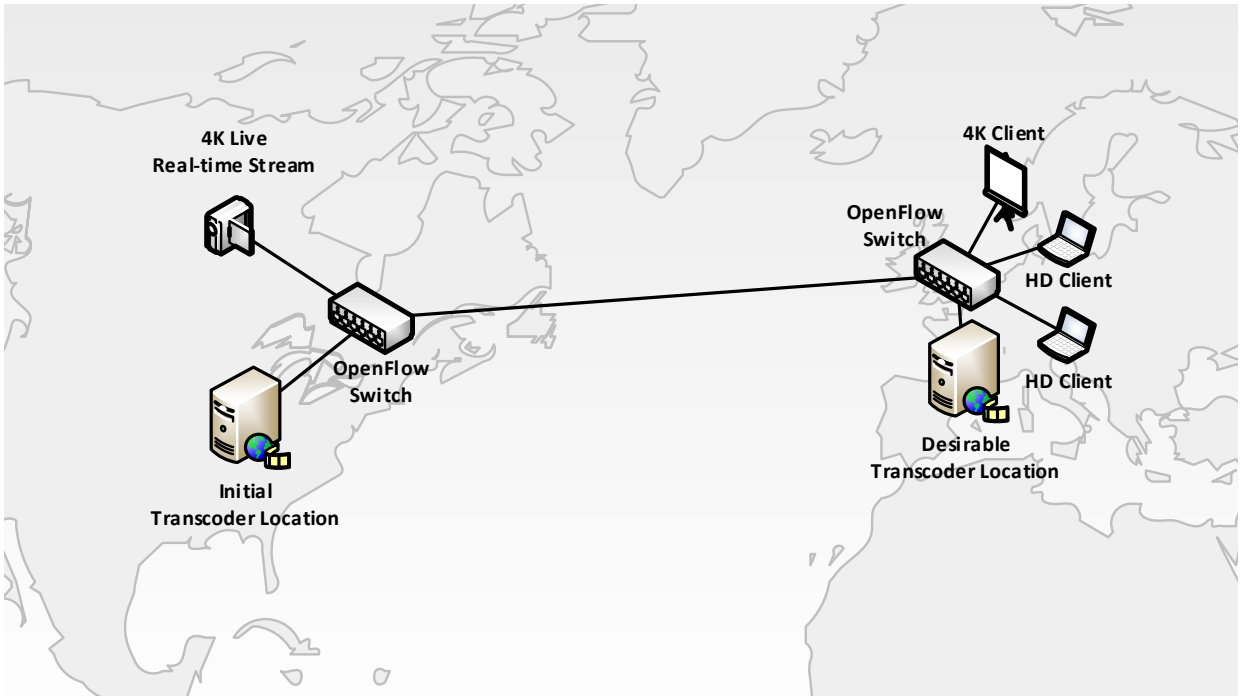


Figure 1.1: Example of a simple use case for the presented research.

as well as multi-second interruptions in the stream. To overcome these issues, a novel system to dynamically migrate transcoders during streaming is presented, which operates with minimal to no interruption in client viewing experience. Additionally, a novel heuristic algorithm for optimising the placement of transcoder resources within a network is presented, which focuses on reducing overall network traffic. Use of the proposed algorithm will enable the creation of a system that can determine when such a migration is beneficial, then instigate migration using the proposed migration mechanism. It should also be noted, that transcoder migration would also be beneficial for optimising several other factors, such as financial cost, latency and server utilisation. Such scenarios are discussed in detail within the relevant chapters. Additionally, the concept of the migration system is not restricted to the movement of a single transcoder; the system can be further expanded to migrate entire Internet Protocol (IP) topologies, that could encompass large numbers of linked transcoders.

OpenFlow is a Software Defined Networking (SDN) technology that separates the control and datapath elements of network devices, allowing centralised control using various software applications. The main benefit of the described flexibility in traffic control, is the ability to influence traffic in ways that are not possible with existing networking systems; this is due to the absence of normal routing protocols and layer 2 switching behaviour, which are not required within an OpenFlow enabled switch. OpenFlow is used to implement a number of the proposed systems, allowing capabilities such as the seamless migration and redirection of traffic within the migration scenario. This type of traffic manipulation is known to be difficult to achieve using standard networking mechanisms, which would make developing the migration system without SDN technology impractical. A more in-depth and detailed description of OpenFlow is presented in Section 2.6.1, along with analysis of OpenFlow controllers in Section 2.6.3.

In addition to the described migration system, the thesis presents an improved video delivery architecture. The proposed architecture includes characteristics developed through the investigation of current video delivery systems, such as Peer-to-Peer (P2P) and Content Delivery Network (CDN) architectures. However, with these systems mainly being used for stored video content, which can be cached, only factors which were beneficial to the delivery of real-time video content were investigated and considered. Applications for the proposed video delivery system can be found in Chapter 6, which presents the improved CDN architecture, utilising the capabilities of OpenFlow. Within the proposed system, the source of the content is selected using controller based load balancing, taking into account server utilisation, geographical location and network link capacity. The controller based selection process ensures timely delivery of content, as well as improving overall network performance, by reducing bottlenecks in the content delivery process. To support the proposed architecture, a max-flow algorithm capable of reducing the probability of blocking is presented in Chapter 7. The presented algorithm distributes traffic evenly across the network, with the goal to minimise the number of congested links. By reducing the number of congested links, the probability that there will be an available path through the network to meet additional demands is increased.

The main question that this thesis strives to answer is: how can traffic be minimised when transporting UHD content across both the current Internet infrastructure and private networks? The thesis will answer this question using theoretical analysis and practical solutions designed

through modelling and testing. As previously mentioned, a large part of the presented work utilises the OpenFlow platform to create the various proposed systems; therefore, it is assumed that in many cases, a fully OpenFlow enabled single operator network is in operation.

1.1 Motivations

The main motivation for the presented research was a desire to improve current real-time UHD content streaming systems. The objective was to both optimise existing mechanisms, such as application layer multicasting and transcoding, as well as developing a novel system to enable dynamic migration during transmission. Achieving these objectives would allow improved scalability for streaming systems, which were not originally designed to carry content such as UHD video. The described increase in scalability would be a result of the reduced network load, facilitated by the optimisation techniques presented.

Additional motivations include the pursuit of an improved Content Delivery Network (CDN) using OpenFlow, enabling optimisations and operations that were not previously possible. The pursuit of the improved CDN is an extension of the previous motivations, as the CDN system is designed with the scenario of providing UHD content to clients, through a private single operator network infrastructure. Following on from the OpenFlow CDN, a novel traffic optimisation technique is also presented; the max-flow traffic optimisation algorithm aims to improve scalability, by distributing traffic evenly across all network links, thereby reducing the probability of blocking.

1.2 Contributions

The key contributions presented include:

- A novel heuristic algorithm to provide optimised locations for transcoders within a given network, when presented with a collection of client video demands.
- A novel mechanism to reduce live migration times of transcoders processing live real-time UHD content, in order to provide a near seamless switchover for clients.
- A novel OpenFlow enabled CDN concept, including: an OpenFlow based NAT system to handle bridging the CDN with the public network, as well as providing optimised server selection; a novel OpenFlow based DNS redirection alternative, which improves on existing implementations; an OpenFlow core network to allow rapid redirection of streams during transmissions, and finally, a mechanism for performing multicasting at the switch level using OpenFlow.
- A novel max-flow algorithm capable of optimising the internal traffic of CDN architectures, including the presented OpenFlow CDN.

1.3 Thesis Description

The thesis is structured as follows: Chapter 2 presents a discussion of relevant literature, as well as background information on relevant technologies used. Chapter 3 provides an examination of

the performance benefits of both P2P and CDN architectures, to provide an insight on improving video streaming services. Chapter 4 provides a detailed description of the proposed heuristic algorithm for placing transcoders within a network, along with a comparison to other methods, such as a genetic algorithm and two Dijkstra algorithms. Chapter 5 presents a method for using OpenFlow to migrate a transcoder while streaming, with minimal interruption to clients viewing experience. Chapter 6 presents the concept of an OpenFlow enabled CDN system, as well as its use as an architecture to deliver live streaming content. Chapter 7 presents a max-flow algorithm, which optimises traffic placement within a CDN architecture, such as the one presented in Chapter 6. Finally, conclusions are presented in Chapter 8 with an overview of the presented work, as well as a discussion of potential areas of future research.

Chapter 2

Background Literature

Within this chapter, relevant literature will be presented along with a critical review of their content. The research areas explored within the presented work is diverse, as it encompasses a number of technologies which all require explanation; technologies that are less relevant are briefly described, with references to literature that provides a more in-depth discussion and analysis of the research area. It should be noted that additional background content can be found in the relevant chapters, where the context of the technologies application can be examined more extensively in relation to presented work. It is important to note that although the scenario of UHD video is used throughout the thesis, the main focus of the work is the switching and control of these high bitrate real-time streams and not the specific encoding or format of the video content. Furthermore, the presented systems are designed to be agnostic to the format and encoding of the content, providing a system that can be utilised in a diverse number of applications.

Several technologies and platforms that are rapidly advancing, are utilised in the presented research; one example of these technologies is the OpenFlow platform and its controller applications, which is an area that is currently in continuous development. The use of these types of rapidly advancing technologies located within a diverse collection of research areas has produced a wide variety of literature, making it important to focus on highlighting relevant work.

Some references being used within this chapter and later on in the thesis are from primary industry sources; this is due to the information required not currently being available in academic journals.

2.1 Ultra-High Definition Media

Streaming UHD content across networks is the main focus of the presented research, with the main aims being to perform this in the most efficient way. The obvious advantages of optimising UHD streaming are the conservation of link capacity and resources, as well as reducing playback delay. Optimising these high bitrate streams is becoming a significant issue as video traffic is becoming more prevalent in current public networks, as well as becoming one of the largest contributions to network traffic [3–5].

2.1.1 Specification

The UHD TV standard accepted by the International Telecommunication Union (ITU) contains two resolutions for UHD video; one for 4K video set at 3840 x 2160 pixels and one for 8K video set at 7680 x 4320 pixels [6]. Picture resolutions are also often presented as megapixel (MP) values, which represents the number of pixels within a frame of the video; therefore, 1 MP is equal to precisely 2^{20} pixels, though it is often referred to as roughly 1 million pixels. These equate to pixel values of 8 MP for 4K and 32 MP for 8K, which represents a large improvement over the current HDTV pixel count of between 1-2 MP. These, however, are not the resolutions that are always used or considered as UHD; this is mainly due to the fact that equipment for 4K and 8K video began production before the official standard was finalised. Additionally, not every organisation will follow the standards set by the ITU, with organisations defining their own standards; for this reason, there is flexibility in the type of content that is considered UHD.

Standard Definition (SD) and High Definition (HD) content is utilised during the prototype development in Chapter 5, so it is appropriate to briefly define its specification alongside UHD. SD video is generally considered anything below that of HD, although official resolutions range from 720 x 480 to 720 x 576 [7]. HD is generally considered to be anything between 1280 x 720 and 1920 x 1080 in resolution, although 1920 x 1080 is usually termed Full HD [7, 8]. There are multiple variations of each video standard, with differences in frame rates, as well as progressive and interlacing display types.

The JVC DLA-SH4KG 4K projectors utilised for the system presented in Chapter 5, are capable of projecting an image at the resolution of 4096 x 2400 [9]. Although the projectors resolution does not follow the exact specifications of any 4K standard, as discussed before, it is still considered 4K. The current implementation at the University makes it possible to produce resolutions up to 8K, with the use of four projectors being used in parallel to create a merged image. However, 8K will not be detailed further, as the computing resources required to transcode content are not available. The successful operation of the system working with 4K content, however, should prepare foundations to allow scaling to 8K if the resources were available.

Ultra-high definition video consumes a significant amount of link capacity, typically in the order of 100 Mb/s for ultra-high quality compressed formats of 4K video [10]. It is possible to calculate the required bitrates for uncompressed 4K video streams using Equation 2.1. 4K uncompressed video streams at the resolution of the University JVC projector would amount to a data rate of 7.55 Gbps or 943.72MB/s; given a frame rate of 24fps, colour depth of 8 bits/colour and a colour model of CMYK (4 bits).

You are able to calculate the upper limit bitrate b in bits/second of a video using the following equation:

$$F_w \times F_h \times F_r \times C_d \times C_m = b \tag{2.1}$$

with F_i, where $i \in w, h, r$ being the attributes of the frames of the video, F_w as the width, F_h as the height and F_r as the frame rate. C_i, where $i \in d, m$ can be defined as the colour aspects of the video, with C_d being the colour depth and C_m being defined as the colour model used.

This upper limit bitrate is often not required for transmitting the video data across the network, as with the use of codecs and transcoders it is possible to reduce transmission rate significantly. Some codecs can achieve compression using a mathematically lossless method, while others can

reduce the data rate even further with minimal loss of visual data; these techniques will be detailed in Section 2.1.3.

2.1.2 Transcoding

Transcoding is the general term for systems that can modify coded media to suit alternative codecs and/or bitrate requirements [11]. Transcoding digital media on the Internet is an important step in the preparation and delivery of video content to end users; it provides not only a reduction in bitrate, so that more content can be delivered on a given set of links, but also allows the content to be tailored and adapted to the receiving device [12]. Transcoding can include adapting the resolution of the video to match that of the client display, as well as transcoding to a format supported by the client [11]. Transcoding can be performed with both live content streams as well as stored content. However, transcoding live streams can be more resource intensive because of the real-time requirement. As previously described, transcoding using certain video codecs can reduce the bitrate of media further, by using either lossless or lossy compression techniques [13, 14]; the use of such techniques are a valuable tool when delivering content to low powered devices.

FFmpeg [15,16] is a software based multimedia tool, one of its many capabilities is its extensive transcoding functionality. It has a very large feature set that makes it ideal for the applications that this thesis requires. One of the main features that makes it suitable, is its ability to accept video packets through a UDP port and send the transcoded output through another. This ability makes FFmpeg the ideal candidate for the scenarios used in Chapter 5, as it has the required functionality built in. It also has extensive codec support and content transformation capabilities, allowing a diverse choice of options when determining optimum streaming conditions. The large list of additional transcoding capabilities of FFmpeg will not be detailed here, as they would not provide any relevant information for the presented work; however, relevant information relating to FFmpeg configuration is included where required in Chapter 5.

An ideal streaming encapsulation would be the MPEG Transport Stream (MPEG-TS) format [17,18], which would easily allow VideoLAN Client (VLC) media player [19] to access the content over a UDP socket; the server could also use VLC to stream the video to the transcoder using UDP, due to VLC also possessing good network streaming support. However, VLC's streaming capabilities will not be detailed further, due to them only being utilised in initial testing before switching to FFmpeg.

It should be noted that although we are assuming a software based transcoding approach for the system presented in Chapter 5, it could easily be applied using hardware encoders with slight modifications.

2.1.3 Codecs

Codecs provide a solution to reducing the bitrate of a video, in order to meet capacity and client device requirements. Codecs help to provide media in a format that can be processed by a receiving device, as not all devices contain the appropriate hardware or software to decode specific content. Codecs usually have two types of operating modes, lossless and lossy [13, 14]; but the lossless mode can also be separated into two subcategories, mathematically lossless and visually lossless. Codecs that use a mathematically lossless algorithm can reduce the bitrate of content, while still

being able to fully reconstruct the original format [13,20,21]. Similarly to mathematically lossless, using a visually lossless algorithm does not introduce visible distortion to the content; however, the content once converted, can never be fully restored.

It should be noted that although a brief description of codec technology is given, the system being implemented in Chapter 5 is agnostic to the codec used; this is due to the network orientated aspect of the migration, which only relies on codecs in general, to aid in transporting the content at a reduced bitrate.

One of the most suitable codecs for use with UHD content is the JPEG 2000 codec; it has several features that allow for reliable and fast transport of UHD media, at a reduced data rate compared to its uncompressed counterpart [13]. It allows on average a 2:1 compression ratio when using a mathematically lossless compression method; but it can improve on this further, as it allows visually lossless compression ratios of 10:1 to 20:1 [22]; it should be noted, however, that these values do depend on the content being processed. The JPEG 2000 codec provides improved performance in both image quality and compression ratios when compared to other codec solutions; a comparison of features with MPEG based codecs is shown in Table 2.1.

Features	**MPEG-2 I-Frame**	**MPEG-2 Long GOP**	**MPEG-4 AVC**	**JPEG 2000**	**JPEG 2000 Benefits**
Intra-coding Only	Yes	No	No	Yes	Easy editing, avoids introducing noise caused by GOP pumping, GOP alignment.
Low Latency	Yes	No	No	Yes	Avoid delays in live news interviews. Reduce AV sync issues.
Efficient Coding	No	Yes	Yes	Yes	High-quality pictures in manageable size files.
Scalable Decoding	No	No	Yes	Yes	Real-time browse even on a laptop. Embedded proxy eliminated mismatch.
Main Artefacts	Blocks	Blocks, motion aliasing	Blocks, motion aliasing	Softness	Downstream encoders use bits to encode your pictures instead of artefacts.
Multi-Gen Performance	OK	Poor if GOPs misaligned	Poor with deblocking	Good	Confidence that the emission picture quality is as high as it can be.
Open Standard	Yes	Yes	Yes	Yes	Interoperability. Choose from multiple vendors.
No/Low Licensing	No	No	No	Free	Lower costs and richer toolsets.
10-bit,4:2:2 Colour	No	No	Sometimes	Yes	Accurate green/blue screen work today. Perfect pictures from your archive tomorrow.

Table 2.1: Comparison of JPEG 2000 with other available image codecs, presented by Bancroft *et al.* [23].

Another more recent codec is H.265 HEVC (High Efficiency Video Coding), which is the successor to the hugely successful H.264 AVC Codec [24, 25]. The HEVC codec builds on top of H.264 AVC so that it can essentially replace all its applications. HEVC focuses heavily on the areas of increased resolution and parallel architectures, in order to perform more efficiently with content such as UHD video. It also looks to improve on many areas of the H.264 AVC codec, such as data loss resilience, coding efficiency and better integration with transport systems [26, 27].

The mechanisms that video codecs utilise to achieve this bitrate reduction is not discussed, as it is not relevant to the presented work; however, as a brief description of their methods, they use a combination of mathematical algorithms that are similar to those used in still image compression; these algorithms are then used in combination with features such as motion estimation and the use of reference frames to compress the video data [28].

The presented work does not focus on detailed codec analysis, as the systems presented are designed to be agnostic to codecs; however, Conklin *et al.* [29] provide further analysis of this research area, with a detailed description of codec evolution with regard to improving streaming video services.

2.1.3.1 Streaming Technology

Following on from the discussion of codecs and transcoding, it is important to briefly discuss the technology and techniques utilised for aiding in the delivery of video content to clients. Technology such as HTTP Live Streaming (HLS) [30] and Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH [31], aims to improve the delivery of video content, by utilising existing HTTP protocols already being used by a large number of devices. HLS was developed by Apple and is an older technology when compared with the relatively new MPEG-DASH; because of this, many systems still currently utilise HLS, such as the live streaming applications described later in Section 2.8. However, with MPEG-DASH now being an international standard, it is expected that it will gain increased adoption over the older proprietary technologies like HLS and Microsoft Smooth Streaming (MSS) [32]; for example, two large video delivery corporations that currently utilise MPEG-DASH as their preferred delivery method are Youtube and Netflix, which serve a very large percentage of all video traffic on the internet. There are, however, many similarities between MPEG-DASH, HLS and MSS, especially in the way they divide content into segments for easy delivery to clients; most of the differences between the technologies relate to the way they handle the segments and manifest files, rather than the way they transport the content.

Figure 2.1 shows how content segments are represented within the MPEG-DASH system. Each video will have a Media Presentation Description (MPD) such as this, describing the available segments, adaptations and representations of the video content; the information contained within the MPD, allows clients to automatically detect and select the content they require. Segments are split along the temporal axis, with the period being configured during initial content set up. Each of the periods found within the MPD has one or more adaptation sets, which can contain various adaptations of the content, such as the audio and video components of the stream. Within each adaptation, there can also be one or more representations, which can contain information on the different variations of the adaptation set data that is available; these variations can include different resolution, bitrate and formats, enabling clients with differing requirements access to the same content. Another important aspect of the varying bitrate representations, is the capability of MPEG-DASH to detect congestion and delay issues and switch to lower bitrate representations; once congestion has been mitigated, MPEG-DASH can then attempt to switch back to the previous higher bitrate segments. This adaptive streaming aspect of the system, coupled with the high compatibility of using the HTTP protocol found in many devices, makes it both a robust and compatible streaming technology.

MPEG-DASH also has a long list of other capabilities that extend beyond those of the previously mentioned proprietary streaming technologies. One of these capabilities is that MPEG-DASH is codec agnostic, meaning it can be utilised with a wide variety of codecs, such as H.264 or H.265. This allows the delivery of content to a diverse range of devices, which may not all support the same formats. Additionally, like HLS, MPEG-DASH can be deployed on ordinary HTTP

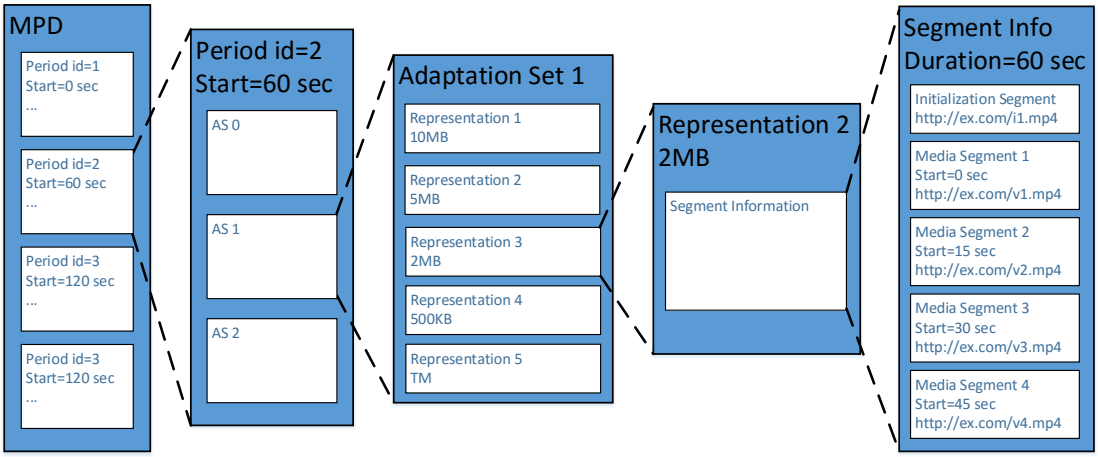


Figure 2.1: MPEG-DASH Media Presentation Description (MPD) format [31].

servers, allowing it to be easily added to existing systems without the modifications required for other systems such as MSS. An additional capability of MPEG-DASH that content providers such as Youtube are already utilising, is the ability to insert adverts during video playback; this is an important feature given that a lot of internet based video services rely on adverts as a funding source. MPEG-DASH has many other features which go beyond the research area explored in this thesis, so they have been omitted for clarity; however, a detailed description of the capabilities of MPEG-DASH is presented by Sodagar [31], who discusses MPEG-DASH with comparisons to HLS and MSS.

2.2 Resource Optimisation

As hardware resources continue to expand in both size and power, it is important to ensure that they are being utilised efficiently; this can be important for many reasons, including reducing financial costs, reducing blocking, environmental impact reduction and also for meeting performance requirements. Additionally, with the increased demand for high bitrate applications such as UHD video, it becomes crucial to optimise traffic within the network, to allow the limited capacity of existing networks to continue to meet the requirements of users [3]; furthermore, the high computational cost of transcoding the video content also needs to be considered [3], as this will influence the availability of possible transcoder placements throughout a network.

Optimisation of resources can use human based prediction using empirical evidence or estimated values; however, it is more common for optimisation techniques to utilise algorithms to aid in providing a better solution. Using algorithms to optimise resources provides a systematic approach to the optimisation process. There are many algorithmic approaches that are available, some of which are utilised within the later chapters; these include:

- Deterministic heuristic algorithms which leverage known characteristics of a problem, to improve both solution quality and calculation time; there is, however, a trade-off between solution quality and calculation time in most algorithms of this type. A heuristic is used to optimise transcoder placement in Chapter 4.

- Non-deterministic algorithms, including evolutionary approaches such as Genetic Algorithms (GAs). GAs involve the application of evolutionary algorithms using natural selection concepts, to provide improved solutions throughout a number of generations [33]. For this process, each generation of solutions is tested and then evolved in the hope of creating a better solution within the next generation. A GA is utilised in Chapter 4 for comparison with the presented heuristic, as GAs are shown to provide 'good' solutions to problems given the correct parameters are configured.

- Black box techniques may use either approach and typically involve multiple techniques to reach a given goal, this includes black box solvers such as CPLEX [34, 35] which is detailed in Section 7.2.

Further information regarding the three algorithm approaches can be found in the relevant chapters in which they are utilised.

It should be noted that most of the current implementations of placement strategies and algorithms described in this section, mainly focus on stored or cached content placement; unfortunately, they do not consider the placement of transcoders for the real-time streaming scenarios, so are not directly related to the specific research area explored. However, although different from live streaming, they are still relative due to shared concepts, such as the positioning of content distribution points close to users. This area of resource optimisation is detailed further in Section 2.8.1.

This thesis examines and provides solutions to optimise both network resources and compute (transcode) resources. This section will only briefly describe the area of resource optimisation, as further detailed information more relevant to specific aspects of the presented work can be found in the relevant sections; these include: Section 2.3, which discusses resource optimisation in relation to cloud environments; Chapter 4, which discusses the optimisation of network capacity and transcoding resources; and finally, Chapters 6 and 7, which include a number of sections which detail specific optimisation of network resources.

2.2.1 Network Resource Optimisation

Network resource optimisation can involve a number of specific optimisation goals, including traffic reduction, blocking reduction, latency reduction and load balancing. Some of these goals are not exclusive and are closely linked; for example, reducing traffic in the network often results in a reduction in blocking, due to there being additional capacity available within the network for further traffic. However, there are also other ways to reduce blocking within the network, such as spreading the load across the network in a load balancing approach. These close links between optimisation goals are important, as they provide insights into multiple solutions to larger optimisation problems. These types of problems often require using multi-objective optimisation techniques, in order to achieve the best solution possible. This is because multi-objective optimisation allows for a far wider search of solution parameters than normal optimisation techniques.

With the existing optimisation goals in mind, it should be clear that each characteristic being improved could benefit specific scenarios; furthermore, it is important to tailor optimisation techniques to the application being utilised on the network. For example, delay tolerant content

delivery would not benefit significantly from latency reduction, but the reduction in traffic would provide improved scalability due to additional capacity being available for further demands. In addition to this, a reduction in traffic would also reduce transmission costs for any links where cost is linked to usage.

Hu *et al.* in [36] present an important observation that network traffic can be optimised by tuning three basic factors: topology infrastructure, traffic distribution and routing strategy. However, aside from the optimisation of the network itself and the traffic traversing through it, it is important to show how the placement of utilised resources within the network can heavily influence the limits of optimisation. The specifics of how resource placement optimisation can improve content delivery is detailed in Section 2.2.2, with the focus of the discussion being the work presented in Chapter 4.

A more in-depth overview of traffic optimisation can be found in Chapter 7, where a traffic optimisation system is presented.

2.2.2 Resource Placement Optimisation

When aiming to optimise a system that uses resources dispersed across a network, it becomes beneficial to look at optimising the placement of these resources; these may include resources such as processing, storage and services. Optimising the placement of these resources can not only improve the systems performance, but also reduces the traffic generated from communication to these resources. This is an area of research that Chapter 4 explores, in relation to the optimisation of the placement of transcoders. This is a similar optimisation problem to that of cache and CDN server location optimisation, however, it is important to separate the two since they have different end goals; cache placement aims to position caches in locations to improve performance and reduce traffic, whereas transcoder placement for live video streams also needs to factor in the distance from the original source when computing solutions.

Cache and CDN server placement strategies have been widely studied in previous research, due to the desirable benefits of improving performance while reducing costs. One of the more recent works into CDN replica placement algorithms is presented by Sun *et al.* [37]. Sun *et al.* focus on clustering requests from users, in order to aid in optimising the placement of replicated data at CDN servers. An important distinction for placement strategies is the focus on the requested content type; original placement strategies that focused on web page replication, such as those presented by Qiu *et al.* [38], are not suitable for use with high bitrate multimedia [39]. It is shown by Wauters [40] that in terms of content replication within CDNs, greedy algorithms seem to provide a reliable solution to placement optimisations. Wauters [40] presents heuristics to work with this premise, however, they do not consider the application of the system to live streaming content. Cidon [41] and Leighton [42] both present a thorough analysis into how the placement of multimedia content within the network infrastructure, can significantly impact the performance and scalability of video delivery systems. They both provide detailed explanations of the issues associated with content distribution, focussing heavily on the conservation of network capacity and the minimisation of financial cost to content providers. Furthermore, Nguyen *et al.* [43] detail the optimisation parameters associated with video delivery systems, such as link utilisation and latency; by optimising parameters such as these, it is possible to improve system performance, in

order to meet the requirements of specific scenarios.

Additional information on this area will not be included, due to the concepts expressed only being partially relevant for the real-time transcoder placement investigated within this thesis; additional relevant information in relation to the optimisation of transcoder placement can be found in Section 2.8.1 and Chapter 4.

2.3 Cloud Technology

Cloud technology has seen a large adoption in the past 5 years [44], with data centres increasing both in number and size to support the rapid growth. All chapters of this thesis assume cloud technology is part of the network scenario. A few well known cloud service providers are Amazon AWS [45], Microsoft Azure [46] and Google Cloud [47]; these all present similar services with storage, compute and other cloud resources available. Amazon AWS, for example, provides services such as Amazon EC2 [48], which provides compute resources hosted in the cloud and Amazon S3 to provide a cloud based storage solution.

It should be noted that there are many other cloud service providers, but some utilise one of the previously detailed providers such as Amazon; Dropbox, for example, uses Amazon AWS for storage of client data [49], allowing them to focus on their software products rather than their storage infrastructure. The use of cloud services in this way allows new services to build rapidly expanding businesses, without the initial financial investment of costly hardware infrastructure that used to be required when supporting such a business. Along with the saving of initial investment costs, using cloud services also reduces running costs; these costs relate to maintenance of the infrastructure, as well as other factors such as data backup management and replacement of out of date or faulty hardware.

As detailed before, the ability for services hosted within the cloud to scale according to load, is one of the main factors that have made them popular for hosting web-based services and content. The importance of these services is due to the variation of load that can be placed on these services, which can fluctuate from minimal usage to overloaded in a relatively short period of time. The fluctuations in load can be influenced by the demographic of users changing, based on global daily cycles. The peak in load on a standard system would have to be provisioned ahead of time, meaning resources would be kept as unutilised when the load is minimal; over-provisioning resources in this way would be very inefficient in terms of resource use. Furthermore, at times, usage could also extend beyond the over-provisioned resources, causing reduced performance or even dropped requests. For this reason, the use of cloud resources becomes an ideal solution, since resources can be adapted based on current load, with an almost limitless pool of resources available to process unanticipated peaks in demand.

Additionally, it should be highlighted that cloud service providers maintain data centres in multiple locations around the world; these multiple locations allows latency requirements to be met, along with any restrictions on client data placement that might also be required. Client data placement might not seem like an issue other than for latency requirements, but it is important to note the restrictions that governments and institutions enforce for placement of user data outside of certain countries.

Buyya *et al.* [44] present a detailed analysis of cloud technology; they focus on providing an in-depth analysis of cloud services, including a comparison of service providers as well as explaining the difference between clusters, grid computing and cloud.

2.3.1 Virtual Machines

Virtual machines are an integral part to cloud technology, as they allow physical resources on a single machine to be separated and multiplexed for use by multiple isolated operating systems [50,51]. This partitioning of resources achieves efficient resource utilisation, by allowing underutilised hardware resources to be consumed by additional virtual machines. This is a desirable optimisation, given that at any given instance, a standard machine is generally not fully utilising all its available resources [52,53]; further to this, it is also important to highlight that without the described Virtual Machine (VM) scenario, the unused resources can be considered wasted and can remain unused for extensive periods of time. Unutilised resources are considered an undesirable characteristic, due to the cost of both maintaining and powering them; additionally, any unutilised resources can be considered an unwanted surplus of hardware resources, which would have been financially costly to initially purchase. An alternate solution to address this resource usage issue would be to install further applications onto the machine, in order to fully utilise all available resources at any given time. However, this provides no separation between applications, making management a complex task; this method also does not allow an easy solution for managing each application's resource usage. Further to this concept, migrating an application from one physical machine to another, can present compatibility issues with hardware, which might even require recompilation of the software due to varying hardware resources. It should be noted that isolation of applications and data, in this case, is important for many reasons other than migration issues, with one main reason being security concerns. Without isolation of applications and data, it would not be possible to safely allow access to the physical machine to multiple clients, as it would be possible to influence and view other clients services and data. It should be highlighted that VMs are assumed to be utilised within a number of the later chapters, but they are used specifically in Chapter 5 as a main component of the presented system.

VMs present an ideal solution to the described problem of efficient resource utilisation; VMs allow virtual resources to be presented to an operating system, so that multiple operating systems can be run on a single physical machine. These virtual resources can be chosen during configuration of the VM, in order to customise a machine with the specific characteristics required for a given task; these virtual resources usually consist at a minimum of Central Processing Unit (CPU), RAM and Storage, but also include many other virtual devices to emulate chipsets, USB, GPU and other system devices. Because the resources presented to the guest operating systems are virtual representations of physical hardware, these can be easily replicated on another physical machine; the easy replication of virtual resources makes the migration of VMs an easier task, only requiring the duplication of configuration and system image data onto the new host machine; this is on the basis that the destination host machine is running the same virtualisation software. VMs also allow easy management in terms of resource usage, as it is possible to either define specific characteristics for a specific VM, or allow things such as RAM to increase when required. Allowing memory expansion on VMs can cause additional problems [54], especially when allowing

the memory of a physical machine to be overcommitted [55]. However, there are a number of methods available to help manage memory overcommitting issues, many of which are discussed by Li [55], who presents an in-depth analysis of both the problems and solutions associated with memory overcommit strategies.

There are various virtualisation platforms (also known as hypervisors), some popular choices include KVM [56, 57] and VirtualBox [58, 59]; it should be noted there are a significant number of other options available, as well as bare metal platforms and container virtualisation such as OpenVZ [60, 61] which offer similar guest OS virtualisation. KVM, in particular, is popular for Linux users, as it provides a well-tested platform for virtualising multiple operating systems. VirtualBox is popular with both Windows and Linux users, as it has a very good GUI for creating and managing VMs. The work presented in Chapter 5 utilised VirtualBox, due to its ease of use and command line control mechanisms, which made managing the virtual machines a simpler task.

2.3.2 Virtual Machine Migration

With the advancement of cloud technology and VMs, it has become crucial for the movement of VMs between machines to enable efficient use of hardware, as well as to meet performance targets. Although the specifics for the VM migration is not crucial in Chapter 5, due to the system not being delay sensitive, it is important to detail the existing systems designed to migrate VMs to allow the concept to be recognised as a novel contribution.

There are several reasons why it may be desirable to migrate a VM from one machine to another [62], some of these include:

- Load balancing across servers to improve available resources available to VMs.
- Reducing latency for client applications by moving VMs closer to the requesting clients location.
- To enable server maintenance to take place without VM downtime.
- Service Policy restrictions requiring the movement of VMs due to either performance requirements or location restrictions.
- Hot spot mitigation, which is defined as when a host server is oversubscribed and unable to provide the resources requested by a VM.
- Consolidation of VMs to minimise the number of physical host machines required, enabling excess host machines to be powered down to conserve energy.

Examples of server load balancing and server consolidation can be seen in Figure 2.2, which presents a clear view of their objectives.

2.3.2.1 Migration Decision

Another important factor to be considered with regard to VM migration, is the decision of when to migrate. Table 2.2 presents a simplified high level view of some of the options available. These can

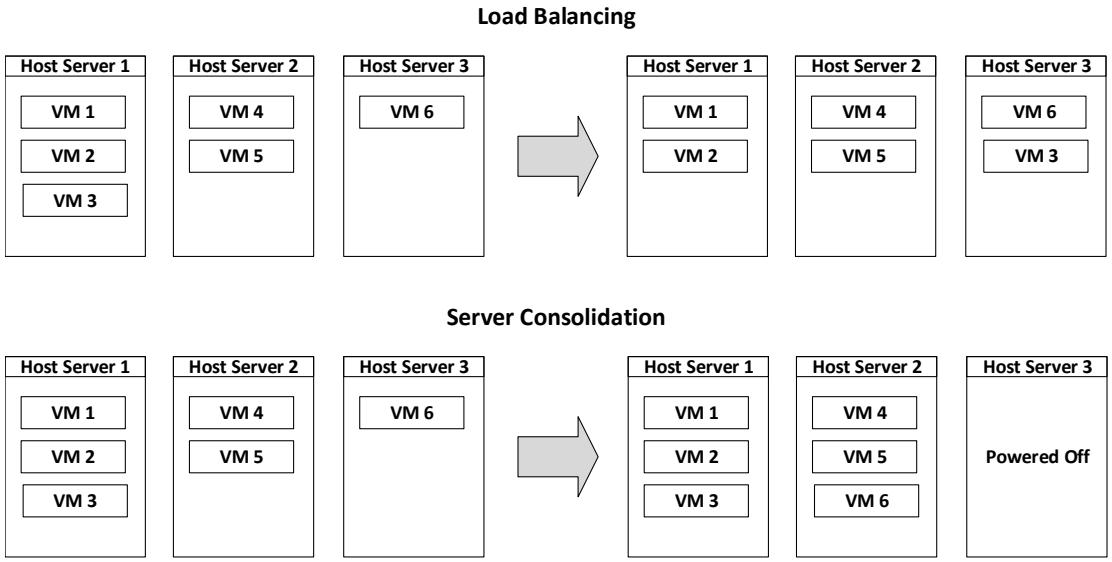


Figure 2.2: Server load balancing and consolidation objectives.

be considered in addition to the idea that certain events can be monitored and used as a migration trigger. These events can include detection of hot spot probability, spare capacity limits across servers and the addition or removal of either host servers or VMs. Additionally, a further influence of migration can include a time based approach, with certain VMs being migrated at certain dates and times, which may be set to coincide with clients day and night cycles or holiday periods.

Objective	**Reason to Migrate**	**VM to Migrate**	**VM Destination**
Server Consolidation	Large number of underutilised host machines	VMs located on lowest utilised hosts	Hosts with high usage but available resources
Load Balancing	Prepare for increase in VM resource use	VMs from overloaded servers	Underutilised host servers
Hot spot mitigation	VM resource requirements not able to be met	VM requiring additional resources	Server with capabilities to support requirements of VM

Table 2.2: VM migration scenarios [62, 63]

2.3.2.2 Migration Mechanisms

There are two types of migration system, offline and online/live [64]. Offline migration occurs when the VM is not functioning and is in a powered off state. Online migration, also known as live migration, is the process of migrating a VM while it is currently switched on and operating as normal; this migration type is far more complex than offline migration, as issues with memory state transfer and synchronisation of running state are difficult to solve. Further discussion on migration mechanisms will focus on the live migration of VMs, as the intent of the presented work is to solve dynamic migration while the system continues to function.

There have been many developed mechanisms for live migration of VMs, as it is desirable for people to improve this system for specific applications. It should be noted that in all the cases of migration mechanisms, there are certain situations where VM storage does not require migration; a scenario to support this would be a migration operation within a local data centre, where the VM image data is located on a shared storage medium that is accessible to both the source and

destination machines. The shared storage approach allows for a relatively rapid migration, as only the VM state is required to be transferred and not the large quantity of data relating to the VM image. Another possible scenario that would reduce the transmission cost, would be the use of a standard VM template being readily available on all machines; if these types of templates were used as a base for the VM to be migrated, only the differences in the image data would need to be transmitted during migration. Beyond these types of scenarios, both the VM image data and state are required to be transmitted during the migration process.

Most of the currently developed migration mechanisms conform to the following categories:

- Suspend and Copy: This method suspends the VM on its source and then begins to transfer data and state information to the new host. As detailed before, the migration process can be significantly improved if shared storage for the VM image data is located on a shared medium, but is generally still slower than other methods. Without using a shared medium for the VM image data, this method could not truly be considered a live migration scenario, due to its time extensive transfer process where the VM would not be accessible.

- Pre-Copy: This method transfers VM data iteratively while it is still running on its initial host. Once the data transferred is deemed sufficient, the VM is suspended on its original host allowing the remaining state data to be copied to the destination; after complete transfer of all state information, the VM can then be resumed from its suspended state [65]. This considerably reduces the time required that the VM is suspended, compared to the standard suspend and copy method.

- Post-Copy: This method initially copies a limited amount of VM data and state information to the destination, so that it can start operation as quickly as possible. Remaining information is then fetched on demand from the original host as needed. This has the risk of causing system faults if information is not found when requested, but generally provides a faster migration time than using a pre-copy solution [66]; there are, however, many mechanisms for mitigating some of these migration issues, Hirofuchi *et al.* [66] discuss many of these within their work.

Although these categories broadly encompass the majority of migration techniques, there are a few that would need to be positioned outside of these categories.

It should be highlighted that an alternative for scenarios which did not require live migration, would be to shut down the VM before the migration event; this method would then only require the transfer of the VM image and configuration, not the memory or state information. This could be further improved using the shared storage method described before, requiring only the configuration to be copied across to the new location.

Another issue that should be highlighted is the importance of transferring the network along with the VM during migration; however, this is not essential in all cases, due to some VMs not relying on a network connection. If a VM is providing a network based service, it is important to ensure continuous operation for this service; to allow a network based service to operate during migration, requires the network to either be modified or migrated along with VM. Network migration does not present significant issues if the VM is migrated within the same network infrastructure [65], as standard networking protocols will be able to automatically update the

path to the new VM location given enough time; however, for more complex migration scenarios where the VM is migrated to an external network, more complex configuration is required to maintain user connectivity with the client. There are mechanisms available to achieve these types of external network migrations, with some utilising tunnelling protocols [67, 68] or proxy servers, to route packets destined for the VM to its new location [69, 70]; one other additional method of achieving this includes the use of packet header manipulation [71], which allows the VM to be reached even after it has been given a new IP address. However, with the recent advancements in SDN technology, it has been possible to achieve improved traffic management and manipulation within cloud environments [72–77]. One main benefit of utilising SDN for migration systems is that redirection and packet manipulation during migration can be configured using a software based approach. Keller *et al.* [78] present LIME (LIve Migration of Ensembles), which is a system that allows the migration of an entire network along with its hosts. LIME achieves this by cloning OpenFlow switches along with the VMs, in order to maintain connectivity between the hosts on their local network. However, it should be noted that additional configuration is required to enable communication outside of their local network. It can be seen with LIME, how SDN provides a platform to simplify and improve the management of the network when migrating network dependent VM resources.

Current migration mechanisms, however, do not consider the scenario of migrating a VM performing live transcoding. Chapter 5 aims to address this gap in knowledge, with further discussion of transcoder migration detailed in Section 2.8.

2.3.2.3 Migration Metrics

To quantify the performance of a live transcoder migration, it is important to determine the metrics associated with this process. One of the main metrics often used is the total time taken for the migration to occur, this includes all processes during the migration from the initial migration request to the starting of the VM on the destination machine. This can be influenced by a number of factors, but is usually the product of the size of the data to be transferred with the network resources available to transport this information to its new host. Another important metric is the downtime of the VM where the user is unable to access or utilise the VM; this delay is usually caused by the suspending of the old VM to allow copying of the last remaining information to the new VM. To achieve improved performance in standard migration scenarios it is important to minimise both these metrics [64]. Other factors such as data transfer, network utilisation or server load can also be used as metrics in certain scenarios.

It should be noted that the downtime metric is crucial for the system in Chapter 5, as reducing this value provides an improved video quality during switchover; whereas the total migration time is not significantly influential on the performance of the system, as long as it is maintained at realistic values. This will, however, be explained further in Section 2.8.

2.4 Network Architecture

The work presented later in the thesis investigates scenarios which involve a number of network architectures, along with presenting an improved network architecture design in Chapter 6. The

scenarios investigated involve the application of different architectures, within the context of delivering real-time UHD video to multiple users. This can be seen as an important area of research, due to the recent and estimated adoption of UHD content described in Section 2.1. However, the current Internet architecture is not designed to allow streaming video services to operate at the bitrates required for real-time UHD video [1, 2]; this presents a significant content delivery issue for service providers [79]. Many adaptations to the original design of the Internet have been successful at allowing a multitude of media types to be transmitted [80]; this is a significant extension of the original basic text delivery, that was intended with the introduction of packet switched networks such as ARPANET [81]. However, even with current advances in switching hardware and link capacity within the Internet, traffic produced by UHD video consumes significantly more capacity than previously seen, requiring a more extensive redesign of the network architecture. Unfortunately, upgrading large areas of the Internet infrastructure is not a feasible endeavour, especially for an end-service provider delivering real-time UHD content. The distinction between an end-service provider and a network service provider needs to be clarified, as an end-service provider only provides a service that utilises the network (e.g. email, video, etc..), whereas a network service provider specifically only provides the network connection and Internet connection services; this is important due to the ambiguous term of service provider, sometimes being used to describe a provider that provides both these services. The inability to update large sections of the existing Internet architecture presents an issue that has several approaches. One such approach is the use of improved architecture design and network technologies, with the aim to reduce network utilisation when transmitting UHD content; this may include technology such as application layer multicasting or content delivery architectures, such as P2P and CDN, but could also be a combination of these to further improve their efficiency.

Another possible approach is the use of a single operator network managed by the content service providers to deliver content to their clients; this is one of the scenarios assumed in a number of the presented chapters, as it provides the best solution to the scenario of transmitting real-time UHD video to multiple users. Using a single operator network allows complete control of the network for traffic optimisation, as well as allowing technology such as OpenFlow to be implemented throughout the network; this level of control would not be possible when using the Internet as a delivery platform. A single operator network would perform best when configured over dedicated light paths through private networks; however, it would be possible to use tunnelling through the Internet to achieve a limited version of this type of network. Using the public Internet even with tunnelling is not an ideal solution, as it would have the latency and congestion issues you would expect with a shared resource infrastructure, with little or no QoS control. This is why it is important to use an architecture implemented using dedicated links, to ensure both latency and link capacity constraints are minimised. The network architectures for delivering real-time UHD content that are presented later, are not designed to replace the current Internet; they are designed to be deployed alongside current technologies, as an aid for specific applications [1, 2]. A single operator network is a realistic assumption in this case, as a single provider often delivers video service to its customers either through unique services (e.g. BT Vision [82]), or through a closely linked CDN architecture (e.g. Akamai [83], YouView [84], etc..); CDN architectures of this type are defined later in Section 2.4.2.

The following subsections will give an overview of some of the network architectures currently utilised for content distribution; it should be highlighted that only the distribution of either stored or live video content was considered with these architectures, other types of content distribution is not discussed.

2.4.1 Standard Client-Server Architecture

Before the introduction of large scale rich media content on the Internet, many services on the Internet were hosted from a single source server which handled all client requests [85]. This presented scalability issues when confronted with the introduction of high bitrate content such as video. The only solution using this type of architecture is to increase the resources available to the server, in order to continue reliably serving clients; this may include increasing Random Access Memory (RAM), adding additional network interfaces, adding faster storage medium or improving the compute power of the CPU. However, this is not a scalable model in today's media rich environment, with video traffic being one of the main contributors to network traffic [3–5]. There is no feasible way to support the reliable delivery of HD video content to tens of thousands of users from a single server, as it would surely exceed either its computational resources or its network capacity. This is the basis behind the design and application of other network architectures such as CDN and P2P networks, which aim to overcome these issues using multiple servers to distribute the load.

2.4.2 CDN

Moving on from standard client-server architecture, CDN architectures use a collection of servers with replicated content to load balance user requests; this allows demand for content to be scaled far beyond what would be possible when using a single server. CDN systems have been an invaluable tool for content delivery within the current Internet [80, 86], providing both website content as well as media content such as video. Along with the capability of balancing user load across servers, CDNs are also capable of providing geographically close server locations for clients in order to reduce latency; these geographically close servers can provide faster response times when content is requested, as well as reducing traffic within the network as a whole, due to the reduced distance the content has to travel through the network. Another aspect of CDN systems that make them an ideal solution for content delivery, is their fault tolerance, which allows one or more servers to fail, while still being able to distribute content from the remaining servers; it should be noted, however, that depending on the system configuration and the specific DNS redirection process being used, this may not always be the case.

To distribute requests and direct clients to their closest server, Domain Name System (DNS) redirection is used to provide the client with the optimum server closest to them [87]. DNS redirection is explained in more detail in Section 6.6, where an alternate implementation is presented.

This is only a brief summary of CDN technology and a more detailed analysis of the area is provided in Chapter 3; additionally, for extensive information on CDN and P2P systems for content-centric delivery, it should be noted that Passarella [80] presents a very in-depth and detailed survey in this area.

2.4.3 P2P

Similar to CDN systems, P2P networks distribute content on multiple machines, which can then serve clients when they request specific content. However, in contrast to a CDN system, the servers that cache the content are not all owned by the service provider and actually include the clients consuming the content; this important distinction is one of the most significant differences between P2P and CDN. A simplified example of how P2P networks operate is as follows: as a client consumes content from a service provider through P2P protocols, they are effectively building a local cache of the content on their own system, this, in turn, can then be accessed by other clients requesting the same content. The caching system has been a significant research area within the P2P field [88,89], due to the storage constraints at clients, which forces the optimisation of cached content; further information on P2P caching can be found in Section 3.3.2.

While the scalability of P2P networks might seem attractive to content or network providers since it seems to require little financial cost, P2P networks do present a number of issues. The first major problem is reliability, as at any given time nodes can leave the pool of peers; this transient behaviour of the nodes makes content delivery unpredictable, creating issues with reliable delivery of content. The extent of this issue, of course, depends on many other factors, as if a node leaves the P2P network unexpectedly, there would most likely be other nodes to take its place and meet the demand; however, there is always the possibility that there would not be another node with the full content cached, or that the remaining node did not have the resources to provide the content within an acceptable time frame. The second major problem with P2P networks is that content sources, the major studios, do not allow unmanaged caching, particularly on users machines, due to risks of piracy [90, 91]. Additional issues associated with P2P technology as a content distribution method are discussed by Rodriguez *et al.* [92], who focus on the feasibility of commercial content delivery through the use of P2P technology.

An important aspect of P2P technology is the content and peer discovery mechanisms involved, these are briefly described in Section 3.1.1. This is only a brief summary of P2P technology, as a more detailed analysis on P2P networks is provided in Chapter 3 where it is more relevant.

2.5 Switching and Control Systems

Network control systems are fundamental in transporting content across both private networks and the public Internet, as they set up paths through the network for content transmissions, as well as determining the routes that traffic traverses. Current implementations for setting up large content streams along private single operator networks, require future knowledge of a content stream; this allows the network path to be configured ahead of transmission time. This manual pre-stream configuration is in contrast to the dynamic routing and path setup that are provisioned within the Internet. To achieve dynamic path setup and routing, a number of layer 2 and layer 3 protocols are used; these protocols allow the setup and calculation of routes through the network, as well as the automatic routing of traffic as it traverses network devices. It should be noted, however, that layer 2 paths are often engineered to follow layer 3 routes through the network. These control systems can be broadly separated into two main categories; ones that set up the virtual paths for transmission and those that determine the routes for traffic along those paths.

2.5.1 Path Setup Control Systems (Layer 2)

Layer 2 path setup is one of the crucial elements to any network, as it provides the ability to establish routes across interconnected physical mediums. The layer 2 paths are configured at the networking devices to route traffic from one port to another, based on protocol dependent parameters. Paths through the network can be configured on demand for specific scenarios, or the can be configured dynamically using certain protocols; these protocols can work at varying degrees of granularity, such as using the source and destination Media Access Control (MAC) addresses in the Ethernet protocol headers for a fine granularity approach, or using a coarser granularity such as the incoming port on the network device. There are a number of layer 2 protocols currently in use within the Internet and some private networks, some of these include: Multi-protocol label switching (MPLS) [93, 94], Carrier class Ethernet [95] and legacy systems such as Asynchronous Transfer Mode (ATM) [96] and Frame Relay [97]; it is important to highlight that some of these can also be classed as being between layer 2 and 3, due to them not conforming to a standard definition of one of the OSI layers [98].

2.5.2 Routing Control Systems (Layer 3)

Layer 3 control systems determine the location of the destination for any given traffic, which is then used to select the optimum route from the source; this is performed using a number of mechanisms, including algorithms such as Dijkstra [99], as well as rules set by Service-Level Agreements (SLAs) or security restrictions. This functionality can be further expanded by allowing the destination discovery to be influenced, allowing the selection of an optimum destination server in order to load balance across multiple servers. The routing functionality performed at networking devices is achieved with the use of layer 3 routing protocols, which perform calculations to route traffic to their destinations; some examples of these protocols include: Open Shortest Path First (OSPF) [100], Routing Information Protocol (RIP) [101], Enhanced Interior Gateway Routing Protocol (EIGRP) [102] and Border Gateway Protocol (BGP) [103, 104].

2.5.3 Assumed Switching Architecture

The main area that this thesis will investigate, is the possibility of using the OpenFlow platform to develop and deploy sophisticated control software for controlling specific content streams. The software will use custom algorithms and flow rules to redirect traffic to optimum destinations and transcoders where required, as well as optimising the use of other network resources such as link capacity. This will improve the content delivery time and reduce bottlenecks that are present in the network. The systems presented are designed to be used on a fully OpenFlow enabled single operator network, not for the current Internet infrastructure; this assumed architecture provides a system where layer 3 routing protocols are not required, as the OpenFlow controller will manage all traffic routing functionality. For this reason, further details on layer 3 protocols will not be discussed, as they are not within the research area of the presented work. It should be noted, however, that OpenFlow does provide a system where these protocols can function without issue, so layer 3 protocols could be utilised for unrelated traffic passing through the network, without being affected by the OpenFlow rule tables.

2.6 SDN

Software Defined Networking (SDN) has become an increasingly popular technology for research and application, both in academia and industry; this is especially true with regard to cloud based environments, where the network architecture needs to be rapidly modified to meet new requirements [105–107]. The increase in popularity has brought significant innovation into the SDN field of research, as a requirement for fast application driven networking becomes a desirable characteristic. SDN in its broadest sense, is a separation of forwarding and control within networking hardware [108, 109], but it also includes a large subset of features. The basic premise of SDN, is the configuration of network resources using a software implementation, which presents a programmable network where a centralised controller can influence the paths of packets in a network [108]. Moreover, SDN presents a basic three layer design, in which multiple applications can influence the control of the network through the controller layer; which can also be described as the network OS layer. The control dictated by the controller application then filters down to the forwarding layer, where flow rules are set on network hardware; a simplified view of this three layer design can be seen in Figure 2.3. It should be noted that a large group of network operators and vendors have founded the Open Networking Foundation (ONF) [110], with the aim to promote and develop SDN technology in industry. This collaboration of organisations has provided additional support to the development and standardisation of SDN technologies such as OpenFlow [111], which is detailed in Section 2.6.1.

Before the introduction of SDN, network administrators had to configure complex network scenarios using basic configuration commands using a diverse collection of Command Line Interfaces (CLIs); this presents many opportunities for configuration errors, along with many restrictions on what could be achieved. These restrictions on the functionality are mainly due to the limited configuration commands available within standard networking equipment, which often required network policies to be designed around command options available within specific network hardware [107]. These restrictions have made SDN a desirable technology to investigate, given its ability to bypass standard networking protocol functionality and provision traffic as defined within a given policy. This configurable design has also allowed networks to be optimised beyond what would be possible with networking hardware using standard L2/3 control protocols [112]; SDNs has allowed improved control over network traffic, improving the optimisation of network traffic and minimising delay. It is also shown by Agarwal *et al.* [112], how that even partial implementations of SDN within a network, can produce benefits in optimisation. These optimisations are also partly due to the centralised control provided by SDN, which enables an improved overview of the network topology when performing optimisation calculations. One example of a large industry taking advantage of SDN for its configurable design and optimisation characteristics is Google [113], who have implemented SDN technology between their data centres [114].

Initial implementations of SDN technology introduced questions on scalability, due to the use of a software based centralised control model. These were mainly focussed on the performance of controllers, with initial implementations having limits on how many flow initiations they could process [115]. However, these initial concerns have been mitigated through improved design and testing of more modern controller applications [116, 117], showing that the scalability of SDN should not be considered any worse than a standard networking scenario.

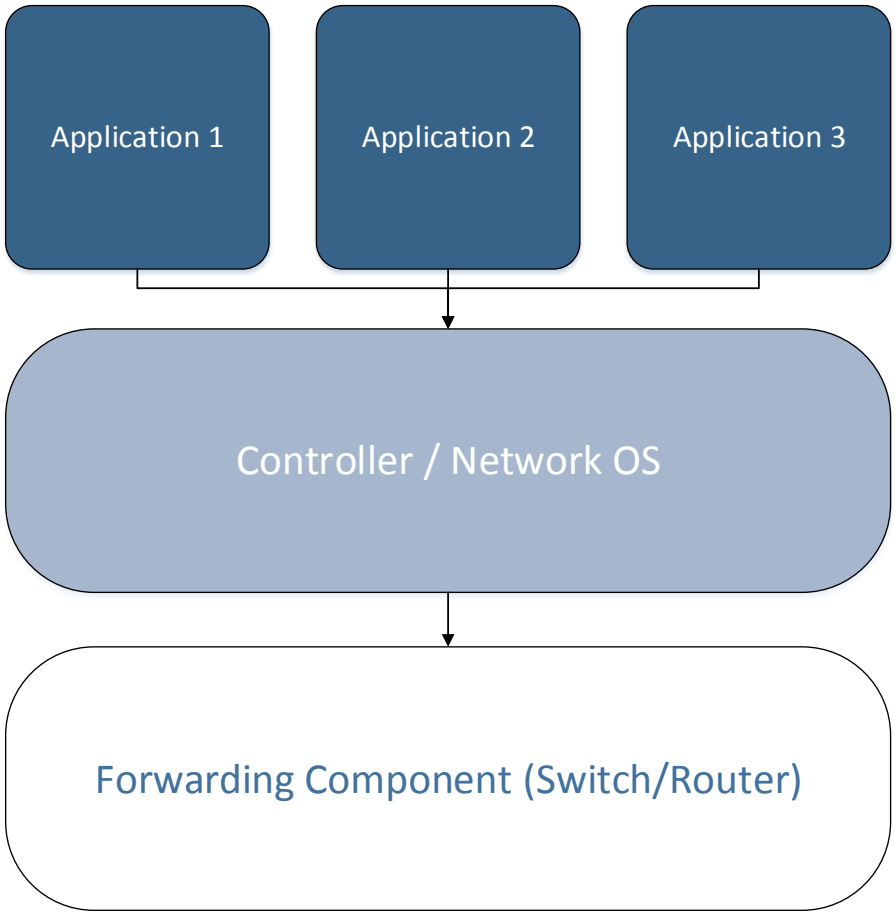


Figure 2.3: SDN platform design.

It should be noted that further subsections will reference and detail elements in relation to OpenFlow rather than SDN, this is due to the use of OpenFlow within the presented research; however, it is important to note that OpenFlow itself is a part of SDN and not a parallel or separate entity.

2.6.1 OpenFlow

OpenFlow is a standard that allows the application of SDN principles, with the data path and the control mechanism separated [105, 118]. The OpenFlow standard also includes the OpenFlow protocol, which handles communication between the controller application and the networking hardware. The controller application can be thought of as the network operating system, providing switching decisions to multiple OpenFlow enabled switches from a centralised location. However, it should be noted that this method of control does not significantly impede the performance of the network devices capabilities or efficiency, which would usually be a concern with a centralised control paradigm; it achieves this by configuring flow rules in the network devices both reactively and proactively. In this way, the controller can assign rules before traffic arrives at the switches to improve initial performance, as well as assigning new rules when unrecognised traffic arrives. It is important to highlight that unrecognised traffic can be defined as traffic that does not

match any rule currently in the switches flow table. It should also be noted that only the first instance of unrecognised traffic causes the controller to be queried; once the flow rule has been configured in the switch for that given traffic, it no longer requires additional communication with the controller, as long as further traffic matches the criteria of the configured flow rule. This method of traffic and flow management provides a very efficient mechanism, whereby after the initial insertion of the flow rule for a given traffic flow is configured, switching is performed by the network hardware without software intervention; this mechanism allows line rate performance that would usually be expected in a standard L2 switch [118]. There have been several studies on the performance of OpenFlow switches with pre-configured flow rules [119, 120], which all present similar conclusions agreeing that OpenFlow is able to maintain a performance similar to or better than standard routing mechanisms. OpenFlow can also achieve similar performance to standard switching hardware when its flow tables are pre-configured; however, it is noted that when OpenFlow flow tables become excessively large, performance does suffer when compared to standard switching mechanisms [120].

It should be noted that there are circumstances that would significantly affect the performance of this separation of control and forwarding; this is discussed by Curtis *et al.* [121], who provide an example scenario based on high performance networks, where fine grained control is being utilised; in this example scenario described by Curtis, a high volume of rapidly changing unique flows can be expected, requiring higher than usual controller input. The high volume of new flows can cause issues where the switch and controller communication is not able to accommodate the high rate of unrecognised traffic; this, however, is explained to be caused mainly by the combination of the limited CPU on the OpenFlow switch and the connecting network capacity. Curtis *et al.* [121] describe being able to process 275 flow configurations per second, however, it should be noted that OpenFlow has improved significantly since this experimentation was performed and other research states figures far beyond this value [119, 122]. Further issues relating to controller scalability are detailed in Section 2.6.3, along with current research focussing on addressing those issues. Apart from the limitations that controller communication may cause, it can be seen that using proactive flow insertion can provide an OpenFlow configured network with improved scalability [122], while providing the centralised control to enable a truly flexible network configuration.

Another benefit of the centralised control of OpenFlow, is the reduction of overhead that would previously be caused by routing protocol communication; however, whether the traffic produced by the OpenFlow controller communication would equal that of the routing protocols, would depend heavily on the network scenario. Curtis *et al.* [121] describe in detail the overheads associated with the controller and switch communication in an effective manner, while also discussing the scalability of OpenFlow in a high-performance network environment. Furthermore, Phemius *et al.* [123] follow on from the assertion that the communication between the switch and controller is a bottleneck, by showing that the latency between the controller and switch can significantly influence the performance of the OpenFlow control system. They also go on to show how the connection method also affects performance, due to the handshake associated with Transmission Control Protocol (TCP) causing delays in flow insertion; however, it is also shown that UDP has its own issues relating to flow control and lost packets. Two specifically developed OpenFlow performance tools that need to be mentioned are OFLOPS [124, 125] and Cbench [126], which

are utilised in various research material to analyse the performance of both the switch and the controller respectively; these, however, are not utilised within the presented work, as the performance assessment of OpenFlow has already been well documented. Following on from this, it's important to highlight the use of proactive insertion of flow rules within the system presented in Chapter 5, as this method makes controller performance less of an issue. However, this is not the case for the system presented in Chapter 6, as it relies on a controller to actively insert flow rules on demand in specific scenarios. Although the performance analysis of the OpenFlow control mechanism in Chapter 6 is not investigated as part of the presented work, it is left as an area of future work and will be discussed further in Section 8.6.

From a research and development view, another main benefit of OpenFlow is the ability to run alongside production traffic, using standard protocols on the same network device; this largely depends on the implementation of OpenFlow within each network device, but the way in which OpenFlow was designed does intend to allow this type of parallel operation [118]. This parallel operation is invaluable for research purposes, as it provides testing platforms at a scale that would otherwise be a costly endeavour. For example, by utilising the dual operation of OpenFlow, researchers can utilise a production network infrastructure, without the risk of influencing normal traffic operation.

2.6.1.1 OpenFlow Standard

The ONF has produced a well defined outline for the OpenFlow protocol, with the current stable implementation of the OpenFlow protocol being version 1.5.1 [111]. OpenFlow was the first standard designed to specifically provide SDN capabilities to network devices [105]. The specification and detail of OpenFlow is too large for inclusion in this discussion, so only a small description of relevant material will be listed, as further information is not relevant to the research carried out.

Originally, switch manufacturers needed to follow strict OpenFlow standards to allow their devices to conform to the "Type 0" OpenFlow switch configuration [118]. A "Type 0" OpenFlow switch provides a set of instructions for the minimal functionality required to be defined as an OpenFlow enabled switch; one of the main specifications states the specific header fields that should be available to be matched, along with the ability to perform an output action on traffic that meets matched criteria. The header fields required for matching in a "Type 0" OpenFlow switch can be seen in Table 2.3.

In Port	VLAN ID	Ethernet			IP			TCP	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Table 2.3: OpenFlow match fields [118]

It should be noted, however, that the development of the OpenFlow standard has advanced far beyond this initial design, allowing a larger set of capabilities and matchable fields. These include the inclusion of IPv6 header fields, as well as packet header field manipulation, allowing header fields to be modified as they traverse the switch. This header manipulation feature is crucial for the system presented in Chapter 5, so it should be highlighted that OpenFlow version 1.2 and above is required for its successful operation [127].

Other requirements for a "Type 0" OpenFlow switch include the ability to perform 4 basic actions to packets arriving at a given port, these are:

1. Forwarding the packet or flow to a specific port/ports, usually at line rate.
2. Encapsulating and forwarding the flow's packets to a controller; this is usually only done for the first packet in a flow for switching decisions.
3. Dropping a flow's packets for security or other reasons.
4. Forwarding the flow's packets through the switches normal processing pipeline; however, this can also be performed using VLANs. This ensures separation of the OpenFlow traffic from production traffic.

By adhering to these basic standards, manufacturers can allow OpenFlow to run on their switching hardware, without disruption to production traffic. As detailed before, this is a great advantage to researchers, as it allows them to test out new protocols and routing techniques on a larger scale, without having to use a dedicated custom built test bed.

Another important characteristic of the OpenFlow standard is the storage of flow rules, which are held within the flow tables of a switch; these also hold statistics for each flow, which can be queried using a number of methods. Originally, OpenFlow switches maintained only a single flow table, however, additional flow tables can now be maintained to extend the capabilities of flow management; these additional tables were introduced in version 1.1 of the OpenFlow standard [128]. Additionally, only ingress flow tables were originally supported, but egress tables are now supported starting from version 1.5 [111].

Further information on the advancement of OpenFlow features across its revisions can be found in its specification documents [111], but are not directly relevant to the presented work.

2.6.2 Hardware

As previously described in Section 2.6.1, it is possible to implement SDN technology such as OpenFlow on standard networking hardware, as long as manufacturers adhere to certain design principles. There are however two types of OpenFlow compliant switches that are available, dedicated OpenFlow switches (also known as OpenFlow-only) and OpenFlow-hybrid switches [111]. Dedicated OpenFlow switches, as the name implies, may only function with the use of a controller application and OpenFlow control, requiring all traffic traversing the switch to be managed by the flow rules put in place by the controller; these devices are considered 'dumb datapath elements' [118], which have very limited functionality without a controller. In some cases, the only functionality without a controller is the inclusion of a default catch-all flow rule, that simply floods packets out of every interface. In contrast to this, hybrid devices enable both flow table based decisions and standard network protocols to manage and direct packets on the switch; these types of devices were briefly discussed in Section 2.6.1.1, with the concept of keeping production and OpenFlow traffic separated. Their hybrid architecture allows network manufacturers to adapt their existing devices, without having to lose existing functionality and without having to drastically modify their design. Although the different traffic can be completely

decoupled from one another, it is also possible to pass OpenFlow flow table routed traffic to the standard L2 switching processes [111].

Another additional function that dedicated hardware switches must include, is the ability to establish a secure channel using the OpenFlow protocol, usually operating on an out-of-band connection; the out-of-band secure channel allows control of the switch to bypass the rules put in place by the controller, as otherwise inserted rules may break the connection to the controller without the capability to regain control.

Pica8-3290 OpenFlow switches were utilised throughout the presented work, as they provided the performance required to handle UHD traffic with header manipulation, as well as the latest in OpenFlow functionality; further information is detailed in Section 5.3.3, where they are first used as part of the proposed migration system.

2.6.3 Controllers

Controllers are an integral aspect to the OpenFlow platform, as they provide decisions on traffic forwarding for OpenFlow enabled switches; as previously discussed, without controllers, OpenFlow network devices have little functionality other than flooding packets out of all ports. It should also be reiterated that controllers usually communicate with multiple network switches using an out-of-band channel using the OpenFlow protocol; this out-of-band communication removes the risk of a flow rule from disconnecting the controller, which would need to be rectified manually. There are a number of controllers currently available, most of which are still in active development with new features being implemented as the OpenFlow standard continues to mature.

One of the oldest controllers that has existed since the first development of OpenFlow, is the NOX OpenFlow controller developed by Nicira Networks [129, 130]; NOX was built with performance in mind, using C++ to efficiently perform time critical operations [130]. However, Nicira Networks released the source code for NOX in 2008, which has allowed it to advance further in development with the help of the open source community [129,131]. During this time, a Python based version of NOX, named POX [129], has been developed; POX, however, will not be discussed further, due to its similarity with NOX. Although NOX is one of the oldest controllers, it is still in active development while being utilised by both the academic community and industry [115, 132].

A more recent controller is the Floodlight controller [133] developed by BigSwitch Networks, which was built on a fork of another controller called Beacon [134]; Beacon was developed by Stanford University, but will not be discussed due to Floodlight effectively being an enterprise replacement [131]. Floodlight is built on Java, which allows easy deployment and mobility between machines; this is in contrast to NOX, which requires compilation based on different architectures. The simple deployment Floodlight offers can be attributed to its ability to run from a simple Java Jar file, providing fast deployment on a large majority of platforms.

OpenDaylight [135] is one of the newest controllers currently in development; however, OpenDaylight actually describes itself as a controller infrastructure containing a collection of applications, tools and services that enable SDN deployment in a large scale network. It is implemented in Java and provides the same mobility benefits as Floodlight; this is important to mention as OpenDaylight was also developed using inspiration from Beacon [136], similar to Floodlight. Unexpectedly, Khattak *et al.* [126] show that OpenDaylight performed significantly worse than

Floodlight in their testing; however, they mention this may be due to a memory leak issue within OpenDaylight. This performance issue is a major problem most likely caused by the complexity of the infrastructure within OpenDaylight, but these issues will hopefully be improved in future revisions.

As detailed earlier, scalability is an important issue when discussing a central control paradigm, which in regard to OpenFlow largely depends on aspects related to the controller. OpenFlow's scalability assessment mainly focuses on the capability of a switch to send flow decision requests to the controller in a timely manner, along with the latency involved in reaching the controller. In addition to these switch to controller communication issues, another important metric with regard to scalability is the performance of the controller application in providing flow decisions to a large group of switches. Although controllers can be located on high performance machines to maintain efficiency during peak load, there is still a requirement to ensure controllers are able to perform at increasingly higher loads; this is especially true as OpenFlow has started to be developed for use on high performance networks. A good performance study of the newer controllers OpenDaylight and Floodlight is presented by Khattak *et al.* [126], where as previously stated, Floodlight performed significantly better than OpenDaylight. Floodlight is able to serve 1335 flow responses per second to each of the 32 switches, whereas OpenDaylight only manages 15; this significantly low value does question the reliability of this testing approach, as it does not seem feasible for there to be such a significant difference in performance based on their similar initial designs. However, the values associated with Floodlight are confirmed to be similar to previously conducted studies, such as the work presented by Fernandez [122]. Fernandez [122] also provides an additional comparison with NOX and POX, showing how NOX achieves improved performance over Floodlight, which is to be expected based on the use of C++ in NOX; POX, on the other hand, performs worse than both NOX and Floodlight. Furthermore, Tootoonchian *et al.* [137] present an improved multi threaded version of NOX that improves its performance by more than 30 times; these advancements in performance provide an insight into how some of the scalability issues related to OpenFlow are due to implementation in the software, rather than limitations with the standard. An additional study of controller applications is presented by Shah *et al.* [138], who provide a performance comparison of a number of controllers; Shah *et al.* also include a detailed description of the architecture, technologies and strategies utilised by each controller.

Although controllers can be improved in their implementation, there have been several other approaches to improving the scalability of controller applications, such as using infrastructure improvements like a distributed design. This is not a new area of study, with systems such as HyperFlow [139] and BalanceFlow [140] being detailed in 2010. It is also noted that there are a number of other more recent research articles discussing improvements to distributed controller design [141, 142].

This section has only provided information on a small subset of available controllers, but these are considered some of the most used in academic research; Nunes *et al.* [108] present a more detailed list of controllers, including in-depth descriptions of their individual characteristics.

2.6.4 FlowVisor

For OpenFlow resources to be provided in a shared environment, such as within the GENI infrastructure [143–145], a mechanism for dividing the OpenFlow resources into multiple isolated elements is required; this is why the FlowVisor system was developed, allowing each "slice" of the network resources to run independently with its own separate guest OpenFlow controller [146–148]. This provides a system that allows a complete virtualisation infrastructure, that can provide VMs with physical OpenFlow resources rather than software implementations such as Open vSwitch (OVS) [149,150]. Previously, users would have to be given control of the entire OpenFlow switch, allowing them to influence other users traffic in certain instances.

The FlowVisor can be compared to a hypervisor, which performs a similar function with computer resources as detailed in Section 2.3.1. The FlowVisor enables efficient use of network resources in the same way that a hypervisor uses VMs to provide efficient use of computer hardware; this efficient utilisation of resources can reduce costs for both the operator and consumer by minimising running costs associated with network resources [151].

The basic design of the FlowVisor is that it operates as an intermediary between the OpenFlow switches and the guest OpenFlow controllers. This means all messages from the guest controllers need to be examined to ensure they are only observing and controlling their associated switches. Using this method, the OpenFlow switches are considered divided into virtual switches, where traffic is separated between each "slice".

It should be noted that the operation of the FlowVisor can be closely duplicated by using either OVS, or Pica8 switches which are also based on OVS. By utilising OVS, it is possible to divide physical network ports into several virtual switches, which are able to be assigned their own controllers. Although using OVS in this way could separate the traffic, in a virtualised infrastructure this may not be suitable, as there is the possibility that certain virtual switches would require traffic to be sent from the same network port.

2.6.5 Traffic Monitoring

There are multiple technologies available to monitor OpenFlow enabled switches and their traffic, some of which include standard packet monitoring tools such as Wireshark [152]. However, other technologies are also available where statistics and sampling of traffic can be directly gathered from the switches. Two of the main technologies available which work in this way, are sFlow [153] and NetFlow [154], which allow collector/analyser software located remotely to be sent the traffic and statistics information. Both sFlow and NetFlow are available within a number of virtualisation platforms, which enables simple and efficient cloud based monitoring; one such network virtualisation tool that includes both tools is OVS [155], which is utilised in Chapter 5.

SFlow is an industry standard for monitoring network traffic [153]. It uses packet sampling techniques to produce statistics on high speed networks, without impacting network or switch performance. Due to the use of packet sampling, SFlow is a very scalable technology, as it can be optimised for certain statistics on a wide variety of networks [156]. These monitoring capabilities would allow it to determine when a transcoder migration would be beneficial, which would be useful for the scenarios presented in Chapters 4 and 5. A similar network monitoring system is presented by Rehman *et al.* [157], who provide detailed information on how sFlow can be used to

monitor traffic within an OpenFlow network.

Alternatively, NetFlow [154] also provides statistics gathering in a similar way. It provides features similar to that of sFlow, with the ability to tailor its sampling mechanism [158] to maintain scalability in larger networks; however, NetFlow also has the capability to collect all the traffic passing through a switch, which is not currently possible with sFlow. Unfortunately, this type of collection can impose scalability issues when there is excessive traffic traversing the switch, not only because of network limitations for sending the statistics, but also because of CPU limitations on the switch [159]. CPU limitations are a common problem with running applications on switches, as they usually have relatively underpowered CPU resources when compared with desktop or server machines [121]. For this reason, this method of collection is not suitable when using high traffic applications, such as streaming UHD media. It should be noted, however, that with some network devices, NetFlow operations are achieved using hardware approaches, which significantly reduces the load on the CPU [159]. It is also important to highlight that NetFlow specialises mainly in L3 traffic, whereas sFlow is designed to work with L2 and above [155]. For these reasons, sFlow is a more suitable candidate for use with the systems presented in Chapters 5 and 6.

2.6.6 OpenFlow CDN

The system presented in Section 6.3 is similar in design to the OpenFlow gateway presented by Liu and Jin [75], who detail a preliminary design of a system aimed at improving VM migration between data centres using OpenFlow. They briefly describe using an OpenFlow switch to perform Network Address Translation (NAT) of packets, in order to minimise routing issues associated with VM migration. However, because the system is heavily focused on providing improved VM migration, it does not consider any other applications or scenarios, such as streaming video services. Liu and Jin also fail to discuss anything more than a preliminary design idea; they neglect to mention any scalability issues relating to the system and leaves a lot of questions in relation to its complete design, as no testing or analysis was performed. The system presented in Chapter 6, however, provides a detailed description of the complete OpenFlow CDN and NAT system. This includes the description of capabilities and features far beyond the system described by Liu and Jin [75], especially in relation to optimising the delivery of video content. Additionally, analysis of aspects such as packet header rewriting is performed later in Chapter 6 within Section 6.7; the analysis of packet header manipulation is presented in order to provide insights into the scalability of the proposed system, which is something that Liu and Jin [75] fail to achieve.

After completing initial research and testing towards the system proposed in Chapter 6, a similar concept to the system presented in Section 6.6 was found to be presented by Wichtlhuber *et al.* [160] in a recent publication. Wichtlhuber *et al.* use a redirection centre local to ISPs to receive content requests, these redirection centres then initiate content distribution from the selected CDN surrogate (cache) to the requesting client. This is similar in concept to the mechanism presented in Section 6.6, however, there is a significant difference in that the system proposed in Section 6.6 does not require a redirection centre. The proposed system does use a centralised controller, but redirection is handled directly at the switches using packet header manipulation, without each user having to contact the redirection server. The removal of the dependence on the controller for each request should improve the scalability of the system, especially in comparison to the

system presented by Wichtlhuber *et al.* [160]. However, this is based on the assumption of not using individual redirections per client request in the proposed system, which is a reasonable assumption based on the use case of delivering video content.

2.7 Application Layer Multicasting

Application Layer Multicasting (ALM) was developed to implement the capabilities of IP multicast technology [161,162] at the application layer, rather than within the network; this is desirable due to the absence of good multicast support in IP networks [163–166]. Furthermore, ALM presents an ideal solution to the complications seen with deploying IP multicast within the internet, as it does not require infrastructure support; ALM can operate without specialised network infrastructure, as it is deployed on end host machines rather than within network devices. One of the issues associated with IP multicast deployment, is the scalability constraints caused by routers having to keep state information for each IP multicast group; unfortunately, scalability is further reduced, as these multicast groups can not easily be aggregated, unlike unicast addresses [164]. However, with ALM working at the application layer, it does not suffer from these issues. Other issues associated with traditional IP multicasting include address allocation, group management, security, authorisation and Quality of Service (QoS) [163].

ALM is used to reduce replicated content being sent through the network, which is achieved by distributing streams to multiple clients at a position closer to the users than the original sender. This permits a single stream to be sent through the network to the multicast enabling node, which then sends the required multiple streams the remaining distance to the receiving clients [164]. This can provide a significant reduction in the overall traffic load placed on the network, especially when considering high bitrate applications such as UHD content.

One example of an ALM technology that has been proposed for deployment within the Internet is End System Multicast (ESM) [167], which involves each user who subscribed to a feed providing the content to another user; it should be noted that this works similar to that of P2P systems. Although this system is able to overcome the issues associated with IP multicast, it does suffer from reduced network optimisation and longer delays when compared to IP multicast [163, 167]. There are numerous ALM protocols that have been developed, each with their own specific use cases and benefits; Hosseini *et al.* [163] provide a detailed survey of a number of available protocols, along with additional information on ALM specifics.

It is important to discuss how ALM can be used in parallel with transcoders, to further reduce link utilisation in the network. However, it is important to highlight the importance that transcoding functions should be performed before reaching multicast enabling nodes, in order to provide effective traffic reduction. Furthermore, software based transcoder applications such as FFmpeg, are able to distribute multiple streams from a single network input stream; this functionality allows FFmpeg to perform the task of an application layer multicasting alongside its transcoding processing. The concept of utilising FFmpeg for these tasks is used within Chapter 5, which presents a system to distribute an UHD stream to multiple clients at varying resolutions.

Additionally, it should be highlighted that the idea of performing multicasting using SDN technology is discussed in Chapter 6, which although is similar to IP multicast technology, is also controlled by an application, making it related to ALM.

2.8 Transcoder Migration

Although there are numerous publications on the specifics of migrating generic VMs, most do not focus on the concept of transcoding and the ones that do only briefly mention it. The proliferation and dominance of streaming video on the current Internet [3–5], seems to highlight a large area of research relating to transcoder placement optimisation and live migration mechanisms that needs to be investigated; the presented work intends to fill this obvious gap in research, to optimise and improve video delivery systems capable of transmitting UHD video. It should be noted that although video traffic is a main contributor to the traffic on the current Internet, this traffic is usually generated from stored or cached content, served from a system such as a CDN [4, 5]. However, with the rise of live streaming applications, there will be an increased requirement for dynamic transcoding resources and improved live streaming architectures.

Twitch [168] for example, has seen a large rise in demand from gamers that want to live stream their gaming sessions [169], in order to interact with their users. This interaction was not previously possible, with methods involving uploading pre-recorded content to online video hosts such as YouTube [170] for later playback. Mobile based live broadcasting applications have also recently become available on modern smartphones, which enable users to stream live content direct from their phone to large online audiences. Periscope [171], Meerkat [172], Kamcord [173] and Mirrativ [174] are examples of these applications, which have recently shown increased demand [175]. These types of live streaming applications operate using limited processing power and a restricted network connection, especially in regard to the mobile platforms; this provides a problem for scalable live streaming, which is why many of these applications use systems such as HLS [30] or MPEG-DASH, which are described in Section 2.1.3.1. However, due to the caching involved with HLS and MPEG-DASH based systems, there is a significant delay of around 10-20 seconds added to the stream [176–179]; this delay can be minimised with configuration of chunk size and cache setting, but still not to the point that it would be considered true real-time streaming. Certain scenarios require true real-time streaming with minimal delay, such as live sporting events where information can appear on social media before the stream has displayed it. Chapter 5 addresses this issue using transcoder migration techniques, which would be ideal for the scenario involving live streaming of a sporting event in cinemas. Chapter 6 proposes a new OpenFlow based CDN architecture to support large volumes of clients for both real-time and stored streaming applications; the OpenFlow CDN would be ideal to support applications such as Periscope, Meerkat and Twitch, without the large delay involved with HLS or MPEG-DASH. More detailed and further information on these concepts and applications can be found in Section 5.2 and Chapter 6.

Further to the information found in Section 2.3.2, the concepts and mechanisms of VM migration can be adapted and extended to the migration of a live transcoder processing real-time video content. The specific research in this area is crucial for the development of the system presented in Chapter 5, which implements a system that can overcome the issues associated with migrating a transcoder that is processing a real-time live video stream. This is something that is absent from current research, as without recent advancements in OpenFlow, achieving a seamless switchover is not possible.

The system presented in Chapter 5 assumes the use of cloud technology, with information

relating to this found in Section 2.3, alongside standard VM migration methods. Furthermore, the scenarios showing reasons for migration in Section 2.3.2, still remain partially relevant in the migration scenario; however, these do not consider the scenario of a real-time transcoder machine currently providing content to clients. This scenario presents additional factors that should be considered, both in determining when and where to migrate, as well as the mechanisms for achieving effective migration without interruption in the video. Further information regarding migration decisions can be found in Section 2.8.1.

Along with focussing on the migration of the transcoder machine, a further area of interest is to migrate the traffic that the transcoder is currently receiving and processing. Network virtualisation is at the core of this area of research, with its adoption in cloud environments being beneficial to aid in efficient resource use, traffic separation and rapid resource redeployment [150, 180]. It is important to note that many existing systems utilise SDN to achieve this network virtualisation [76, 150, 180, 181]; additional standard and SDN based approaches to network migration are discussed further in Section 2.3.2. However, there are a large number of other alternate SDN based approaches that were omitted, due to their similarities with the other described systems in Section 2.3.2 [72–77].

It is important to highlight how the usual metrics for measuring migration performance, do not all apply in the case of live transcoder migration. This is due to the main goal for the migration, being the improvement of the client viewing experience during migration. The performance of the system is therefore influenced by the downtime experienced by the transcoder VM, as this will interrupt the video being transmitted to the client. The downtime in the given live streaming scenario is considered the period in which the VM is not network accessible, which prevents its ability to process the network stream traffic; this differs from standard VM migration metrics, which measure only the actual VMs operating downtime, which is not a sufficient measurement of performance in this case. The network based downtime metric is suitable for any application with its main function being the processing of network data, or applications that are network reliant; these types of applications require network functionality to operate, so this needs to be taken into consideration when looking at migration performance metrics. Liu *et al.* [75] also highlight this as an issue with current performance analysis research, as they also focus on a similar idea of improving migration times for network reliant VMs; however, they do not investigate the scenario of live transcoder migration, which requires a significantly reduced migration time.

Migration statistics such as total migration time do not directly affect the viewing quality of the client, but are still important to consider in relation to the cost and performance to the service provider; this is a consideration that is not investigated extensively with the presented work, as service provider costing caused by migration time would be minimal due to the fast and efficient migration type being utilised in Chapter 5. The migration type used in the completed system of Chapter 5, does not require the transfer of the VM data; the mechanism only requires the transfer of the configuration data to a standard Linux template VM, assuming that the template has the correct tools pre-installed. This migration mechanism was further detailed in Section 2.3.2, where it can be seen that it requires considerably less time to migrate than other migration mechanisms. The short migration time can be attributed to the relatively small amount of data to transfer, which also has the added benefit of minimising traffic during migration. The configuration to

be transmitted in the case of a basic transcoder scenario, could be as simple as the network configuration along with the current transcoding settings; this small amount of information would be enough to start a cloned machine capable of continuing the work of the original transcoder. It should be noted that the tools used in Chapter 5 are often bundled with Linux distributions, so the assumption that a suitable template would be available at each data centre is valid.

Additionally, it should be noted that the concept of distributed transcoding described in this section is not a new area of study, as early works in this area suggested the inclusion of transcoding resources in the network layer forwarding components [182]. However, most of these early works violated the end-to-end principle that underpins the pragmatic deployment decisions of modern IP based computer networks [183]. With the more recent deployment of cloud systems and CDNs, the "end-system" has become a more widely distributed notion that can draw upon other distributed resources to carry out storage or processing. Thus, we can enable this distributed transcoding paradigm without violating the end-to-end principle of contemporary IP networks. Through the use of cloud architectures and SDN, transcoding becomes a flexible, relocatable, resource that is available to the application layer, rather than an addition to network layer processing that is less flexible in nature.

2.8.1 Optimising Transcoder Placement

As detailed in Section 2.3.2, there are a number of reasons as to why it is beneficial to migrate VMs to different locations, be that geographical locations or just to another local server. One of these advantages include efficient use of hardware, by moving resources closer to clients to reduce delay or network load; this principle can also be used to move VMs to areas where the cost of running is cheaper [184]. This presents the basis of an optimisation problem, which focuses on placing a finite amount of resources in optimal positions throughout a given network; this is the concept explored in Chapter 4.

This optimisation problem is a directly related research area to the migration of VMs, as the optimisation would require migration of resources once optimisation had determined locations for the resources. It is assumed as before that the transcoding resources are thought of as VMs, running transcoding software such as FFmpeg. The optimisation of resource placement is not a new area of study, with multiple research articles detailing different approaches; however, most of these relate to the placement of stored content [37,39,40,42,185] or VM load balancing mechanisms [63], without the consideration of real-time video streams and transcoding resources. There are also other related placement strategies relating to CDN proxy placement [86, 186], but they are not relevant due to their stored content use cases. Shevchuk [187] gives a brief look at the different structures of transcoder locations, but he seems to lack any results relating to performance against a set of demands; Shevchuk only gives an overview of the available transcoder positioning structures and doesn't in any way attempt to model or test them. Further to this, Colle *et al.* [3] present an argument that placement of transcoding resources need to be optimised jointly with the distribution method, which is an important point to consider.

As stated earlier, the lack of research into transcoder resource placement presents a clear gap in knowledge, which would be beneficial for the advancement of UHD streaming. This is an important research area considering the dominance of video traffic in the network, coupled with

the rise of consumer focused streaming applications. However, it is important to highlight that the optimisation of transcoder placement differs substantially from the optimisation of stored video content. For example, stored content would usually be replicated at a large number of locations, given the abundance and low cost of storage available at modern data centres; this is in contrast to the high computational cost of transcoding [3], which makes saturating the network with transcoders a financial burden. This mandates provisioning a finite number of transcoders to be placed for streaming applications, which makes optimisation a more complex task. An additional point to consider is the delay tolerant nature of stored content due to buffering at the client; this is especially true once streaming has begun, as the goal is then to maintain continuous playback at the required quality while minimising jitter. Due to this delay tolerant characteristic, it is possible to have improved flexibility in resource placement when compared to delivering real-time content.

The optimisation of transcoder placement would not only reduce the number of streams between the transcoder and publisher, but also place the transcoders in locations which would conserve capacity in the network; this reduction in traffic would reduce blocking in the network, allowing additional demands to be fulfilled. Chapter 4 presents an algorithm that aims to optimise the stated transcoding resource placement problem, showing that it does successfully reduce both blocking and network load; the scenario used for the optimisation analysis, is the delivery of real-time UHD content to a dispersed group of clients, with transcoders also performing application layer multicasting.

Chapter 3

CDN/P2P Compared

Before investigating possible improvements with new and existing video delivery systems, it is important to discuss P2P and CDN architectures. They are both capable of providing large numbers of clients with video content while providing many benefits over single server configurations.

P2P has always allowed great scalability and resilience due to its non-centralised and dispersed node architecture, but the things that make it scalable and resilient also impose some inherent weaknesses [80, 188]. The main weakness of P2P technology is the reliance on unpredictable transient nodes, which are not controlled by any central authority or automated means [189,190]. Peer nodes in P2P networks are unreliable in nature and can become unavailable without warning, causing issues with a data stream [191].

In contrast to P2P, CDNs have a constant set of servers, providing content and services to requesting clients. These high availability servers are fixed and only go offline in very rare circumstances, such as node or route failures. This provides a very reliable service to clients, but CDNs are expensive to operate [189, 190] and create bottlenecks if a large quantity of clients request content at the same instance. Additionally, even though server downtime is rare, if an unplanned outage occurs, standard CDNs do not incorporate built in recovery mechanisms unlike P2P networks [192]. Another potential disadvantage of CDN based systems, is that they can be very inefficient if there are large groups of requesting clients geographically distant from the source CDN servers; this inefficiency is caused by the wasted link capacity consumed by identical content streams being transmitted between the CDN server and the clients. Even though the traffic is identical, the current incarnation of a CDN based server provides each client individually, with no regard for geographical location [193,194]; it is important to note that the geographical awareness statement relates to the content delivery from a single selected server in the CDN, after any DNS redirection has been used to redirect the user to the optimal server. The replication of content is wasteful of network resources, especially considering architectures such as P2P allow clients to cache content for distribution to other clients close to them [80,188]. This can provide a significant reduction in the traffic traversing the network, as well as providing resilience to network and server failure. There are existing systems that can be implemented in parallel with CDN systems, such as application layer multicasting, which aims to reduce the replicated network streams; however, the deployment of application layer multicasting type systems do not have the flexibility or resilience that can be obtained from using a P2P based caching approach.

With both P2P and CDN featuring desirable and non-desirable characteristics, it becomes

clear that developing a system that incorporates the advantageous characteristics of each system together would be a worthwhile area of research. This chapter details the capabilities of both P2P and CDN technology in the context of video delivery, as well as investigating current architectures that aim to combine them in fusion architectures. Furthermore, analysis of the performance of each individual architecture is presented, along with a discussion of possible techniques to improve their performance; this also includes analysis and discussion of observed behaviour, that was used to improve the real-time content delivery systems presented in the later chapters.

3.1 Comparison of Technologies

In addition to the obvious differences between P2P and CDN previously described, there are a number of other variations that make each architecture unique. These variations can be generalised as the components that the fusion architectures aim to utilise, combining the beneficial components of each system to minimise the issues and weaknesses associated with them.

	CDN	**P2P**
QoS	Yes	No
Cost	High Cost	Low Cost
Client Management	Managed	Not Managed
Scalability	High Cost	Low Cost
Server Capacity	High Utilisation	Low Utilisation
Content Control	Controllable	Not Controllable

Table 3.1: Comparison of CDN and P2P Technology [192]

Table 3.1 gives a brief overview of some of the basic characteristics of each system. It can be observed from this that P2P has issues with content control, including authorisation and control over its clients. This is due to the lack of management with P2P nodes, which cache the content locally and distribute it without restriction to any requesting client; the only way to restrict this would be to impose additional security on top of the P2P system, such as content encryption or a separate authentication mechanism. Content control is not essential for all applications, as certain types of content can be freely distributed, but for content which requires licensing and/or subscription, P2P is not a feasible option [90–92]. In contrast to P2P, CDN provides considerable control over its content, as it is managed from a central location, with the option of authentication taking place on the content servers [192].

One major advantage of using P2P technology for content delivery is its low cost, as it requires minimal content hosting and management resources. In principle, only a single server is required to initially begin dissemination of the content, after which, all connecting P2P participants become active content providers themselves. This presents a huge saving in cost when compared with a CDN based approach [88, 166, 192–195], as only a single server is initially required; furthermore, after the content has been downloaded by a significant amount of P2P clients, the initial content server can be removed, since other P2P members can continue the delivery of the content. However, P2P can experience performance issues in relation to this reduced cost, especially when initially providing the content; as without additional P2P nodes to load balance the delivery of

the content, the initial server can become oversubscribed until additional nodes are available. In contrast to this, CDN architectures have a relatively high cost, but have improved performance with the capability to provide varying QoS to different content and/or clients [194]. P2P also has several other performance related issues, including as previously mentioned, that all P2P nodes are transient nodes which can become unavailable at any time; this could interrupt transmissions of content, producing delays while the content is retrieved from another source.

3.1.1 Content Delivery

The mechanism in which clients request and retrieve their content vary between P2P and CDN, and it is important to highlight these differences for analysis of the combined P2P-CDN architectures.

The content delivery approach for CDN based architectures utilises DNS redirection to route resource requests from clients to appropriate servers. Servers may be selected on a number of factors, some of which can include content type, server utilisation or distance from the client. Once the request is received at the relevant server, the content is then sent to the client directly.

P2P has more than one mechanism for performing peer discovery to identify hosts of requested content. One such method is Distributed Hash Table (DHT), which operates to discover content using a key to address hash matching system. The DHT system works on an overlay network, with one of the largest being Mainline DHT [196], which is used for trackers in the BitTorrent system. There are several notable designs of DHT, many of which are analysed by Kelaskar *et al.* [197]; Kelaskar *et al.* provide a detailed comparison of a large number of DHT systems, with the focus on providing an in-depth study of the research area. Further details of peer discovery mechanisms are not discussed, however, Lua *et al.* [198] presents a thorough and in-depth survey of a large number of P2P discovery mechanisms.

3.2 Methodologies

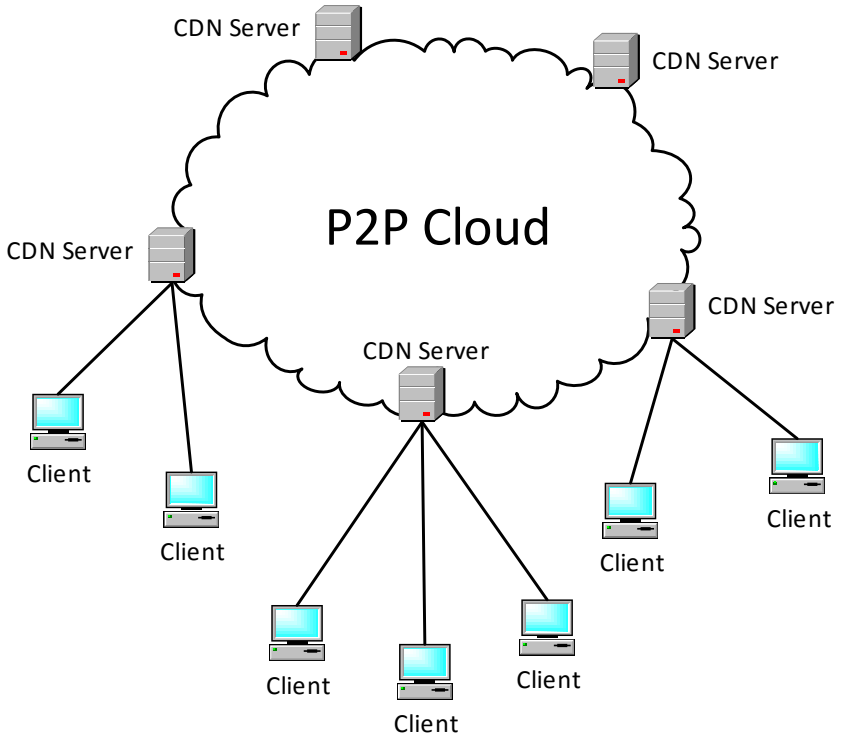
There are several existing approaches to combining the characteristics of P2P and CDNs, all of which aim to achieve a system that performs better than standard CDN or P2P in certain scenarios. One of the consequences of combining aspects of the two technologies is a conflict of design principals, which in some cases can introduce new complications not previously encountered in the individual architectures. To minimise the risk of this occurring, it is important to maintain a clear objective for the new architecture, instead of just merging all aspects of the two technologies together. Some of these existing combined systems will be briefly detailed and examined in this chapter, but some have been omitted due to their similarities.

One method for combining the two technologies is to use P2P technology as the central backbone at the core of the network; using P2P as the core network provides a highly distributed system, which is able to distribute the content efficiently and provide load balancing services [190, 192, 199]. An example of this architecture is shown in Figure 3.1(a). The P2P enabled core also provides improved resilience to node and network failure, allowing other core nodes to replace any which may have become unavailable. With the content replicated and distributed throughout the core in this way, only a major network or machine outage would affect the delivery of content. The described P2P core is also combined with a set of machines which are selected

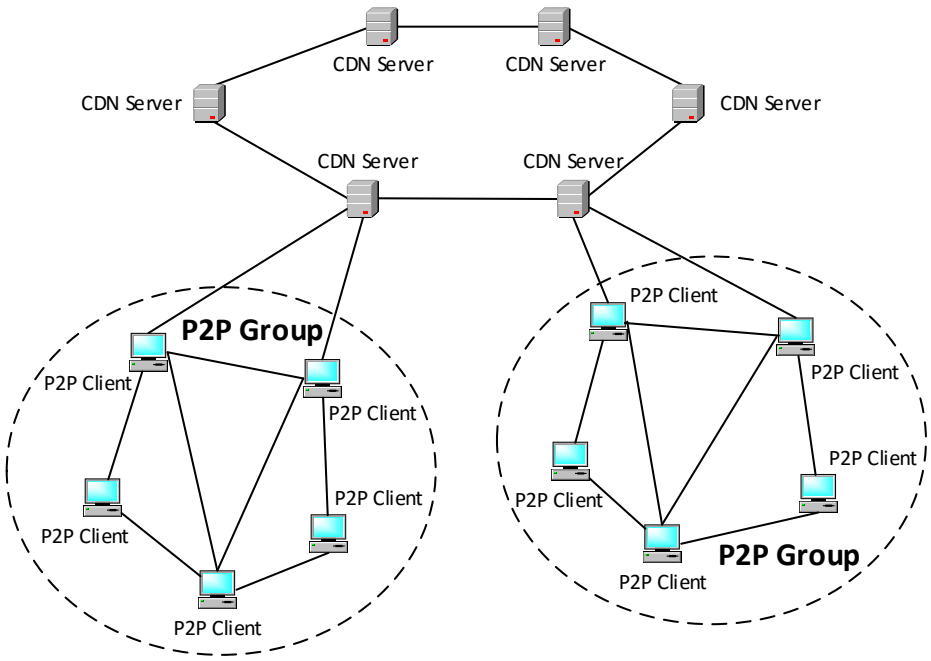
based on performance, capacity and availability; these selected servers act as a type of CDN server, providing content to devices that can not operate alongside the P2P architecture. Typical devices that would be served by the CDN servers would be low powered devices such as mobile phones, which have limited resources and a relatively small power source. Although it would theoretically be possible for these low powered devices to operate with the P2P network, the compute power required would greatly diminish their power supply in comparison to a standard CDN approach. The described P2P core architecture provides a very scalable system, while also ensuring compatibility with older or less powerful devices, such as mobile phones or tablets. Unfortunately, the system relies heavily on the availability of the selected CDN servers, which although chosen due to their high availability, could leave the network at any point without warning since in principle they are considered standard P2P nodes. If a selected CDN server does suddenly become unavailable, it can place increased demand on the remainder of the network while a replacement is discovered and configured; the overflow of demand could cause performance issues, as well as delays for clients trying to access content. It should be noted that if individual CDN servers occasionally fail or disappear, the system will be able to continue operation without much disruption; however, if more than one CDN server starts becoming unstable or unavailable at regular intervals, it will degrade the performance of the network, as other servers are overwhelmed by the influx of clients from the unavailable nodes.

In contrast to the previous fusion method, it is also possible to use CDN servers as the central core of the network. The CDN core approach would then allow groups of clients to converge and form their own individual P2P networks; every client in each of the formed groups would then act as a P2P node, performing media caching to enable them to serve content to other requesting clients in their group [88, 89, 191, 193–195, 200]. An overview of this architecture can be seen in Figure 3.1(b), which shows the formation of the P2P groups. One benefit of this system is that the P2P groups reduce the load of the central CDN servers, as well as conserving network capacity which would be wasted repeatedly sending identical content to individual geographically close clients; it is important to highlight the similarities of this mechanism to application layer multicasting, which performs a similar function to reduce this type of repeated content streams. The CDN core architecture also makes use of the high availability CDN servers to make a robust core system that is unlikely to fail, due to the redundant links and multiple servers. The separate groups of P2P clients can be grouped using several mechanisms, including geographical location, media content, subscriber permissions, as well as many other characteristics that may suit the specific application. The scalability of this architecture is an important issue, but it can be observed that depending on the grouping parameters, the system has the possibility to scale exceptionally well. The excellent scalability can be attributed to the fact that each client entering a P2P group contributes to the delivery of content once it has been cached, which in effect reduces the load placed on the core CDN servers.

The methodologies discussed above have mainly been designed for providing content to multiple subscribers, other methodologies exist for other use cases, but they are not relevant to the presented work so have not been discussed.



(a) A hybrid network with P2P core and CDN edge servers



(b) A hybrid network with CDN core and P2P groups

Figure 3.1: P2P-CDN fusion architectures.

3.3 Existing Fusion Methods

There are several existing methods that have been previously examined by researchers, in this section a brief overview of a subset of these will be examined.

3.3.1 Network Architecture

Several P2P-CDN architectures have previously been developed, most of which follow closely to one of the architecture types mentioned earlier in Section 3.2; the two architecture types can be reviewed in Figures 3.1(a) and 3.1(b).

Both architectures are scalable, allowing a significant number of clients to connect while maintaining performance. Each of the architectures attempts to include different aspects of the parent architectures, unfortunately, not all benefits can be successfully merged and some unwanted disadvantages are introduced. For example, both architectures suffer from a reduction in the content control usually found in CDN systems. This is due to the introduction of P2P processes, which disseminates the content to clients in an unrestricted manner; thus, once the content is part of the P2P system, it becomes hard to manage as there is no direct control over each of the client nodes. The CDN core based architecture in Figure 3.1(b) maintains reliable service and scalability by delegating content delivery to its P2P groups, whereas the P2P core based architecture in Figure 3.1(a) can improve its scalability by delegating P2P nodes as additional CDN servers. Both architectures reduce the cost of content delivery when compared to a standard CDN only architecture, as with the introduction of P2P, the system is able to function with a reduced quantity of dedicated CDN or originating content servers.

3.3.2 Caching Mechanism

An important factor in P2P and CDN architectures is the caching mechanism, which determines what content to store at any given node; this is especially true in regard to P2P nodes, as they usually have limited resources and are unable to store and provide a large amount of content. Although CDN servers usually have significant resources available, these resources are not unlimited, making it necessary to manage stored content using some sort of caching mechanism. This section will mainly focus on P2P based caching found within the P2P groups in Figure 3.1(b), but these caching techniques could also be adopted by CDN based nodes. It is also important when designing the caching process in P2P architectures to factor in the transient nature of P2P nodes, as this can heavily affect the content caching performance. This makes caching media a more complex task, as you have to factor in the current clients accessible on the network at any given moment, to determine what content to cache on specific clients.

By grouping P2P clients together, such as in the architecture presented in Figure 3.1(b), it reduces the complexity of placing optimal cache placements due to the reduced node numbers; this makes the implementation of the grouping mechanism for the clients an important factor in the CDN core architecture. Possible grouping mechanisms that can be used include using geographical location or media content for determining group designation. Grouping the clients based on geographical location has the benefit of reducing network traffic by minimising the hops between serving clients and requesting clients. However, due to the transient nature of P2P nodes,

grouping clients using geographical context can present a significant issue, as the group could be rendered ineffective if most of the clients in a given area go offline for a long period, such as during the night.

There have been several attempts to optimise caching at nodes [37,40,88,193], each with their own merits; they range from random implementations, to advanced heuristic algorithm based methods directed towards specific scenarios. One popular approach for calculating optimal cache placement for video is to use the popularity of the content at that specific time, along with the current presence of other caches of the content within the P2P group [88]. This method ensures client storage and link capacity are only utilised on peers that will provide the greatest benefit to the system. Dan *et al.* [88] present an algorithm that utilises these factors to determine whether a segment of video should be cached at a given client; the algorithm decides when to remove ageing segments that are no longer popular, as they have a high probability of not being accessed again.

The algorithm presented by Dan *et al.* [88] generates a "use value" V_i, which is used to determine if the segments of content are to be cached, as well as when older segments are to be replaced; this is given by:

$$V_i = \frac{V_p}{V_s} \tag{3.1}$$

where V_p is the video popularity rating and V_s being the number of segments of the video already cached within the P2P group.

Using this algorithm, video which is currently in high demand is cached more often throughout the P2P group, providing an improved service for this content to requesting clients. Older videos which are less popular would still be cached in part at certain nodes, however, there is a possibility that clients requesting this content might receive poorer quality of service than that of random cache placements.

3.3.3 Non-P2P Client Video Delivery

In Figure 3.1(a), it can be seen that P2P based content is served directly to low powered or less featured devices that are unable to join the P2P pool. These devices, most likely to be mobile devices such as smartphones and tablets, are served the content from delegated P2P nodes acting as CDN servers. Although modern smartphones and tablets usually have the compute and storage resources required to operate in a P2P environment, power constraints restrict them from actively being used in this way; for this reason, standard content delivery is still preferred in these situations. Research into this issue is presented by Wuyao *et al.* [190], who express a number of methods for efficiently serving power restricted devices. One of these analysed methods proposes the serving of media directly from designated CDN servers, where the selection of the particular CDN server is controlled by a load balancing server; this method of CDN server selection distributes the load across all the CDN servers, minimising the probability that a server could become oversubscribed.

Using the described load balancing server selection process, the system can provide a very efficient streaming system. The system combines the highly scalable P2P system used to serve the majority of clients, with the distribution of content through CDN servers to any clients not eligible for P2P based delivery. This enables both P2P and non-P2P based clients to receive a

high quality of service while also delivering the content in an efficient and cost effective manner.

3.4 Comparison Results

The results were collected from a model implemented in the R language, with the goal of analysing the performance of the different architectures. The performance analysis focused on each architecture's ability to reduce congestion within the network, as well as assessing the scalability of each architecture by observing any demands that were unable to be met due to link congestion. The R language was utilised in this case due to its built-in support for network graphs and excellent statistical analysis capabilities; this enabled rapid development and simplified the processing of collected results.

Server capacity in this simple model was assumed to be infinite, as the aim was to investigate network utilisation and not server load balancing solutions. It is important to note that with the high adoption of cloud technologies, servers located in a cloud environment such as Amazon EC2 can be scaled up to meet extremely high load, with the only restrictive characteristic being financial cost; therefore, nodes possessing near infinite capacity were considered a reasonable assumption.

The model used randomly generated synthetic network topologies of varying size, which had restrictions to ensure they were fully connected graphs. The topologies were created with a node degree following a Weibull distribution with shape parameter 0.42, conforming to the results of a survey of ISP router level topologies [201]. Additional parameters were configured for certain scenarios, in order to analyse specific characteristic effects; this added configuration of network parameters allowed the collection of a more diverse and reliable result set. Some of these characteristics included: the number of CDN servers, the number of unique video content available, the number of clients requesting a specific unique video and also network link capacity. The exact values of these parameters will be stated in the relevant sections. Video content was modelled using a set of unique video file names, which were then requested by groups of randomly selected clients in the network, with each group requesting a different unique video.

The implemented model calculated the total load in the network, using arbitrary units so that the network capacity improvements of each technology could be analysed. Total network load can be described as follows:

$$\sum_{d \in D} d_{s,t} |E(d_{s,t})| \tag{3.2}$$

where D is the set of all demands, $d_{s,t}$ is the bitrate of the video demand d between source s and destination t, and $E(d_{s,t})$ is the set of edges within the path taken by d.

Although network load can be seen as a combination of bitrate and edges traversed, they are provided with arbitrary values as it has no direct correlation to real content values. These values along with the values for link capacities were chosen solely to achieve correct dimensioning of the network, but it should be noted that video content load values are proportionally dimensioned to correctly mimic different video qualities such as HD and 4K content. The reason for using arbitrary values is because, at this stage, it is only the load on the network as a whole that is being examined and not the accurate representation of video content. The focus for initial modelling is on traffic reduction in the network, so all the links in the network were modelled

as being relatively high capacity links in comparison to the demand sizes; this enabled a clear view on how traffic is placed in the network, without the consideration of blocking, as there were no obvious bottlenecks related to the network architecture. However, blocking is investigated later in this chapter, so it is important to maintain the possibility of blocking when demands are sufficiently increased. Furthermore, it is still possible for certain links to become congested given enough traffic, so dropped streams are expected in certain scenarios. One such scenario is the standard client-server architecture, which is used as a comparison technology to the CDN and P2P architectures.

It is important to highlight that the majority of the model results presented in this section are run on blocking networks, and while although the links are of high capacity, they do have a set limited capacity; this allows a clear view of the benefits of P2P and CDN architectures in relation to scalability, with details of demands that were not able to be met. However, results were also gathered from a non-blocking environment to illustrate the advantages of each architecture without capacity constraints, as there are additional traffic reduction improvements provided with P2P and CDN that need to be shown.

Results were collected over 40 runs of each scenario to ensure accurate results; this also reduced the probability of rogue results affecting average calculations. In some cases, the number of repeats was further increased to gain a clearer view of trends appearing in the results.

3.4.1 Results

In this section, details of specific testing scenarios for the R model will be described; results will then be presented, followed by a discussion of any observations. The CDN and P2P architectures are tested alongside a standard client-server architecture, whereby a single server provided content to all requesting clients. It should be highlighted that this is only for comparison, the performance of a single server system is not being analysed, since it is well known that this architecture type is not inherently scalable.

3.4.1.1 Network Size Variation

To test the scalability and performance of the different architectures, they were modelled using multiple network sizes; this provides a clear view on how the different architectures perform as both demands and network nodes increase in size. To make this a fair testing environment, model characteristics such as the number of CDN servers and client numbers were also scaled alongside the network size. Initial CDN server numbers started at 2, with the initial network size set at 20 nodes. Network size was then increased in steps of 20 up to 200 nodes, with the number of CDN servers increasing proportionally from 2 to 11 as the network was scaled up to 200 nodes, i.e. 1 extra CDN server for each 20 network nodes added. The link capacity was maintained at the arbitrary value of 1000 throughout all the model scenarios. The video demands were given arbitrary values from 5 to 25, in steps of 5, across 5 unique videos; with the number of clients requesting each of these unique videos scaling from 4 to 40, in steps of 4, in parallel to the steps in network size. Other client and video values were also tested in larger combinations, however, the results produced similar conclusions when using larger link capacities; a representative example of these results is shown in Figure 3.3. The model used for the congested network results shown in

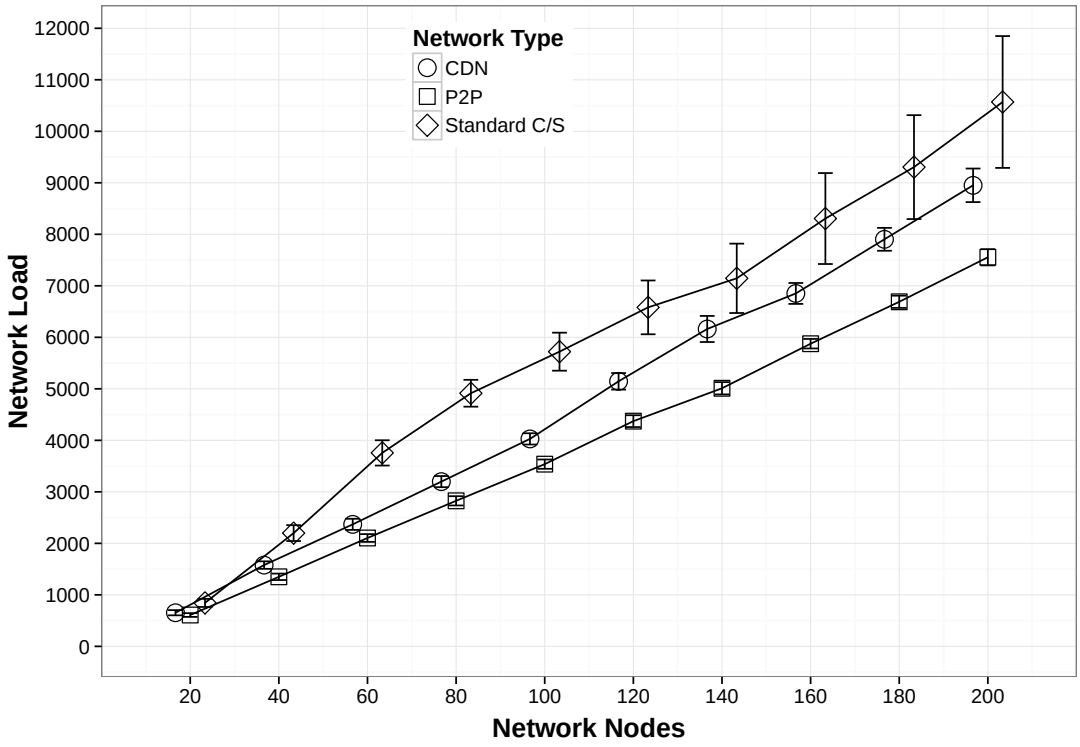
Figure 3.3 used a similar design as the standard network described, with only two characteristics being modified from the previous scenario. One of these characteristics was the client numbers scaling, which were set to use values from 5 to 50, in steps of 5; while the other change was an increase of the video demands to values scaling proportionately from 15 to 75, in steps of 15, across the 5 unique video demands. It should be clarified that results presented in this section use "position dodging" to better display results in close proximity to one another. "Position dodging" is a feature within plotting software that allows similar results from variable result sets to be displayed clearly on the same plot in close proximity. This may appear as if different x-axis values were used, but this is not the case as closely grouped results relate to the closest x-axis marker. To further clarify this, x-axis values for all results are integer values ranging from 20-200 in steps of 20. Additionally, all results use error bars to show 95% confidence intervals, however, in some cases the intervals are too small to see.

It can be seen in Figure 3.2 that both P2P and CDN architectures provide a significant improvement over standard client-server architectures, especially when the scale of the system is increased beyond 60 nodes in size. P2P technology appears to provide the most significant reduction in network traffic; it is noted that adding additional CDN servers would improve the results of the CDN architecture, but this would be costly in a real life scenario. This is an important point to highlight, as it provides an example where the cost of the system would directly influence its performance; such that the more servers that were assigned to the architecture, the greater the improvement in performance. However, adding servers does not guarantee an improvement in performance, as factors such as server placement will heavily influence the architectures performance [42]. Additionally, there is also a limit at which adding more servers offers no significant improvement to the system, as the network would eventually become saturated with content servers. This presents an important balance between performance and financial cost, which could only be optimised by knowing specific requirements for the different applications of the system.

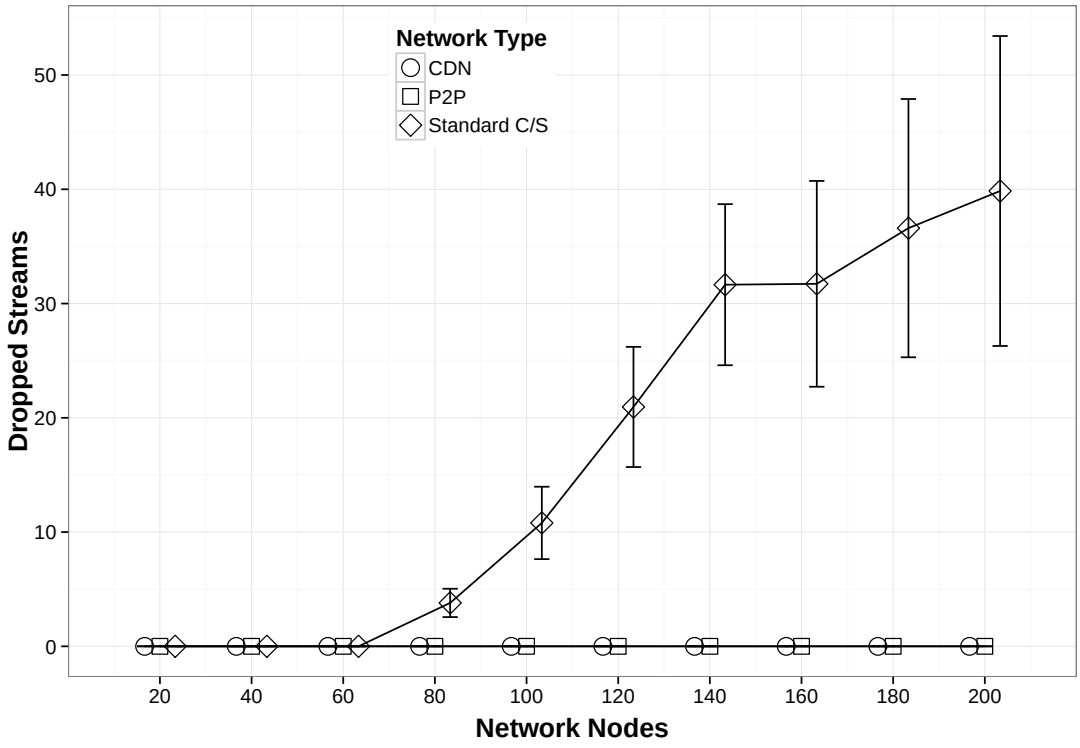
It should be reiterated that the results presented in Figure 3.2 are using a scaled network characteristic system, that scales factors such as demands and CDN servers as the size of the network increases. This ensures the network is provided with a significant volume of demands, to allow the network to be tested under a moderate load. The scaling of CDN servers is also important, as it is expected that to serve a network of a larger size, additional CDN servers would be required to efficiently serve the additional regions of the network; however, as previously discussed, the specific quantity of CDN servers would need to be determined based on specific applications and analysis of their usage. The specific implications of CDN server numbers are detailed in Section 3.4.1.2.

Figure 3.2(a) provides an insight into the reduction in network load that can be achieved by utilising P2P or CDN technology. This reduction in network traffic is likely to reduce the probability of blocking, leaving additional link capacity available for further demands to be fulfilled. P2P and CDN architectures also distribute the load from demands across all available resources, which is an improvement over the limited resources that are available for a standard client/server system; the added distributed resources available to P2P and CDN architectures allows efficient resource usage, providing the ability for additional demands to be met.

The scalability improvements can also be seen more clearly in Figure 3.2(b), which provides an



(a) Network Load



(b) Dropped Streams

Figure 3.2: Architecture performance comparison for uncongested networks of varying size.

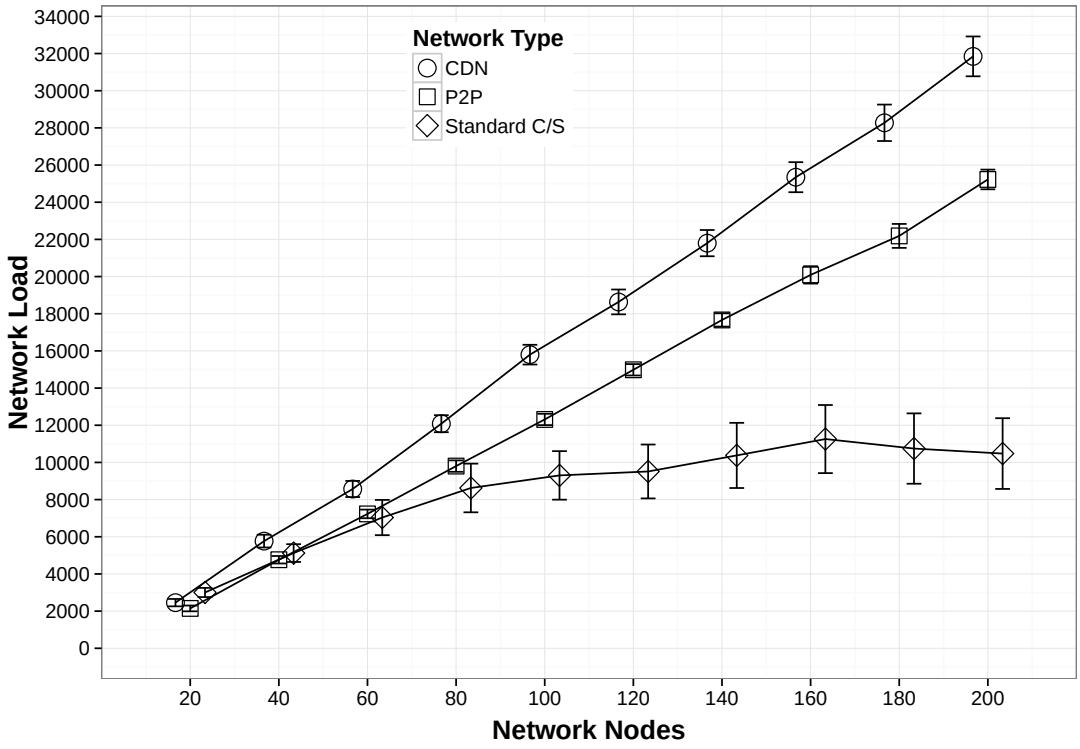
insight into the blocking characteristics of the network architectures. It is clear from the plot that both P2P and CDN scale very well, as they are able to serve all demands without issue. However, the standard client-server architecture suffers as the network scales in size, showing an increase in dropped streams when the network scales beyond 60 nodes in size. This presents a clear problem with standard client/server architectures, which is why most video content providers use other systems such as CDNs or ALM to distribute their content to clients.

The results shown in Figure 3.3 were gathered using a model with characteristics that ensured a heavily congested network; this caused both CDN and the highly scalable P2P architecture to drop traffic in some instances. The same conclusions can be gathered from these results as those previously found in Figure 3.2, with CDN producing higher network load than P2P. This increased network load contributed to the CDN architecture dropping slightly more streams than P2P. The standard client-server architecture produces significantly different results than shown before in Figure 3.2(a), this is due to the increased size of the demands causing congested links more rapidly than before. There is a visible plateau in the network load for the client-server architecture, that points to this value being a limit on the demands possible to be distributed from a single server, based on an average number of connected links; this can be confirmed by looking at Figure 3.3(b), which shows a rapidly increasing number of dropped streams when using client-server architecture.

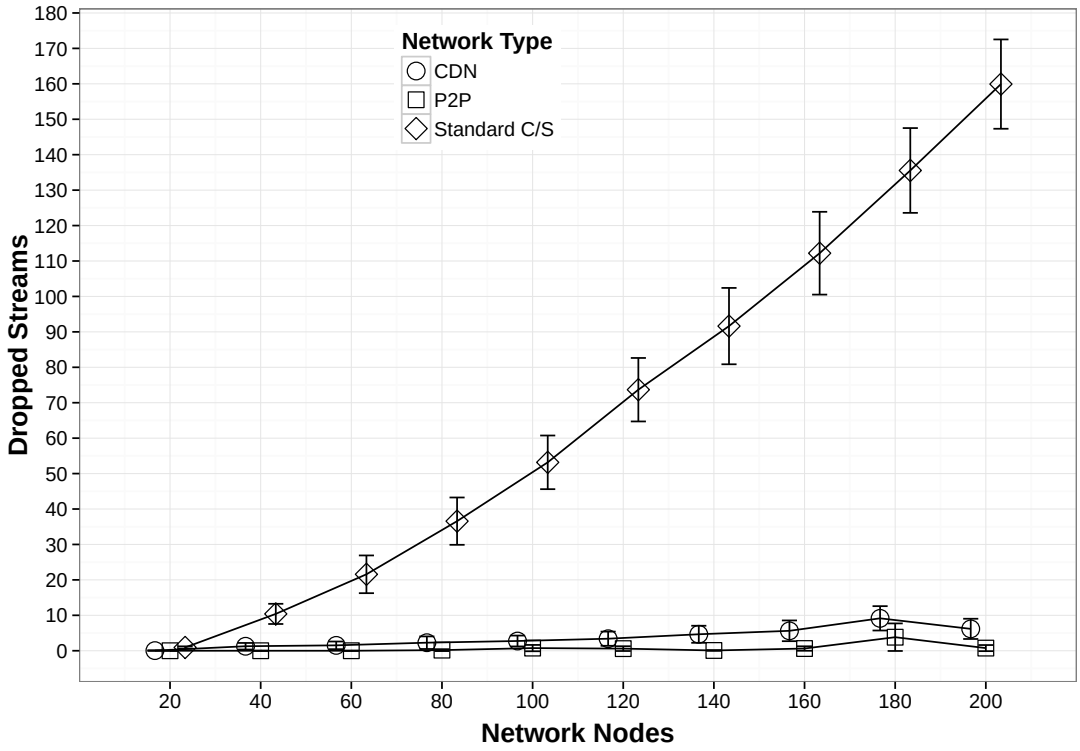
An interesting point that needs to be highlighted in the results shown in Figure 3.3(b), is that the highly scalable P2P is shown to be dropping streams in certain instances; this is interesting because of the highly distributed design of P2P, which should overcome normal congestion issues. However, there are a number of reasons why this distributed design might still allow a small number of streams to be dropped. One such reason would be if there was significant congestion in the network preventing the requesting client from establishing a path to any of the current nodes with the cached content. Another possible reason for this dropped traffic, could be due to the demands being added sequentially to the model, as within the model each individual demand is applied to the network one after another until all demands have been processed; this design was implemented to help model a real world scenario where clients would join a P2P network and request content over a period of time. Unfortunately, this design can mean that the first request for a specific unique video can fail to find an available path to the initial source of the content, if previous requests for other unique videos have caused congestion for required links. An alternative model design could utilise optimisation of the order of demand insertion; however, this would not relate correctly to a real-time video network where demands would not be known ahead of time. Similar to this explanation regarding an initial request for content with only the initial source, it can be expanded to also include the scenario of if a requesting client is located in a position where previous demands have utilised all of its connected links; this would mean any video requests regardless of cache placement would be dropped, as they would be unable to reach the client.

3.4.1.2 CDN Server Variation

The performance of CDN architectures relies heavily on the replication of the content across a number of servers, as this allows for load balancing client requests. Further to this, it should be



(a) Network Load



(b) Dropped Streams

Figure 3.3: Architecture performance comparison for congested networks of varying size.

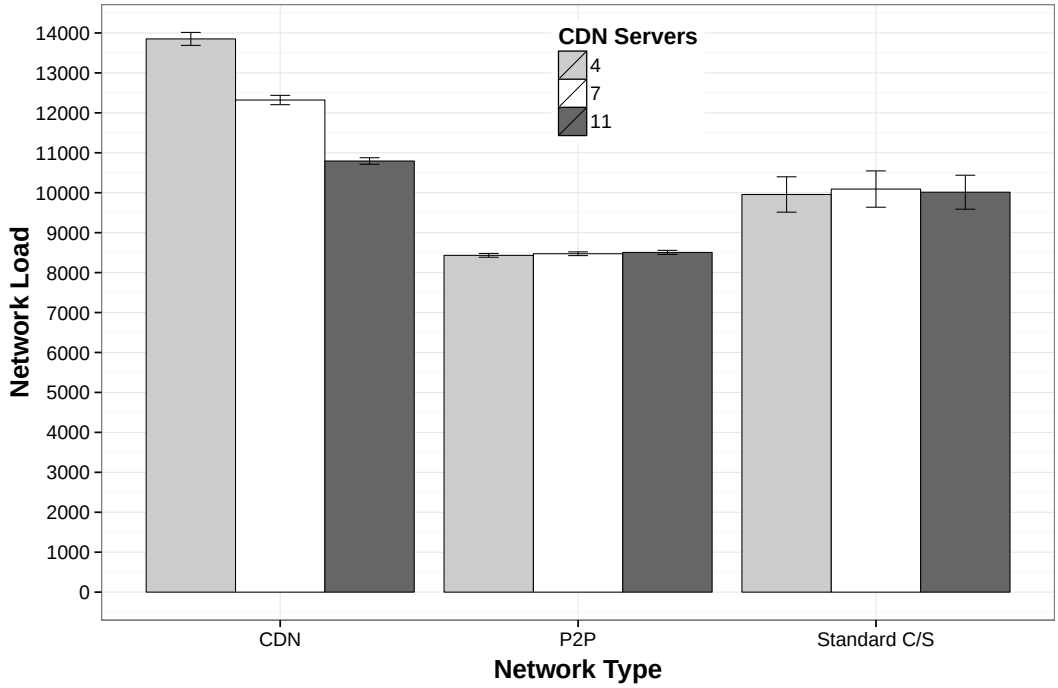
clear that the number of servers has an important role in provisioning for a reliable and scalable CDN system. By varying the number of CDN servers, we can gain an insight into how this number affects the performance of the system.

While varying the CDN servers, other model characteristics were maintained unlike in the previous scaled examples. Two scenarios were modelled, one with a 200 node network and the other with 400 nodes; other factors between these two scenarios were maintained to allow direct comparison. Expected results were also observed when modelling other network sizes, with network characteristics being scaled accordingly. Client numbers were fixed at 50 requests per unique video, with there being 5 unique videos with the same arbitrary values as the standard model detailed in Section 3.4.1.1. Link capacity was also kept at the arbitrary value of 1000 as before.

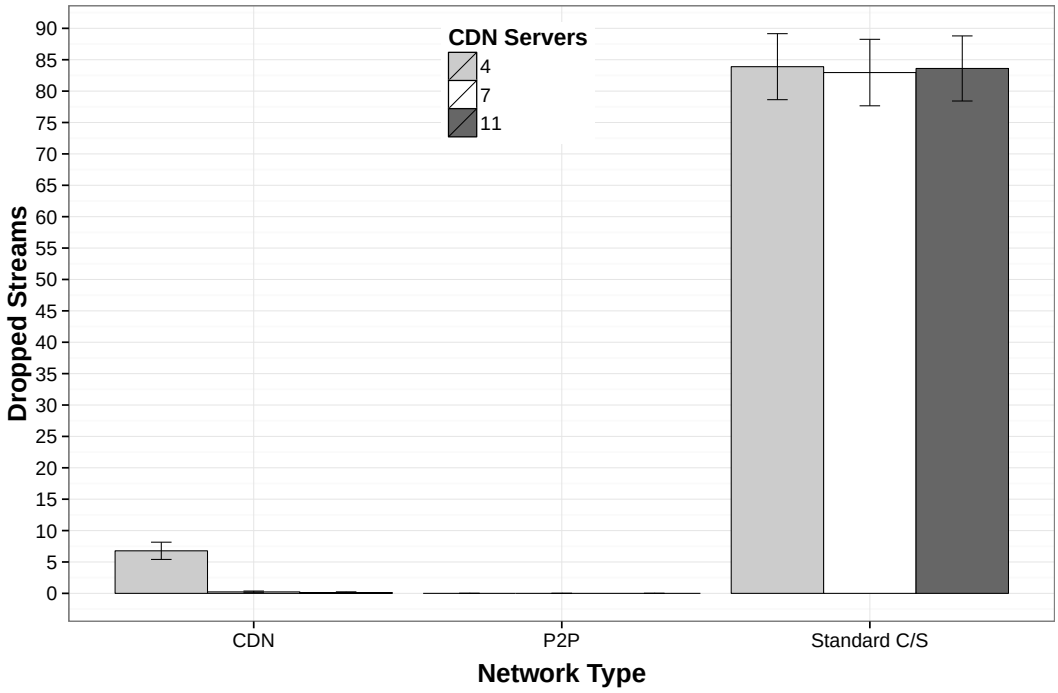
Figures 3.4 and 3.5 present results collected from running tests on a CDN architecture, including results from the respective P2P and standard client-server architectures for comparison. Although the client-server and P2P architectures can be seen to show slightly varied results for different CDN server number values, they are not affected by this value and are only shown in this way to display the equivalent experimental conditions found during that specific scenario. From Figures 3.4(a) and 3.5(a), it can be confirmed that the previously mentioned correlation stating that an increase in the number of CDN servers reduces network load remains true. Figures 3.4(b) and 3.5(b) also present evidence to further confirm this, showing that as the number of CDN servers is increased, the number of dropped streams is reduced. These results provide a clear observation of how CDN architecture has to maintain a balance between the cost of additional servers and serving its clients efficiently. It is important to highlight that if we were to increase the number of servers further, it would start to closely duplicate the performance of the P2P architecture; however, adding additional servers would require significant financial cost with minimal improvement in client viewing experience, as past the point of eliminating dropped streams, only a minimal improvement in latency would be visible to the clients.

It is evident from the results in Figures 3.4 and 3.5 that P2P provides improved scalability over the CDN architecture, reducing network load as well as serving all client requests. However, it is important to note that this is a very simple P2P network model and does not factor in possibilities of clients leaving the pool of peers. Furthermore, it is also important to highlight that this model does not provide results for real-time delivery of content, as it only models content delivery without a time restraint; this implies that results collected could only be reliably used to describe systems which delivered non time critical content. With the system not being a time based model, it was decided this would be an appropriate mechanism to demonstrate the full potential of P2P systems once the content had been fully dispersed through the network. It also should be noted that the model used a complete content caching mechanism at the P2P nodes, meaning each video a client accessed was fully downloaded and cached ready for distribution to other clients; this is not a representation of all P2P based delivery systems, as some only cache partial content for each video.

It is also important to highlight the misleading network load values presented in Figures 3.4(a) and 3.5(a) for the standard client-server architecture, which although are lower than that of CDN, are due to the significant amount of dropped streams and not a reduction in traffic; this can be seen



(a) Network Load

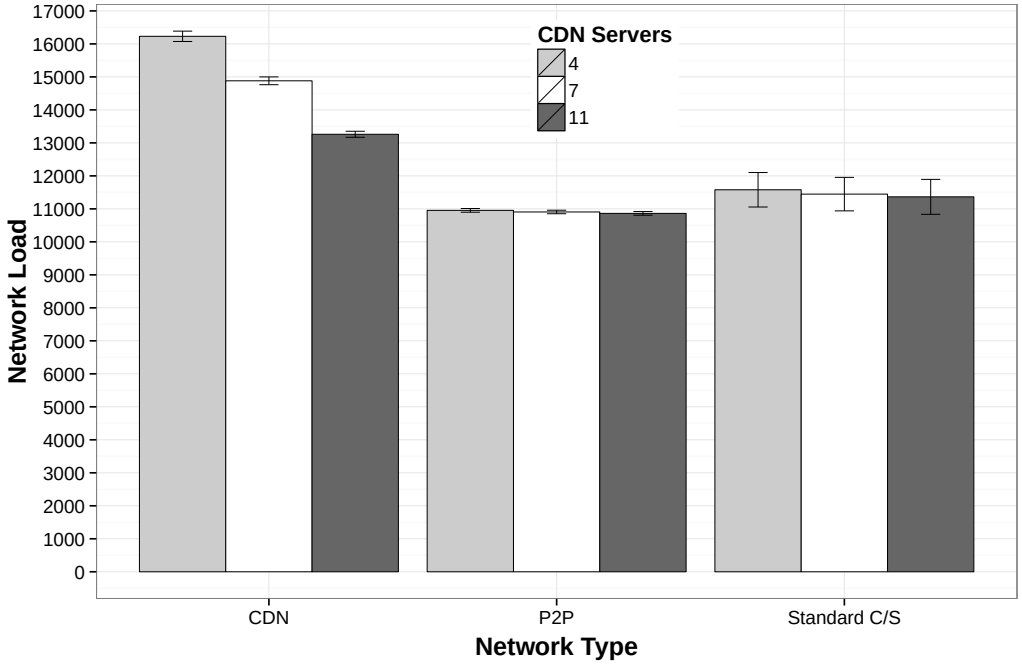


(b) Dropped Streams

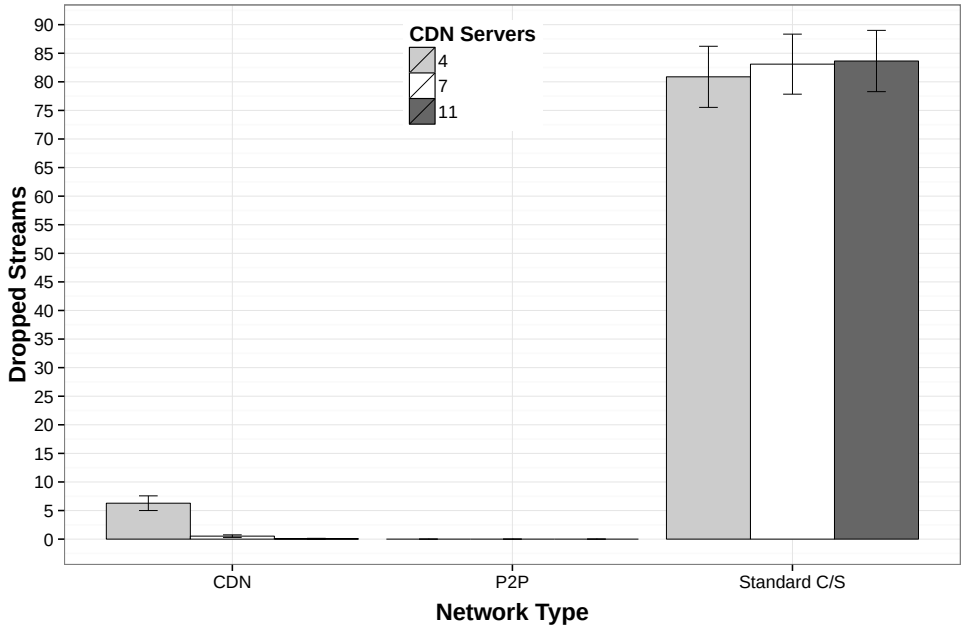
Figure 3.4: CDN performance while varying server numbers in a 200 Node network.

clearly in Figures 3.4(b) and 3.5(b), with dropped streams far above that of the other architecture types.

The comparison between the 200 node and 400 node scenarios displays the characteristics that one would expect, with increased load shown in Figure 3.5(a) for the 400 node network being caused by the additional links that traffic in the network has to traverse.



(a) Network Load



(b) Dropped Streams

Figure 3.5: CDN performance while varying server numbers in a 400 Node network.

Figure 3.6 provides an alternate comparison of the different architectures, with their performance being observed in a non-blocking network environment; this is in contrast to the other presented results, which focus on measuring performance based on the number of demands successfully met. The results collected for Figure 3.6 are from a 200 node network, with links set to have infinite capacity. Client numbers were fixed at 50 requests per unique video, with there being 10 unique videos with the same arbitrary values as the model detailed in Section 3.4.1.1. From the results shown we can see that both P2P and CDN architectures provide a significant improvement in reducing network load when compared with the standard client-server architecture. P2P, as expected, provides the best performance with regards to this, as it has the largest distribution of cached content throughout the network, further reducing the distance of a cached copy to any given client. CDN also shows expected results, with network load being reduced as the number of CDN servers are increased.

3.5 Summary

From the research and results presented in this chapter, it can be seen that both CDN and P2P architectures present several beneficial qualities for providing content delivery. The number of caches in either technology is the dominant characteristic that determines the performance of the system, due to the locality of the content to users. The implementation of caching within a set number of CDN servers is ideal for stored content applications; however, for real-time streaming scenarios the concept of using transcoders and application layer multicasting is a reasonable equivalent. Efficient network placement of resources is also important in CDN architectures, as it helps to maximise the benefits of the limited resources available [42]; this can also be seen

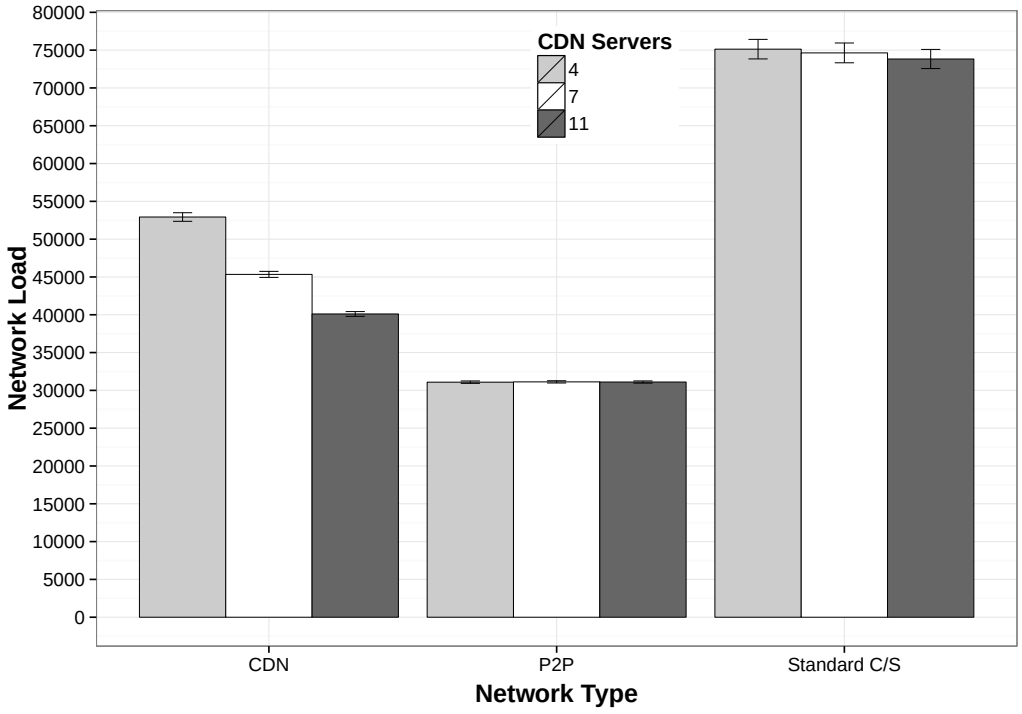


Figure 3.6: CDN performance while varying server numbers in a 400 Node non-blocking scenario.

to be important for placing transcoding resources in a real-time streaming scenario, as desirable characteristics such as locality to clients and limited resource utilisation are similar to those found in CDNs. This presents an optimisation problem that focuses on positioning a finite number of transcoding resources within a given network. The aim of this optimisation problem is to reduce both network traffic and blocking, with the scenario of streaming real-time content to a large number of clients, while using transcoders as application layer multicasting nodes; this optimisation problem is explored within the next chapter, where an algorithm is presented to provide a solution to the described problem.

Chapter 4

Optimising Transcoder Location

Advancements in multimedia technology have brought UHD content (*e.g.* 4K) services to a larger audience [202], which has had a significant impact on the traffic utilisation of some network paths, especially those based on transoceanic cables. Available capacity on these links will inherently become increasingly constrained, as people begin to stream real-time UHD content more frequently; because of this, it is desirable to develop systems that can minimise the additional traffic. Following on from Section 2.8.1, this chapter describes an approach that aims to reduce the traffic generated by real-time video delivery services. The specific approach that is presented, involves the use of a novel heuristic algorithm that provides a solution for efficient transcoder placement within the network; this algorithm is designed to reduce network load, thereby also reducing the probability of blocking. Although the presented heuristic solution is focused on providing real-time UHD content due to its high bitrate requirements, it is also relevant for other lower bitrate live streaming scenarios; detailed information on UHD content bitrates and the issues associated with its transmission can be found in Section 2.1. The presented heuristic algorithm could be utilised in a static placement system, however, its intended use case involves utilising it with the dynamic placement system presented in Chapter 5. The combination of the placement optimisation with the migration system, would provide a system that could dynamically migrate transcoding resources to optimised locations during normal system operation. It should be noted that the work described in this chapter, including some of the presented results, are taken from previously published work [203].

Existing systems currently in operation attempt to reduce the capacity requirements for the video content, as well as minimising any repeated traffic within the network. Two main mechanisms for achieving this, are transcoders [12] and application layer multicasting [163]. Transcoders can convert the media stream in various ways, such as by reducing bitrates, adjusting resolution and changing video format. These techniques can be used to reduce the link capacity consumed by the content stream, usually by sacrificing the visual quality of the content. However, this is not always the case, as lossless compression techniques can also be utilised [13]; additional information relevant to transcoders and compression techniques can be found in Section 2.1.

In the absence of good multicast support in IP networks, application layer multicasting can be used to reduce repeated content being sent through the network [163]. Application layer multicasting achieves this reduction by distributing streams to multiple clients, at positions closer to users than the original sender. This permits a single stream to be sent through the network to

the multicast enabling node, which then transmits the required multiple streams the remaining distance to the receiving clients. Branching the stream in this way can provide a significant reduction in the overall traffic load placed on the network, especially when streaming high bitrate UHD content; it should be clarified that ALM can work in parallel or individually with transcoders, to further reduce traffic requirements. Without application layer multicasting, it is doubtful that a large scale real-time UHD streaming system would be able to efficiently function, due to the bottleneck caused be the single source of the video content. Transcoding software such as FFmpeg [15] can act as an ALM node, as it has the capability to distribute multiple output streams to different destinations when receiving a single input stream; therefore, it can be assumed that any described transcoder node can also function as an application layer multicasting node. Further details on application layer multicasting can be found in Section 2.7.

Advancements in cloud technologies as described in Section 2.3, as well as the proliferation of cloud resources around the world, have provided the ability to easily create globally distributed systems. Cloud based services have also had a positive effect on the design of software services, including the introduction of using virtual machines (VMs) as transcoding resources. This concept has many benefits over previous systems, such as CDN based systems, which require the setup and use of dedicated machines and links placed throughout the network to perform transcoding services. A pool of dedicated transcoders may be an inefficient use of resources, as it is unlikely that all transcoding machines would be fully utilised at all times. Another issue is that certain machines could be oversubscribed under heavy loads, as they may not have been configured since they were first configured. VMs however, which can be easily and quickly created, destroyed and migrated within multiple cloud data centres [184], have the ability to scale according to the load they are currently experiencing. This feature makes them a cost effective solution to providing transcoded content to geographically diverse clients. The ability to scale with current demand has also facilitated scalability at a level that was previously not financially viable for most applications.

The proposition described in this chapter, is to use transcoders that are dynamically placed, according to need, using a cloud based computational resource. This basic proposition, while attractive, nonetheless has two inherent problems. However, both are addressed by the methods described in this thesis. The first problem relates to the issue of where the transcoders should be best placed for efficient network utilisation, which is solved using the heuristic algorithm presented in this chapter. The second problem that needs to be overcome, is how to effectively allow traffic to be seamlessly switched between transcoders when the transcoder resources are moved; the system presented in Chapter 5 aims to address this issue. The transcoder placement problem needs to take into account that the general number of clients can move location over time, overloading networks in some areas and underutilising them in others. One solution to this would be to over-provision the network so that it is saturated with transcoders, with these also providing application layer multicasting services. However, this is an inefficient use of resources, which would be costly to implement and maintain.

The balance between the number of transcoders being provisioned throughout the network and the performance of the system, is an important factor due to the costs involved. As previously stated, the ideal solution to providing the best service to clients, would be to saturate the network with transcoding resources; this, however, would incur a large financial cost and would most likely

be unsustainable as a business. One goal to overcome the issue of the performance/cost balance, would be to identify the point at which adding additional transcoders adds no further improvement in service. Finding the point at which this occurs would rely on statistical analysis of previous usage data, in order to identify the number of transcoders required to meet all demands. Using collected data in this way, would allow a relatively easy way to provision the correct amount of transcoders for a given content transmission; however, if additional transcoders were found to be required during streaming, the system presented in Chapter 5 would allow the deployment of these additional transcoders in the live system, without disruption to clients viewing experience.

The described placement problem is addressed in this chapter using a presented novel deterministic heuristic algorithm; this algorithm optimises the placement of a given quantity of transcoders, with the aim to reduce the total network load. The optimisation process focuses on maximising the benefits of application layer multicasting and transcoding, by aiming to move transcoders closer to large groups of clients. The presented algorithm is specifically designed for use in conjunction with the new migration system presented in Chapter 5. The combination of the two solutions will enable a UHD streaming system that can be optimised during a continuous transmission, with minimal disruption to viewers.

Using a modified version of the network model described in Chapter 3, the developed heuristic was tested against a generic solver that was developed. This generic solver was in the form of a genetic algorithm, which used evolutionary techniques to provide improved solutions over a number of generations. Additional information on resource placement optimisation and algorithm types can be found in Section 2.2, with further details on work relating to transcoder placement being presented in Section 2.8.1.

4.1 Placement Problem Statement

The problem of optimising positions for a set number of transcoders in a network graph will now be described. The placement of these transcoders can influence whether they provide a positive or negative effect on the total network load; so the focus of the presented heuristic is to ensure the transcoder positions have been optimised to minimise the total traffic in the network.

The placement of transcoders will be described over a graph $G(E, V)$, of edges E describing physical links in the network and nodes V representing attachment points for clients, servers and transcoders. The set of source nodes providing content will be denoted $S \subseteq V$; $T \subseteq V$ is the set of destinations for the content; and, $A \subset V$ is the set of compute resources in the network that are candidates for transcoder placement. In practice each node may host a number of clients, a smaller number of nodes will host sources and transcoders. This scenario assumes that candidate transcoder nodes, $a \in A$, are hosted in an elastic cloud computing technology such as Amazon EC2 giving the possibility to scale up resources as needed, with only cost being the restricting factor. However, each link, $e \in E$, in the network has a limited transmission capacity $u(e)$.

The traffic travelling through the network from s to t is defined as $R(s, t, b_q)$ where b_q is the bitrate for codec q, and where $b_q \in B\,\{codec\ bitrate : HD, 4K, ...\}$. These traffic requirements can be stated as a set of demands, $D = \{d_{s,t} \mid s \in S, t \in T, d_{s,t} = b_q, b_q \in B\}$. In practice it may not be possible to accommodate all the demands due to link capacity constraints, thus the successful demands are defined as $D' \subseteq D$. The problem then becomes to choose a set of paths for the

demands, denoted $P(D)$, to meet the objective

$$\max |D'| \tag{4.1}$$

subject to:

$$\sum_{p \in P(D)} x(e_p) \leq u(e) \quad \forall e \in E \tag{4.2}$$

$$x(p_d) \geq d_{s,t} \quad \forall d \in D \tag{4.3}$$

where: $x(e_p)$ is the traffic on edge e associated by a demand path $p \in P(D')$, the set of all paths for the solution; and, $x(p_d)$ is the traffic on a path fulfilling demand d.

The process described here is stage one of the problem to be solved, as it allows for optimisation of transcoder placements in the network; however, if this process is to be used with a dynamic system with optimisation throughout content transmission, it is required to move transcoders while streaming the content. Without dynamic migration of transcoders, the optimisation could only be made before the streaming takes place; this may be an improvement over random placement or statistical analysis based placement, but it is not enough to maintain efficiency when dealing with varying client demands during transmission. A solution to the problem of dynamic migration of transcoders is presented in Chapter 5.

4.2 Algorithmic Solution

The problem described in Section 4.1 is known to be NP-complete [204] and thus, there is no known polynomial-time optimal solution; this is due to the fact that testing every single combination of the variables requires a time exponential in one or more of the factors that can be varied. Consequently, this chapter presents a heuristic algorithm to provide a *good* solution in a reasonable time. The heuristic is compared to a genetic algorithm (GA) which can find good solutions although in a much longer period of time. The metric chosen for comparison is run time under the condition that the solution quality between the two algorithms is the same (or very close).

A GA was chosen for comparison as it provides a solution that has a high probability to be very close to the optimal solution, as long as suitable parameters are chosen [205]. Although a GA is relatively slow compared to a heuristic approach, it is considerably quicker at finding good solutions (sometimes optimal) than testing every conceivable solution and can generally get closer to the optimal solution than a heuristic algorithm [206]. Because of the long run time of a GA, it is generally reserved for offline route calculation. However, adequate solutions can be achieved in shorter time by reducing certain solver variables such as the number of generations or chromosomes.

Using a combination of the GA results and analysis of the problem, it was possible to construct a heuristic algorithm that could be used in place of a GA in an online system. This is the main purpose of this chapters work and will be considered after briefly describing the design of the GA. In both cases, the algorithms were constructed in R to make sure that the results were comparable in terms of run time.

4.3 Genetic Algorithm Design

The GA was implemented to perform both chromosome crossover as well as mutation functions. The combination of these functions provided a reasonable solution when applying them over multiple generations. The number of generations used during testing, was optimised to give both a good solution and running time that was as short as possible. The value for the optimal number of generations was determined by producing results for varying generation counts; these results were then analysed to determine the point at which adding further generations did not provide a significant improvement to the solution.

To produce the best solution without significantly sacrificing the speed of computation, the ideal values for the GA were found to be: 10 Generations, Population of 50, crossing and mutating 50% of the chromosomes at random each generation. Although 50% is a high level of mutation, it was found to be close to the optimal value based on analysis of solution quality and run time of a large number of tested scenarios. To simplify the GA functionality, it was decided to remove chromosomes with errors after crossing or mutation, rather than apply a repair function; it is important to note that this was implemented to reduce the delays in computation that would be caused if a large number of chromosomes needed to be altered. A chromosome was considered in error if it contained more than the required transcoders. The fitness function used within the GA, uses a combination of hop count obtained through Dijkstra's Shortest path algorithm and the content bitrate; the fitness function is based on the group scoring mechanism also found in the heuristic, where the hop count to the closest available source is used.

4.4 Heuristic Algorithm Design

The heuristic was developed through the use of both a mathematical approach as well as trial and error testing techniques; these methods provided a heuristic which performs almost as well and sometimes better than the GA, but produces a solution in considerably less time. This is necessary for the algorithm to be considered when looking for online traffic routing scenarios.

The heuristic looks at the available locations for transcoders and scores them using a fitness function, the best location is then picked as a starting location. The fitness function uses a combination of Dijkstra's shortest path algorithm, the load and codec used for the demands and also the number of connected links available to the transcoder location. The locations of the other transcoders are then chosen by selecting locations that are a certain separation from current transcoders, then running a group scoring function to determine the next best location. Reducing the number of suitable transcoder locations in this way provides relaxed selection conditions for the algorithm, which accelerates the selection process.

This approach is much faster than a brute force approach of scoring every combination, but it still provides an acceptable solution for the given problem. By keeping the locations of the transcoders separated as much as possible, it provides a high probability that there will be a transcoder at an acceptable distance from each client. Furthermore, separating the transcoder as much as possible also reduces the possibility of a transcoder being overwhelmed by clients while other transcoders remain unused; this is true for networks with a reasonably uniform distribution of clients.

The basic design of the heuristic is shown as Algorithm 4.1, which details most of the optimisation process. It should be noted that the DIJKSTRA function shown in the algorithm uses Dijkstra's shortest path algorithm to return the hop distance from the given nodes in graph G, while λ represents the separation constant that is used to optimise the algorithm for speed or accuracy; this separation constant is explained in more detail in Section 4.5. Additionally, the DEGREE function used in Algorithm 4.1 returns the number of active links available on the given attachment point. The group scoring function (SCORE) performs similar actions to the scoring mechanism shown from lines 4-14 in Algorithm 4.1, but instead of using all the possible transcoder locations, the closest transcoder to each client from the given group is used for the scoring function; this is performed across all the client requests, with their scores being added together to generate the final group score. The FINDCLOSESTTRANSCODER function used in SCORE uses Dijkstra's shortest path algorithm to find the closest transcoder to the client, then returns the location of this transcoder. The process of the heuristic is also presented as a flow diagram in Figure 4.1, to further provide clarity in its operation. It can be seen in Figure 4.1, how as previously described, the separation constant reduces the number of locations required to be scored during its calculation process. The reduction in possible transcoder locations improves the speed of the scoring process, while still maintaining a good solution quality. Detailed information on how the separation parameter effects solution quality, can be found in Sections 4.5 and 4.6.

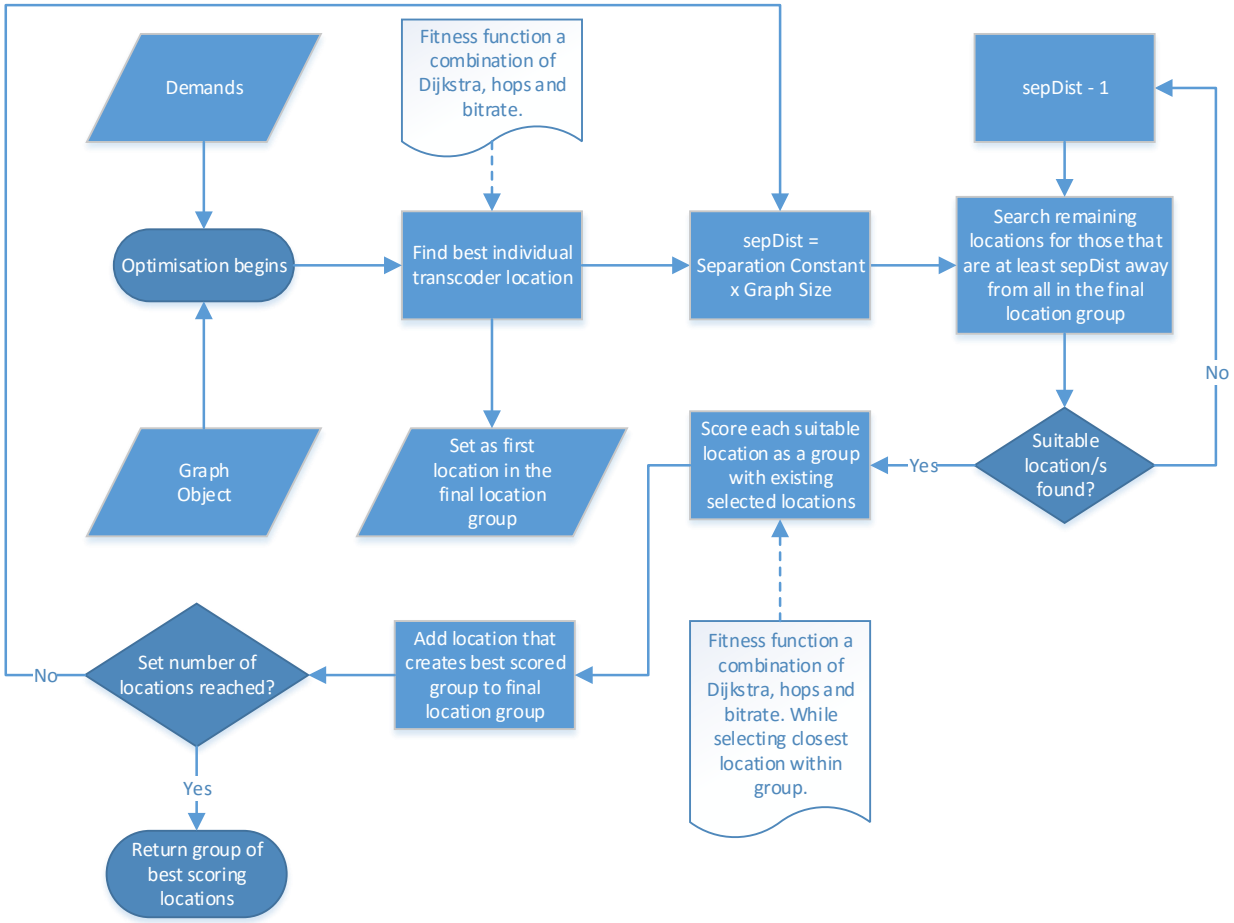


Figure 4.1: Heuristic flow diagram.

Algorithm 4.1 Transcoder placement for up to N transcoders, with separation constant λ

```
1: minScore ← ∞
2: transcoder ← null                                   ▷ a transcoder location
3: transcoders ← null                                  ▷ a list
4: for a ∈ A do
5:     score ← 0
6:     for t ∈ T do                                    ▷ for all destinations T
7:         dist ← DIJKSTRA(G, a, t)
8:         score += (dist × d_{s,t})
9:     end forscore ← score/DEGREE(a)
10:    if score < minScore then
11:        minScore ← score
12:        transcoder ← a
13:    end if
14: end for
15: transcoders ← APPEND(transcoders, transcoder)
16: if N > 1 then
17:    sepDist ← λ × |V|
18:    for 1 . . . (N − 1) do
19:        suitLoc ← null                              ▷ list of suitable locations
20:        while suitLoc == null do
21:            for a ∈ A do
22:                for k ∈ transcoders do
23:                    if sepDist < DIJKSTRA(G, a, k) then
24:                        suitLoc ← APPEND(suitLoc, a)
25:                    end if
26:                end for
27:            end for
28:            sepDist − 1
29:        end while
30:        grpScores ← ∞                               ▷ list, initially length 1
31:        bestLoc ← null
32:        for all j ∈ suitLoc do
33:            testGroup ← APPEND(transcoders, j)
34:            gScore ← SCORE(testGroup)
35:            if gScore < MIN(grpScores) then
36:                bestLoc ← j
37:            end if
38:            grpScores ← APPEND(grpScores, gScore)
39:        end for
40:        transcoders ← APPEND(transcoders, bestLoc)
41:    end for
42: end if
43: return transcoders

44: procedure SCORE(locations)
45:    groupScore ← 0
46:    for t ∈ T do
47:        trans = FINDCLOSESTTRANSCODER(t)
48:        dist ← DIJKSTRA(G, trans, t)
49:        groupScore += (dist × d_{s,t})
50:    end for
51:    return groupScore
52: end procedure
```

4.5 Methodology

Results were obtained using a modified version of the R model presented in Chapter 3, with the scenario of delivering real-time UHD content from a single server; the described scenario would utilise transcoders placed within the network to adapt the content to client requirements, such as downscaling a 4K stream to a HD format. As described previously in Chapter 3, the model used randomly generated synthetic network topologies of varying size, which had restrictions to ensure they were fully connected graphs. The topologies were created with a node degree following a Weibull distribution with shape parameter 0.42, conforming to the results of a survey of ISP router level topologies [201]. Additionally, the model was also configured to utilise application layer multicasting at the transcoders, except for certain scenarios where stated, in order to investigate its benefits when delivering real-time content. The ALM functionality was utilised in combination with the transcoders being used to reduce the bitrate of forwarded traffic, based on the video quality being requested. With this combination of application layer multicasting and transcoding, only one original quality stream was required to be transmitted to the transcoder for each unique video; additional content requests would then be fulfilled directly from the transcoder, without placing additional load on the streaming server.

There are three parameters that the model focuses on collecting: the run time of the heuristic compared to the GA; the comparison of the solution quality, which is the value of the objective function (Equation 4.1); and also the solution quality relating to the reduction in network load that is accomplished in certain scenarios, especially when concerning non-blocking environments. To allow a fair comparison between the GA and heuristic algorithm, the GA run time was measured as either: the time to achieve the same or better score in the fitness function than the heuristic; or, when the stop condition is reached. The latter is to allow for the fact that sometimes (rarely) the GA does not reach as good a solution as the heuristic. To further test the quality of the heuristic compared to the GA, the GA was run for a longer period after it matched the heuristic objective function result, in order to see if it could improve on this value.

The performance optimisation of the GA was briefly described in Section 4.3, as is standard procedure for determining GA operating parameters. The heuristic algorithm required more detailed specific parameterisation, which is described here. The algorithm has been designed such that it only requires one parameter to be adjusted: specifically the separation parameter introduced in Section 4.4. It was observed that this parameter affects both the run time and quality of the algorithm.

The separation value can be set to a range of values that theoretically fall between 0 and 1; however, selecting a value over 0.1 was found to provide in almost all cases the same solution quality. This presents a practical range of values between 0 and 0.1. However, through testing and analysis across numerous scenarios, the effective lower bound of the separation parameter was found to be 0.01; beyond 0.01, no improvement in solution would be achieved, just a significant increase in run time. This places the effective range for the separation parameter between 0.01 and 0.1. The exact use of the separation constant can be seen in Algorithm 4.1, which shows how it is used along with the number of network nodes in the network to produce a separation distance; this ensures performance is maintained as the network increases in size. Transcoders that are at least the separation distance away in hop count from current transcoder locations are

selected for scoring. From this, we can ascertain that a smaller separation distance (i.e. a smaller separation constant) will generally produce a larger selection of transcoders that meet the distance requirement, with a larger distance (i.e. larger separation constant) producing the opposite effect. Consequently, we can assume that although the larger pool of suitable transcoder locations will take longer to score than the smaller group, the larger group will have a greater probability of finding a better quality transcoder location than the smaller group. Using this information it is clear that we can configure the algorithm using the separation value for either a fast solution output, with a higher separation value, or a better quality solution, with a smaller separation value.

There are a number of characteristics within a given scenario that can affect the performance of resource optimisation algorithms. One of the most important of these is the ability for the algorithm to scale with the size of the network; therefore, we would expect that their performance should not drastically diminish as the number of nodes increases. Other network factors that could also influence the algorithm performance include the proliferation of data centres that a transcoder could be migrated to, as well as the number of transcoders that will be up and running in different geographical locations at any given time. In the presented work, the scalability of each algorithm is examined by increasing the network size, along with other factors which are described in Section 4.6.

It is also important to note that the number of clients and distinct videos will be a large determining factor to the algorithm performance. Having a low number of clients, or a high number of distinct videos, can reduce the effectiveness of application layer multicasting at the transcoders. The reduction in performance can be attributed to the observation that for application layer multicasting to provide benefits, a distinct video will need to be streamed to more than one client from a single transcoder. To investigate this further, the number of clients within the network was also altered for certain scenarios.

Results were collected over 20 repetitions of each scenario to ensure accurate results; however, in some cases, the number of repetitions was further increased to gain a clearer view of trends appearing in the results. The use of a high number of repeat sets reduced the effects of any spurious results, which may have been the result of abnormal random networks, or non-standard distributions of random resource and demand placements.

4.6 Optimisation Results

For scenarios where the separation value was altered, it is expected that different solution qualities and run times will be observed with the heuristic. Furthermore, due to the GA run time being quantified by reaching the same result (or better) than the heuristic, the GA run time will also depend upon the separation value of the heuristic. Therefore, the GA is denoted with a separation value on the relevant plots, so it can be correctly paired with the appropriate heuristic utilising a given separation value; however, it needs to be made clear that the GA does not explicitly use the separation parameter during its calculation. It should be further clarified that results presented in this section use "position dodging" to better display results in close proximity to one another. "Position dodging" is a feature within plotting software that allows similar results from variable result sets to be displayed clearly on the same plot in close proximity. This may appear as if

different x-axis values were used, but this is not the case, as closely grouped results relate to the closest x-axis marker. To further clarify this, x-axis values for all results are integer values ranging from 100-600 in steps of 100. Additionally, all presented results use error bars to show 95% confidence intervals, however, in some cases the intervals are too small to see.

Model parameters were scaled as the network increased in size, in order to maintain a moderately congested network that produced small amounts of dropped traffic during the blocking scenarios. Possible transcoder locations were set randomly around the network, with the number of locations equal to 10% of the network size; these locations represent the data centres available for the transcoder VMs to be migrated to. There were 5 unique videos available to stream, with arbitrary bitrate values of 30, 60, 90, 120 and 150, respectively; these videos are considered to be the same quality and bitrate, but of different lengths in duration. Each of these unique videos had clients associated with them, with some clients requesting different qualities of the content; following on from this concept, it needs to be highlighted how only a single full quality stream is sent to the transcoder for each unique video. Once the transcoder has received the content for a specific unique video, it then converts the content to the required formats/bitrates and distributes them to the relevant clients. The number of clients requesting each unique video was set at 20% of the network size, except where otherwise stated. For each group of clients requesting an individual unique video, the quality of the requested content was varied within the client group. 95% of the clients in each group requested a HD stream, with the required bitrate being reduced to 25% of the original 4K bitrate. The remaining 5% of clients were sent the full 4K stream at its original bitrate.

Apart from the results shown in Figure 4.4, all results were gathered in a non-blocking environment with network links set to have infinite capacity; in contrast to this, the described blocking based network scenario had its network links set with a capacity of 1000 arbitrary units. For the cases where it is non-blocking, in terms of comparing network quality, the objective function in Equation 4.1 is instead expressed as the total traffic admitted to the network, termed *network load*; the objective, in this case, is for the *network load* parameter to be as low as possible in order to allow further demands to be met. This reduction in traffic translates to being able to fulfil a larger number of demands, due to the increased available capacity within the network. Formally, we define network load L as:

$$L = \sum_{d \in D} d_{s,t} |E(d_{s,t})| \tag{4.4}$$

where $d_{s,t}$ is the bitrate of the video demand d, defined earlier, and $E(d_{s,t})$ is the set of edges within the path taken by d.

It is also important to note that apart from Figure 4.2, the GA was given the stopping criteria of reaching a result similar to that of the heuristic, so that the time analysis would be a fair comparison. But even without the stopping criteria, it is shown in Figure 4.2 that the GA does not provide a significantly improved solution even when given the large amount of time required to finish its calculation; this was reasoned as an acceptable result based on this fact.

It is important to mention the importance of using transcoders to both transcode content to other formats, as well as using them to perform application layer multicasting. Figure 4.3 shows the effects of utilising transcoders and application layer multicasting in a non-blocking environment. It

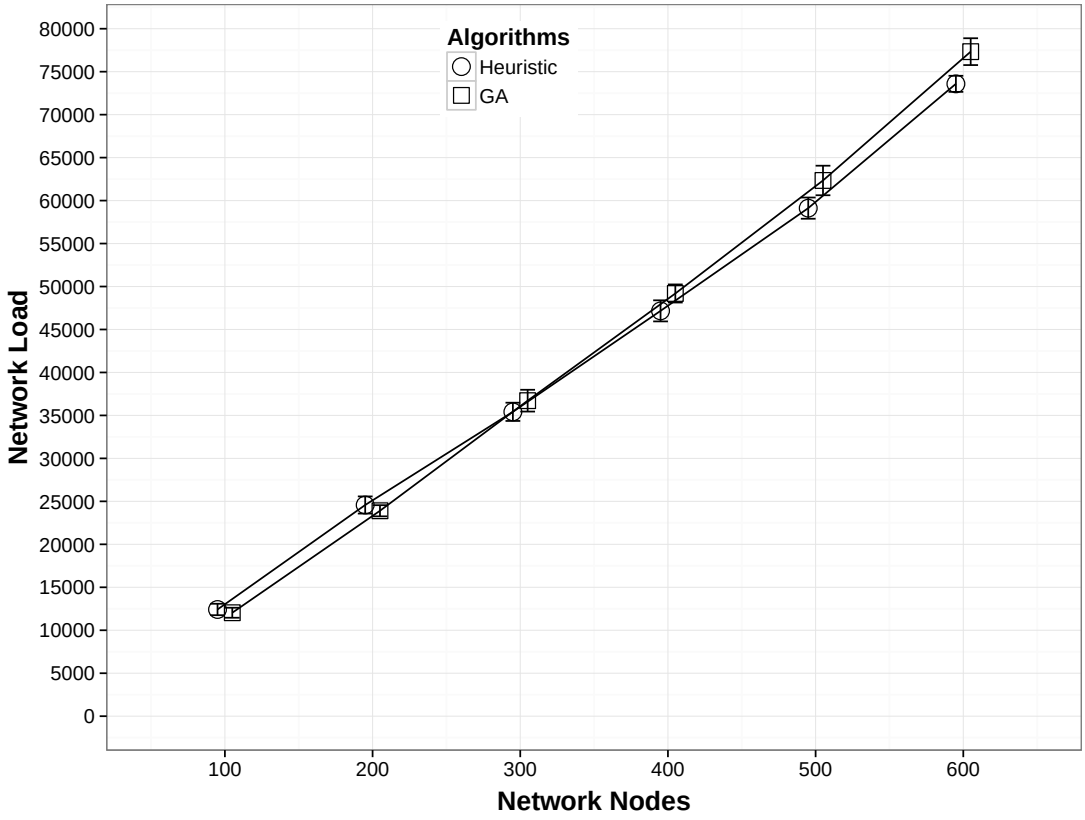


Figure 4.2: Network load for the heuristic and GA with no stopping criteria.

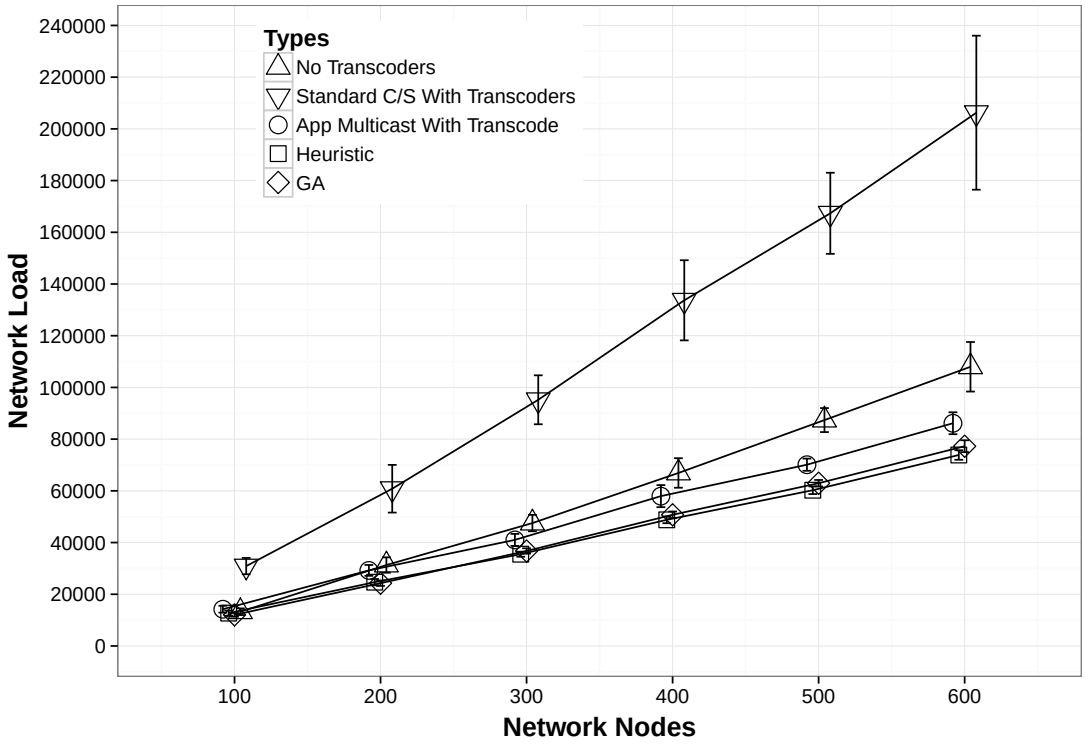
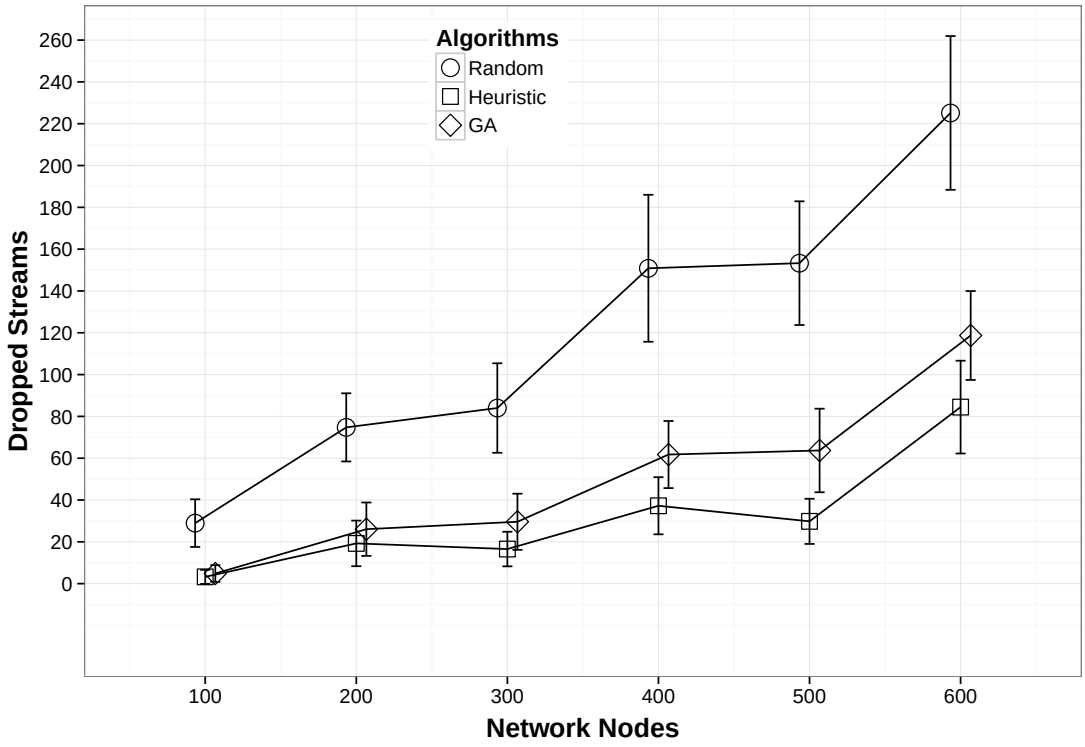


Figure 4.3: Network load for each scenario, within a non-blocking environment.

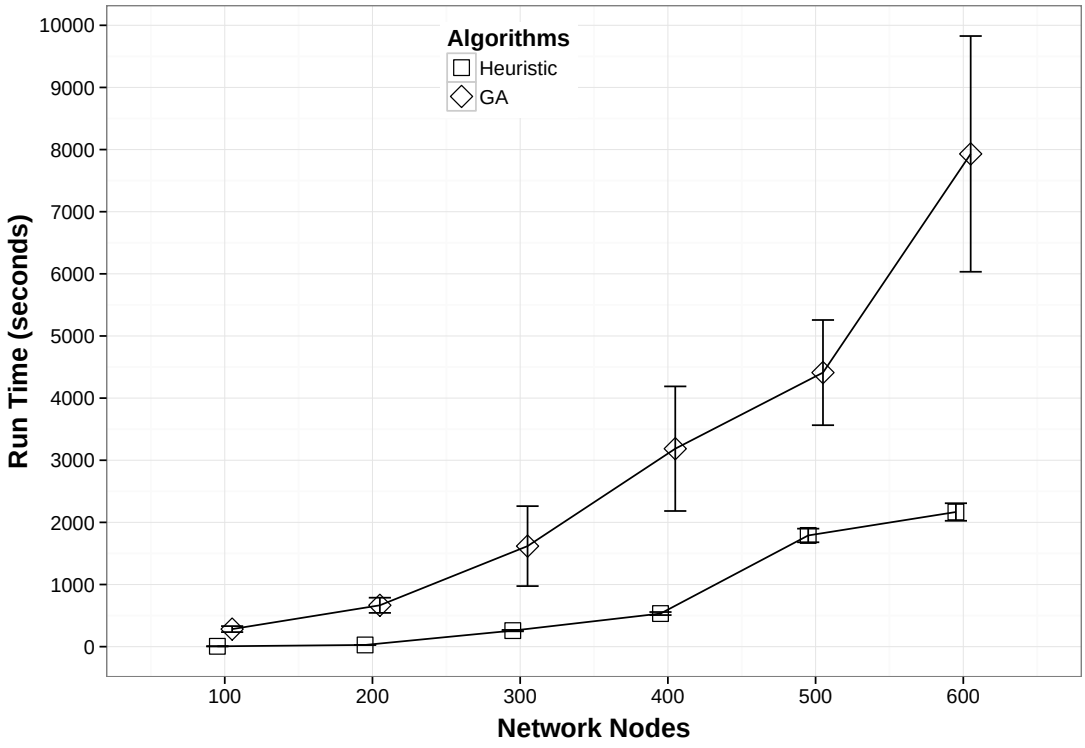
is clear from these results that using transcoders as application layer multicasting nodes provides an advantage in reducing overall traffic load in the network. If these results were collected in a constrained network which allowed blocking, it would be expected that an increased amount of blocking would occur for scenarios that did not use transcoders as application layer multicasting nodes. "Standard C/S with Transcoders" presents results for a scenario where transcoders are utilised without the aid of application layer multicasting, forcing each client request to be provided by the original server while still having to pass through a transcoder. It needs to be highlighted that the larger degree of variability that can be observed with the "Standard C/S with Transcoders" scenario, is due to the random placement of the transcoders. The transcoder placement heavily influences the length of the paths taken by demands, as transcoders placed far from both the server and clients can drastically increase the path length; this produces the increase in network load shown in Figure 4.3, which can be confirmed using Equation 4.4. A similar variability can also be observed with the "No transcoders" scenario, which is a standard client server architecture; for this scenario, the random placement of the server and clients again causes slight variation in the length of the paths travelled by the demands. "App Multicast With transcode" provides a similar scenario to that of "Standard C/S with Transcoders", with random placement of transcoders; however, it is configured to utilise application layer multicasting at the transcoders, which as seen in 4.3, provides a significant reduction in network load.

Figure 4.4 presents the results given from the objective function (Equation 4.1), while using the best case scenario of the heuristic with 0.01 as the separation value. The results in Figure 4.4 show that the heuristic improves on the GA solution, while remaining significantly faster at achieving the result. The random placement is included for comparison purposes only, with its run time in Figure 4.4(b) being omitted due to the negligible time factor. It should be noted that for networks above 700 nodes, the performance of the heuristic does begin to diminish in quality, although not to the point that brings it below that of random placement. The number of nodes reflects switching and attachment points for devices, which includes servers, transcoders and clients. Thus, a network size of 600 is already large and may support a very large number of client devices. The variation in results that can be observed in Figure 4.4(a), is a result of the highly unpredictable nature of the large collection of factors within the network scenarios, which are not affected directly by the transcoder placement algorithms; these include network topology, client placement and server placement, which all heavily influence the number of demands that are able to be met regardless of transcoder placement. It is possible that the variation in these results could be minimised by including additional repeat results.

Figure 4.5 shows how the separation parameter can affect the outcome of the solution quality, as well as the run time of the heuristic algorithm. Note that the results in Figure 4.5 for the comparable GA are denoted with the separation value for identification purposes only, the GA does not use this value in its calculations. The smaller separation value is observed to provide a better solution, however, it takes longer to achieve this. In contrast to this, the larger separation value provides a slightly worse solution in a shorter period of time. However, there is only a marginal difference in solution quality, and certain scenarios would find this more than adequate for the improvement in run time. It should be noted that separation values other than 0.01 and 0.1 were used during testing, but the two values presented in Figure 4.5 were chosen to give a

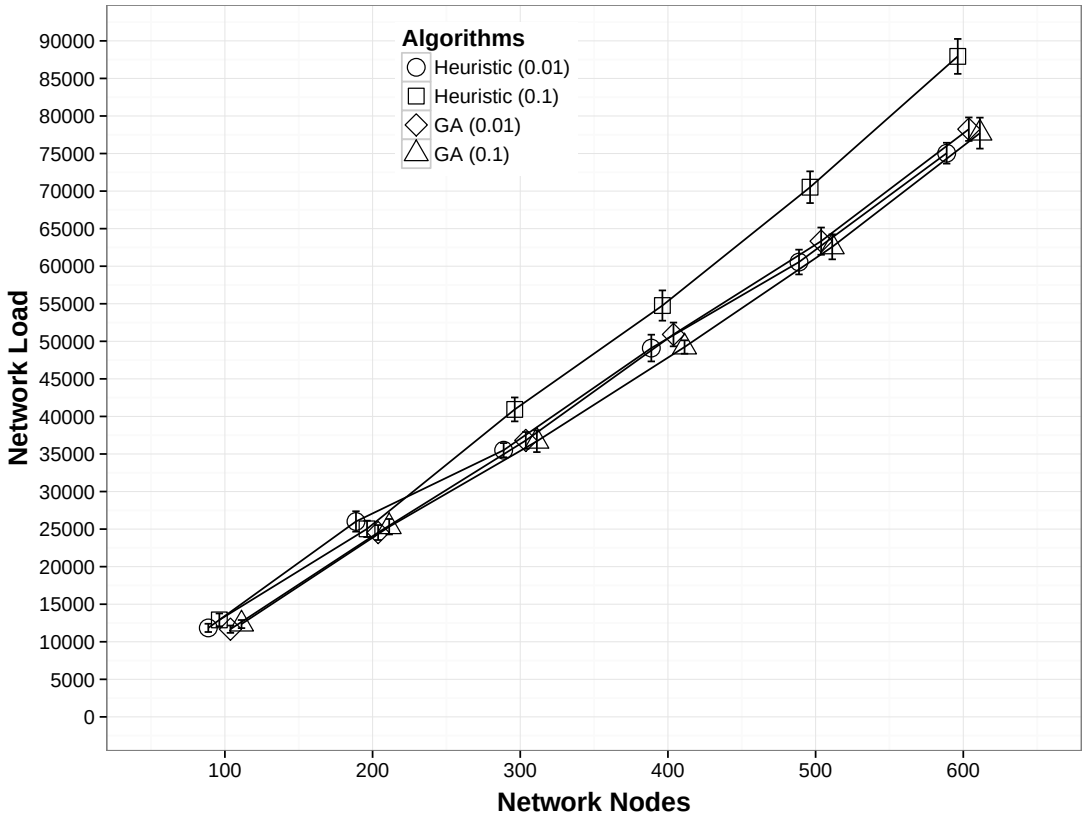


(a) Blocking Reduction

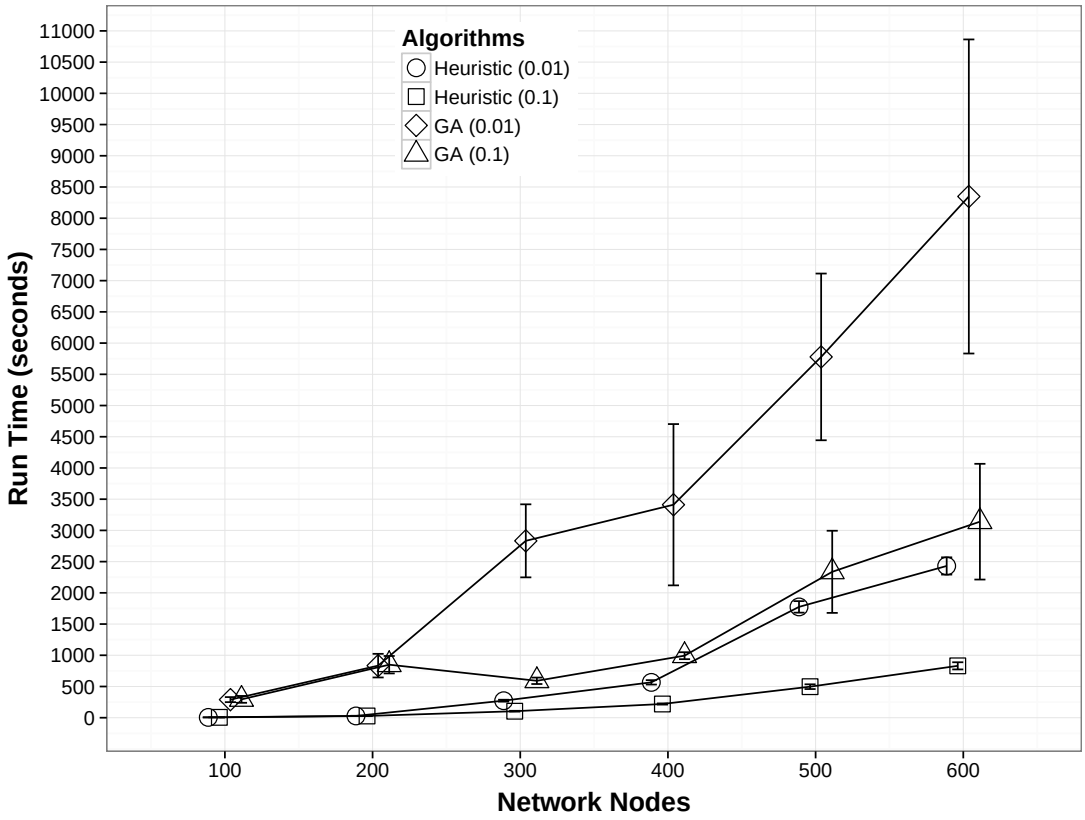


(b) Run time performance

Figure 4.4: Algorithm solution performance across varied network sizes.



(a) Network load performance



(b) Run time performance

Figure 4.5: Algorithm solution performance for varied separation values.

clear example of the how the balance between solution quality and run time can be influenced. Separation values between those presented in Figure 4.5 produced results that would be expected, lying between the results of 0.01 and 0.1. As discussed in Section 4.5, using values above 0.1 does not introduce an improvement in solution quality or run time, so values above this were omitted.

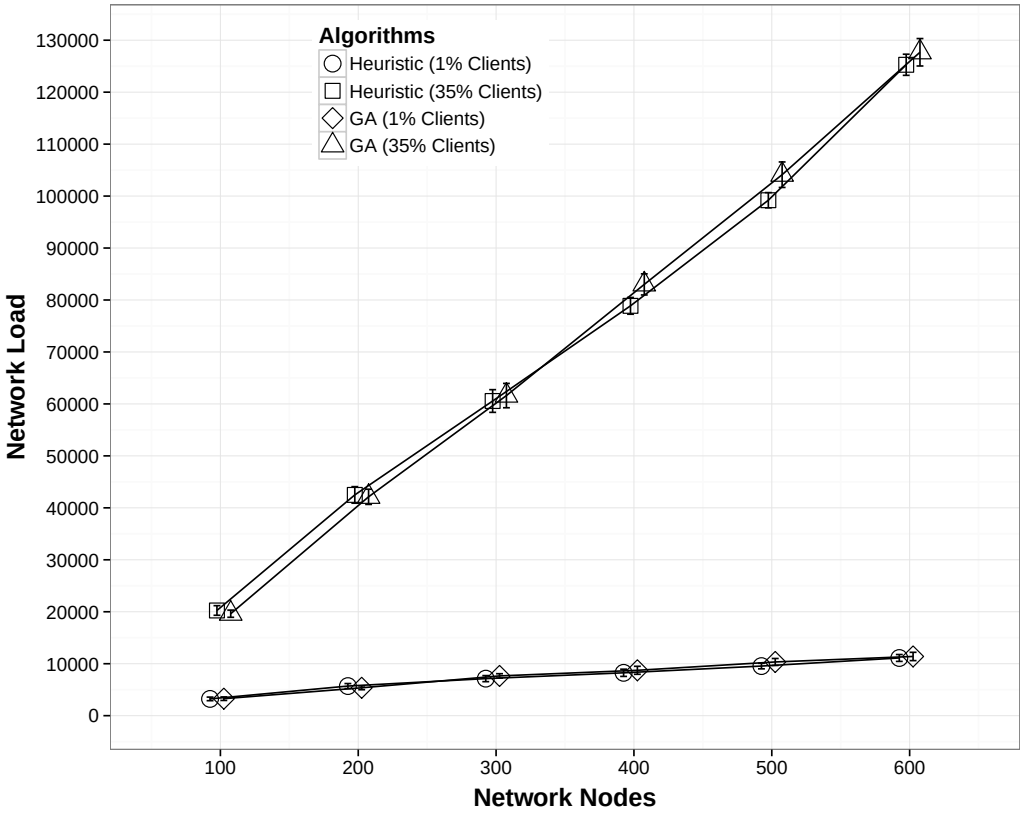
From Figure 4.6 we can determine that the solution quality of the heuristic is maintained throughout the variation in client numbers; along with the run time being consistently lower than that of the GA when client numbers begin to rise. The 1% and 35% results shown, indicate the percentage of total network nodes being utilised as clients. Furthermore, it should be noted that client values between 1% and 35% were also tested, but they were omitted from Figure 4.6 for clarity; however, it is important to highlight that the results collected between these values show an expected gradual increase as client numbers rise, in line with the presented results.

It should be noted that the variability of the run time with regard to the GA shown in Figures 4.4(b), 4.5(b) and 4.6(b), is because of the unpredictable behaviour of the GA; this is due to it being based on the random process of selecting what it believes to be best candidates for creating solutions, until it reaches a solution considered similar enough to the heuristic. This type of algorithm is expected to behave in this way, so the variation is considered to be within reasonable levels.

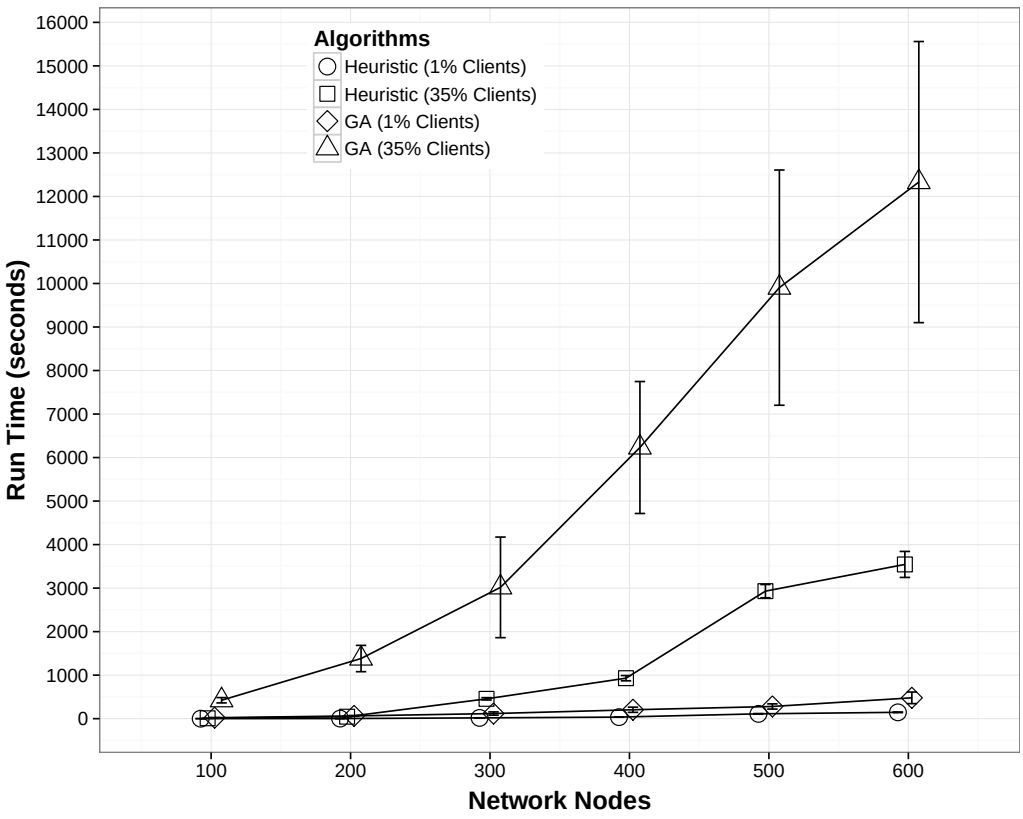
From the results shown in Figures 4.2, 4.5 and 4.6, it can be seen that the heuristic algorithm performed as well, or better, than the GA. Additionally, with respect to network load, the heuristic also on occasion produced a better solution than the GA. Even when utilising the less efficient separation value of 0.1, the heuristic still obtained a result that was within 11% of the GA network load value. More importantly than this, the heuristic is significantly faster at providing a solution than the GA, which means it could be utilised in an online system for optimising live traffic in a network. This is an important factor, based on the intended scenario of using this system to dynamically migrate transcoders based on active client demands.

4.7 Summary

It is clear that the heuristic produces solutions rivalling and in some cases improving on those from the GA; furthermore, it is shown that the heuristic provides its solution in a considerably shorter period of time. Given that the heuristic algorithm can efficiently place transcoding resources in a network based on current user demand, it becomes desirable to utilise this with a system capable of moving transcoders throughout the network. This is already possible with the use of virtual machines; however, there is not currently a system available that allows seamless migration of these transcoders while they are currently processing real-time streams. With a system such as the one described, the presented heuristic algorithm could be used to maintain efficient resource placement during continuous system operation, without interruption to the clients viewing experience; a system to achieve this is presented in the next chapter.



(a) Network load performance.



(b) Run time performance.

Figure 4.6: Algorithm solution performance for varied client numbers.

Chapter 5

Transcoder Migration Using OpenFlow

This chapter will detail a proposed and tested system, whereby a transcoder can be migrated from one geographical location to another; the aim of the proposed system is to improve on existing migration systems, by introducing minimal to no interruption in the video stream it is currently processing. Achieving this presents the opportunity to move transcoding resources closer to a dense area of receiving clients, with the objective being to reduce traffic load within the network as a whole, as well as optimising network usage on constrained links.

The presented system provides a solution for optimising the distribution of real-time live video content, which is unable to be cached in CDN based systems due to its requirement for instant viewing in parallel to recording. A feasible use case for the presented system, would be a sporting event being held in the USA, where some clients were requesting the feed from the UK. A transcoder would already be in use within the USA to adapt the content to specific client needs, while also performing application layer multicasting closer to dense client areas to reduce traffic; however, the currently placed transcoding resources would most likely become inefficient if the client user base migrated, or if additional clients requested the stream in areas such as the UK. This inefficiency is due to the static characteristic of the transcoder resources, which would have been set up and positioned optimally before the initial content stream was initialised based on predicted client demand. It is already possible for new transcoders to be set up during transmission for any new clients that connect; however, migrating existing clients to these would disrupt their viewing experience. The mechanisms for using previous statistics and other predictive reasoning to place transcoders throughout the network, can lead to over-provisioning the network with transcoding resources, which would be costly; alternatively, under-provisioning is also possible, this would cause bottlenecks in the network which can lead to reduced performance, as well as diminished quality for clients. For instance, if there was no indication that clients from the UK would be viewing the content stream, a significant amount of link capacity would be consumed on constrained transatlantic links if the client numbers in the UK were to increase. The additional traffic on the transatlantic link could cause issues in providing reliable streams, as well as congesting the link for other traffic. The system presented in this chapter aims to solve these issues, as well as adding further benefits beyond reducing the traffic on constrained links.

The presented system will be built on top of the functionality of an OpenFlow controlled

platform, which allows for the migration of traffic to occur in a seamless operation. This is achieved using OpenFlow's ability to redirect client media streams to the newly positioned transcoder, using direct manipulation of flow rules in switches. The system was tested and demonstrated using video content up to 4K resolution, which was required to be transcoded into smaller resolutions, such as HD in certain scenarios.

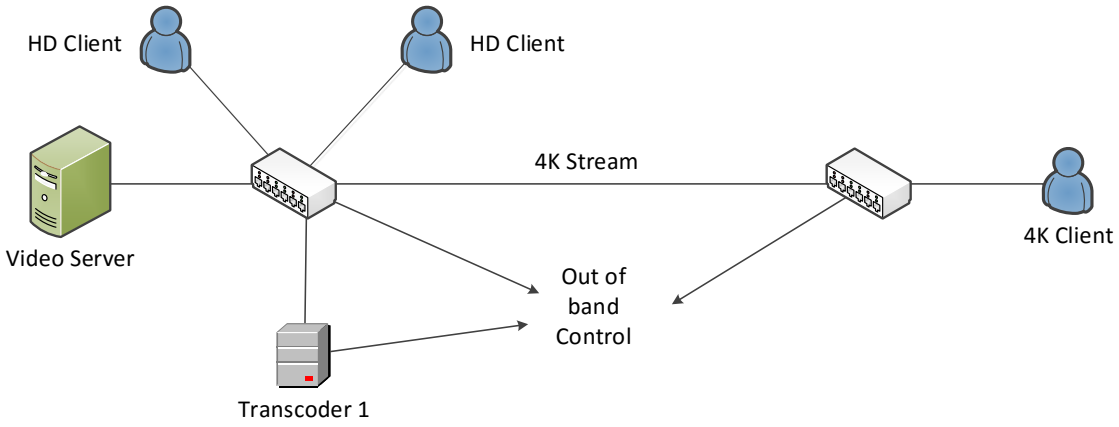
It is assumed that the proposed system would be deployed on a L2 based single operator network, with OpenFlow enabled switches being used throughout the infrastructure. It is possible for the system to function over a standard L3 network, as long as OpenFlow enabled switches are used to bridge the connection; however, this would require additional mechanisms to be configured using OpenFlow's packet header manipulation, in order to effectively tunnel the L2 traffic over the standard L3 network.

Figure 5.1 shows the basic concept of the system presented, as well as the overall architecture design. Figure 5.1 presents a clear example of how moving a transcoder can conserve significant capacity along a connecting link. It can be observed that when the same video content is requested by multiple clients in the same or different resolutions, a reduction in repeated content can be achieved, by transmitting a single 4K stream to the second switch and transcoder. The link joining the two switches in Figure 5.1 is intended to represent a constrained link, such as a transatlantic cable between the USA and the UK; in the presented scenario, the migration decision is determined by the sizes of the client population on either side of the transatlantic link. This scenario is also represented in Figure 5.2. It should be noted that this is a very simplified example for use with the basic test setup in Figure 5.1. A more advanced approach to solving the migration decision problem was described using the heuristic algorithm presented in Chapter 4.

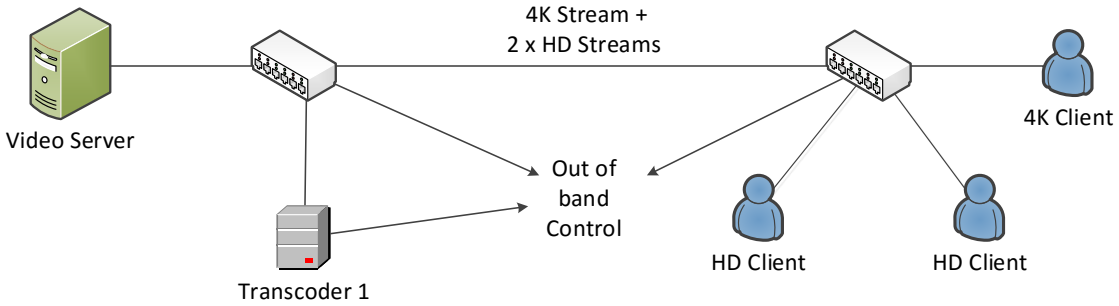
It may be considered that a simpler solution to the example shown in Figure 5.1 would be to keep both transcoders running, instead of migrating the initial transcoder; however, this would not mitigate the issue of migrating the streams of the clients currently streaming from the initial transcoder. The seamless migration of the streams from one transcoder to another is what the presented system aims to achieve, whether the initial transcoder is migrated or an additional transcoder is instantiated. In a more complex example, it may be required to leave the existing transcoder serving a group of clients that are in close proximity, while also migrating other streams from a group of geographically distant clients to another transcoder closer to the group.

5.1 Scalable Video Coding

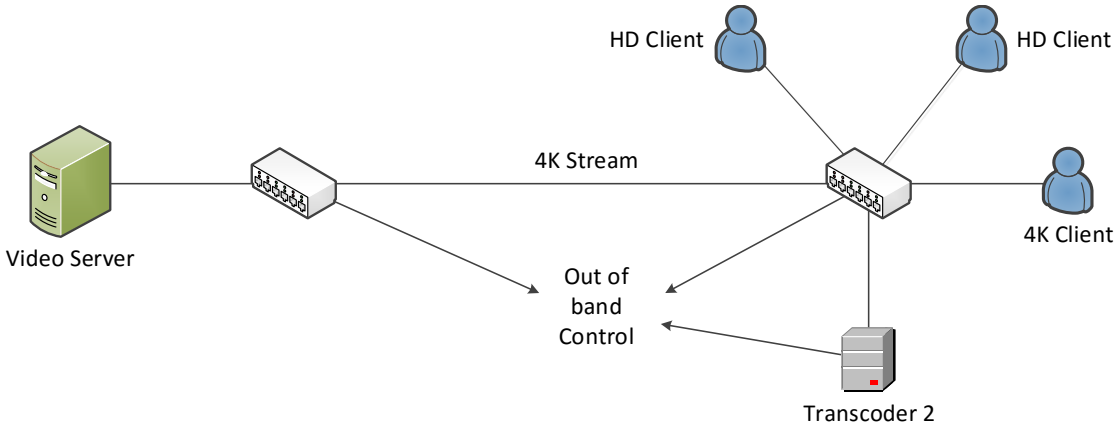
It should be highlighted that the system presented in this chapter could also be applied to not only transcoder migration applications, but also for migrating standard application layer multicasting nodes. Migrating these nodes becomes desirable when using a scalable video coding system, which would have already encoded the content with multiple resolutions at its source. This scalable video scenario would not require transcoding close to the clients, but would still require application layer multicasting to be applied, in order to relay the stream closer to the clients. This scenario would work in a similar effect to the transcoding scenario, as it would work to reduce both network and server load, while also reducing traffic on constrained links. It should be noted that FFmpeg, which is used for the proposed system, can perform this application layer multicasting task: it can listen for input on a given network port and then use its video copy attribute to relay the



(a) Initial State



(b) Migration Desirable State



(c) Post Migration State

Figure 5.1: Migration concept for traffic reduction.

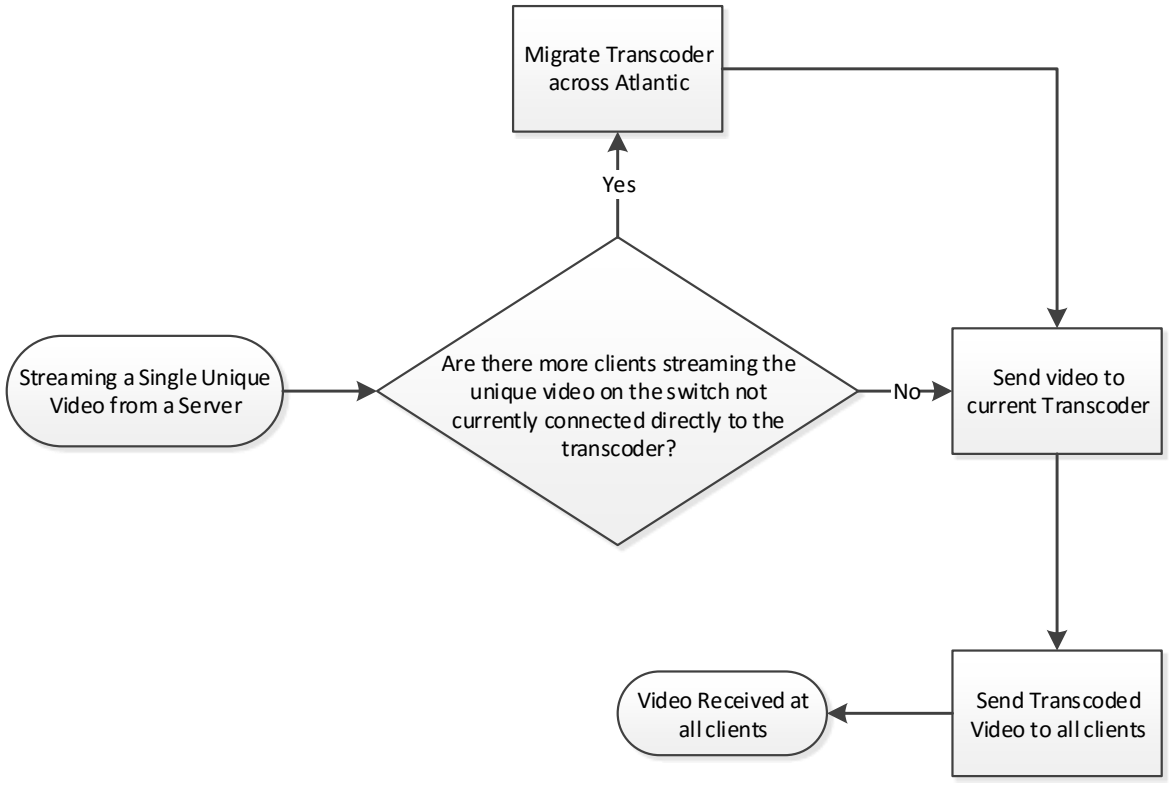


Figure 5.2: Basic migration process.

content to multiple output streams; these multiple output streams would be configured to deliver the content to the multiple requesting clients. Although this scenario is not detailed further, it is important to mention that the migration mechanism presented would function in the same way as described, even when only using application layer multicasting for a scalable video stream.

5.2 System Applications

The system presented in this chapter would support a number of scenarios not previously possible. Most of these scenarios relate to the recent popularity of home streaming applications, which were not previously available.

One such application for the presented system, is the delivery of a real-time UHD video stream of a live sporting event or other performance; these types of live events require the live stream to be sent to a large number of geographically diverse clients. These clients would most likely consist of large venues such as cinemas, which already provide live performance showing at their venues to large audiences [207–209]. The proposed system would involve an initial server that would be close or directly connected to the recording source. This would then use transcoders placed throughout the network at cloud data centres to serve a large number of clients. These transcoders would be initially placed using the heuristic algorithm presented in Chapter 4. The initial number of transcoders would be based on expected load, however, this is not crucial, as using the proposed migration system the number of transcoders could easily be adapted during streaming. Therefore, if a large group of clients starts to appear in a location that is a significant

distance from the nearest transcoder, a migration could be initiated to either move a transcoder to a more efficient position or create an additional transcoder if required.

The described system configuration can be mirrored with other scenarios, such as the home user streaming examples described in Section 2.8. Applications such as Twitch [168] have recently become more dominant among gamers, who want to stream their gaming session live [169]. Previously, gamers had to rely on recording their session and then uploading it to a video content provider such as YouTube [170] for later viewing. Unfortunately for gamers, this process separated them from their audience and lacked the interaction that a live broadcast provides. This requirement for improved user interaction is why live streaming is now seeing a large rise in popularity, as gamers can chat with the gamers during their broadcast and influence their game play based on popular demand.

Live broadcasting applications have also arrived on mobile platforms such as smartphones, which allows users to stream content straight from their camera to a large dispersed audience. Some examples of these types of applications include Periscope [171] and Meerkat [172], which both have similar functionality. These applications have the same issues as game streaming applications, in that they are using a limited network connection combined with limited processing power; this is especially true with regard to the mobile platforms. Due to these restrictions, current streaming services are implemented using technologies such as HLS [30] and MPEG-DASH, which utilise caching and the Hypertext Transfer Protocol (HTTP) protocol to enable content distribution to a large user base. Unfortunately, technologies such as these, provide a significant delay between recording and viewing the stream; with delays in the order of 10-20 seconds [176–179]. This long delay is acceptable in certain situations, however, for true real-time streaming, other options such as the system presented in this chapter would need to be utilised. The proposed true real-time streaming system would be beneficial to live sporting events, especially with the wide availability of social media, where details might be revealed before they were viewed. Other scenarios requiring as close to real-time as possible would include any scenario where user interaction was required, such as control over remote systems (robotic arms and machinery) or communication between multiple people; these mandate that the video being viewed was as close to real-time as possible, in order to ensure efficient control and interaction.

It should be highlighted that the large scale user base of some of the described applications, would be best served using a combination of the system presented in this chapter, combined with the OpenFlow CDN described later in Chapter 6. Additional information on large scale streaming applications can be found in Chapter 6, where these applications are discussed further in relation to the proposed OpenFlow CDN system.

5.3 Test Bed Design

Relevant technologies utilised within the proposed migration system will now be briefly discussed; however, more in depth description of OpenFlow and its controller applications can be found in Chapter 2.

5.3.1 OpenFlow Implementation

Further to the information provided on OpenFlow within Chapter 2, it is important to highlight the features being relied on for the operation of the proposed migration system. Only a small subset of features available within OpenFlow are utilised, with one of the main features being flexible flow rule insertion, which allow redirection of packets beyond what standard networking systems can easily achieve. Another feature of OpenFlow that is utilised, is the ability to modify packet header information; this allows packet header information of the receiving device to be modified as it traverses the switch. Packet header manipulation is integral to the system, as it allows a second transcoder machine to receive traffic destined for the first transcoder, without any modification to the machine's networking processes. Additionally, the centralised control aspect is also utilised in the proposed system, to maintain control of the two switches throughout the migration process.

The described features converge on the higher level concept of OpenFlow, which is to allow non-standard switching processes to occur regardless of the protocol used. This main concept of OpenFlow allows technologies to be created that were previously not easily possible; without this advancement, it is doubtful that the presented system would be feasible at its current performance.

5.3.2 Open vSwitch (OVS)

One of the tools that is heavily utilised throughout the prototype systems detailed in Section 5.4 and Appendix A, is Open vSwitch (OVS) [149, 150], which is a software based OpenFlow switch implementation. OVS allows a single machine to virtualise OpenFlow switching functionality without the cost of dedicated switching resources. The OVS platform is far into its development with a large array of features; because of this, it has been adopted in many well used software platforms [181, 210]. Recent versions of OVS far surpass the capabilities of hardware based OpenFlow switches produced by manufacturers, mainly due to their fast adoptions of OpenFlow revisions. OVS is able to achieve its rapid advancement based on its purely software design, which is not restricted by any hardware requirements; however, this software based design does limit its performance when compared to a dedicated hardware switch [124].

One main advantage of using OVS over physical hardware, is the ability to rapidly create a large OpenFlow based network with little financial cost. This is due to an OVS instance being able to create multiple virtual bridges (switches) on a single machine, while also allowing connection of both external ports and virtual bridge ports to the virtual bridges. Therefore, OVS provides an ideal system for testing scenarios, as it would not be feasible to obtain expensive OpenFlow hardware without knowing the feasibility of the research idea. OVS is also an open source application allowing free usage, but more importantly, it has an active development community that is able to maintain a stable and well tested design; this large community also contributes to the previously mentioned high rate of development, which allows OVS to introduce features long before switch manufacturers.

For these reasons, OVS was used within the prototype stages until OpenFlow hardware with similar capabilities was available; it should be noted that the limitations with OVS detailed later in Section 5.4, were due to the network and compute resource constraints, not a lack of performance associated with the OVS software.

5.3.3 Pica8 OpenFlow Switches

After initial testing using OVS installed on a standard desktop machine, it became necessary to move development to OpenFlow switching hardware with improved performance. However, it was important that all the features previously utilised within OVS were maintained, such as the OpenFlow version compatibility; this was important to allow the current implementation of the system to be ported across to the new hardware, without significant modifications. However, it should be noted that once the system is developed as an extension to an OpenFlow controller, the system will be able to function with any OpenFlow enabled switch; it is only the current implementation of the custom migration control application that limits the current device selection. After research into possible options, the OpenFlow switches available from Pica8 [211] seemed to provide the exact feature set required, as well as allowing previously developed control mechanisms to be maintained; this significantly improved the speed and simplicity of converting the migration system to use physical switching hardware. The specific model that was utilised was the Pica8 P-3290 switch [212], which provided the performance and functionality required for the 4K transcoder migration scenario.

One of the main advantages of using Pica8 switch hardware is the fact that they use a modified version of OVS as their OpenFlow implementation. This provides the same accessible control interface and feature set that was being used previously in the testing stages, but with the added benefit of improved performance which was able to handle 4K traffic. It is important to note, that although the Pica8 switch is based on a software implementation of OpenFlow, it achieves increased performance by offloading certain switching tasks into hardware; utilising specific hardware elements during the packet matching process allows it to achieve performance not possible with purely software based implementations. A further benefit of the Pica8 switch over OVS running on standard machines, is the additional ports available on the Pica8 switch; on a standard machine you are limited by the small number of network interfaces available to link with OVS, whereas the Pica8 switch has 48 1Gbit/s ports available, with the option of extending these with 4 10Gbit/s interface modules. The large number of ports provides a large improvement in performance, as you are no longer required to route all traffic through a limited number of ports using virtual interfaces to separate traffic. Further to this, it should be highlighted that separating traffic using VLANs on virtual interfaces or using other tunnelling protocols, it can introduce large overheads that affect the throughput of the system.

5.3.4 Floodlight Controller

For all implementations of the migration system, including initial testing phases, Floodlight [133] was used as the OpenFlow controller. However, it is important to highlight that flow rules added manually into the flow tables are able to override the default rules put in place by the Floodlight controller. This override behaviour is controlled using a priority system that can be configured when inserting flow rules into the flow tables. Floodlight was only utilised to provide normal source MAC learning functionality for the OpenFlow switches, whereby the switch associates incoming packet source MAC addresses with an interface, in order to allow simple forwarding of packets without flooding all interfaces. Without a controller application to provide this simple functionality, the OpenFlow hardware has little to no functionality; however, most OpenFlow

implementations include a default low priority flow, which floods traffic out all ports when a packet is received to emulate a standard layer 2 hub. The Floodlight controller was configured to communicate with the switch hardware using out-of-band communication, as it allows it to bypass any OpenFlow rules which may disconnect the controller from the switch; this is standard practice for OpenFlow controller communication, as well as a lot of other network control systems.

Although Floodlight was used to maintain normal switching functionality, it was not utilised to insert the custom flow rules required during the migration process; this is mainly due to time constraints in developing and integrating the control system into a Floodlight module. The method that was used for inserting these custom flow rules is detailed in Section 5.3.4.1. Additionally, as previously stated, more in-depth information on the Floodlight controller can be found in Chapter 2, along with available alternatives.

5.3.4.1 Flow Rule Insertion

For the current implementations of the migration system presented in this chapter, the flow rules are inserted directly into the switches using direct system calls through Secure Shell (SSH) connections; these SSH based system calls were initially performed using batch scripts with password-less SSH access, but later versions utilised a custom Java application developed to orchestrate migration system. Both these implementations bypass the Floodlight controller, which is mainly for simplicity during development; however, an additional benefit of bypassing Floodlight was that it minimised any issues that could be attributed to the communication between the controller and switch, including any delay issues that it might have caused. The separation of the migration control from the Floodlight controller enabled rapid resolution of any issues, with the confidence that the controller was not the cause.

In the future, it may become necessary to implement the current system to work with the controller application directly; this may be due to lack of SSH access to the switching resources, or the use of OpenFlow devices which do not utilise OVS. There are various ways to implement this idea, but they are left as future work due to time constraints; however, a brief discussion of how this could be achieved is presented in Section 8.6.3.

5.3.5 Traffic Flow Monitoring

As previously discussed, for the system to become fully automated and to decide an optimum switching time, it would be required to either use the heuristic algorithm developed in Chapter 4 and/or a custom traffic monitoring system. For traffic statistics on a flow rule based granularity, it is possible to query switches for detailed information stored within their flow tables; this includes information such as the number of bytes and packets that have matched a given rule. However, using flow rule based statistics is only accurate when there are flow rules configured to identify the exact traffic you intend on monitoring. To overcome this issue, it is possible to insert fine granularity flow rules that match a coarser granularity rule, in order to separate the packet statistics within the flow tables. Unfortunately, creating a large number of fine granularity rules would only be an effective solution in certain situations, as unnecessarily increasing the size of flow tables will cause unwanted load on the switch. Using the information gathered from the flow tables as described would allow the calculation of bitrates of client video streams, as well as the

number of clients; this would then provide the information required to determine if a migration would be beneficial.

As an alternative to flow rule based monitoring, there are other monitoring systems that could be utilised; two possible candidates include NetFlow [154] and sFlow [153], which can be included within the OpenFlow infrastructure and allow remote monitoring of multiple systems. They are both similar in design, in that both require switches to be configured to send packet statistic information at set intervals to a remote machine through standard IP communication. The information can then be viewed and processed at the remote computer, which would allow a centralised overview of traffic utilisation across the network. Further details including a detailed comparison of the two monitoring solutions can be found in Section 2.6.5. It should be noted that for some of the prototype stages, sFlow was utilised to monitor traffic flow for both diagnostic and performance information; however, it will not be discussed further, as it was not essential to the proposed system and was only used for the described monitoring during the prototype stages.

In order to automate the process of migration when utilising the heuristic presented in Chapter 4, one of the described traffic monitoring solutions would need to be integrated with the migration control system. Combining these together would provide the traffic data required to determine whether a migration should be initiated, as well as the mechanism to perform the migration if required.

5.4 Prototype Development

This section describes and discusses the issues discovered and overcome during the various stages of prototype development. These provide a rationale for many of the design decisions found within the final approach, which is presented later in Section 5.5. This discussion includes various naive implementations, as well as details of some of the less successful techniques which were deemed unsuitable for the final UHD migration system. Detailed implementation details of prototype stages can be found in Appendix A, with prototypes ranging from using simple user generated packet relay migration to VM based migration of live HD transcoding. The final prototype implementation can be found in Section 5.4.2, which describes the final challenges and issues encountered before the final UHD migration system in Section 5.5 was developed.

5.4.1 Observations of Early Prototype Systems

During initial implementations of prototype systems, such as the system described in Appendix A.1, it became clear that OpenFlow was able to significantly improve the performance of migrating network dependent resources. The improvement in performance is specific to network orientated resources, as the system only attempts to reduce the switchover of the streams from the perspective of the client receiving the stream. This is in contrast to a lot of performance metrics used by other migration optimisation techniques, as they often only analyse the downtime in the resource that was migrated or the speed of the movement for the resource; the fact that they do not look at the effect on clients receiving service from the resources is a clear area of interest when dealing with real-time transcoding. This concept is further discussed in Section 2.3.2.3 and Section 2.8.

Initial prototypes used VLC player as both the initial server and client application, while FFm-

peg [15] was used as the transcoder software running on the VMs that were migrated. However, using VLC player as the initial streaming server became unsuitable when handling video content above SD resolution. For prototype systems beyond the SD based system shown in Appendix A.2, FFmpeg replaced VLC player as the initial streaming application. The switch to FFmpeg allowed quality to be maintained throughout transmission while also allowing improved control on video bitrate and image quality. Further discussion on the reasons for the switch can be found in Appendices A.2 and A.3.

Throughout the prototype stages described in Appendix A, one reoccurring issue that was encountered, was the restrictions caused by the large processing overhead of transcoding live video content. This often meant prototype systems were unable to reliably stream HD content at a reasonable quality in real-time, as both transcoder and network resources were constrained. At each stage in development, the processing resources being utilised were increased, moving from a single host machine running all VM and OVS instances, to multiple servers hosting the VMs and OVS instances. The increase in processing resources at each stage did enable improved quality video to be streamed, however, HD content was not successfully able to be transmitted until the prototype described in Section 5.4.2 was developed. Although the processing resources were a significant factor in the restriction of video streaming, the protocol used to connect VMs to the OVS switch was also a large contributing factor in some instances. For most systems it was possible to connect the resources with direct virtual connections, as the OVS instance was on the same machine as the other resources; however, the prototype system described in Appendix A.3 required tunnelling protocols to achieve the connection with the OVS instance, which was located on a different machine. This was a significant restriction in the performance of the system, with a lot of research into the possible solutions being performed. Discussion of suitable tunnelling protocols and the issues associated can be found in Appendix A.3; it should be noted however that tunnelling was not required for the final UHD system presented in Section 5.5, as a dedicated OpenFlow switch was used with direct L2 connections.

It was important when designing the prototype systems that the migrated transcoder would begin transcoding the content stream as soon as possible, in order to reduce the disruption to the client. During the development and testing of the system presented in Appendix A.2, it was discovered that although FFmpeg can be tuned to initiate transcoding rapidly on receiving a stream, the reduction in the "probesize" parameter caused instability in content detection reliability. The result of this would cause streams to be incorrectly identified, halting the transcoding process and disrupting the stream to the client. To overcome the issue of instability while also achieving rapid switchover at the client, certain actions within the migration process had to be performed at specific stages during the migration. This concept is first discussed as an idea within the prototype system in Appendix A.3, where it introduces the concept of allowing both the initial transcoder and the copy of itself to function in parallel; this allows FFmpeg on the newly copied transcoder to analyse the stream correctly before the switchover occurs. Details of this initial idea and its results can be found in Appendix A.3, however, a more advanced version of this is presented later in Section 5.5.

5.4.2 Final Prototype Discussion

The final stage of prototype development involved using dedicated machines in place of the VMs for the roles of client, server, OVS and transcoders. The setup utilised relatively high powered servers, custom built to provide 4K playback, so they should have been adequate resources available for the system to be successful. The servers provided considerable improvement in computational resources, as the aim was to overcome the shortcomings of the previous prototype systems and deliver improved image quality and experience during migration. Using the 4K viewing system as a client also provides an ideal situation for analysing the viewer experience, as the stream can be easily monitored at all stages throughout the video stream; additionally, since 4K projectors were utilised, the displayed video could be more closely inspected for artefacts and quality during the stream. It should be noted that although 4K viewing equipment was being used here, only full HD content was being streamed and tested; the reasons for this will be discussed later in this section.

Unfortunately, a hardware OpenFlow enabled switch was not currently available for use, so a physical machine was used to run OVS. This presented improved performance over the VM based scenario since it had a dedicated network port assigned only to itself and not other VMs; additionally, using a dedicated machine for the OVS instance provided increased compute resources, which is required for the increased traffic that HD content generates. The basic overview of the described implementation is shown in Figure 5.3, where it can be seen that 5 machines are required for its operation: 1 for OVS and Floodlight, 2 for the transcoder software, 1 for the client which would be connected to the 4K projector and 1 for the server to send the initial video content to the client via the transcoder machines. Additionally, Figure 5.3 also shows how each of the utilised machines used a single network interface within the system; however, another interface on each machine was also required for the control network, which is not shown in Figure 5.3. Similar to the previously developed system described in Appendix A.3, each machine was connected to OVS using GRETAP interfaces to tunnel the traffic over the standard networking infrastructure. A Black Diamond 12804 Extreme switch was used to physically connect each of the machines together using 1 Gbit/s connections, which was hoped to be sufficient for the HD content experimentation; the use of this high performance switch aimed at minimising any underlying network issues that might have restricted the migration system. However, as discussed later, the 1 Gbit/s link to the machine hosting OVS presents a severe bottleneck for network traffic. Figure 5.3 provides a clear overview of this issue during migration between the two transcoders, as during this time both transcoders are transmitting content between the OVS switch; it can be seen that during this stage there are 6 HD streams traversing the single link connected to the machine hosting OVS; the combination of all the machines traffic overloads the link, causing both congestion and dropped packets for the video stream. This is one of the main bottlenecks that restricted the testing to HD content, rather than the desired 4K content.

After initial testing, it became clear that finding suitable FFmpeg settings would be difficult. This is, in part, due to the fact that it is a complex task to discover a configuration that enables real-time transcoding while not degrading the image quality of the video beyond a point that would significantly impact the clients viewing experience. The configuration was also constrained by the previously explained capacity restrictions, which were caused by the OVS machine with a

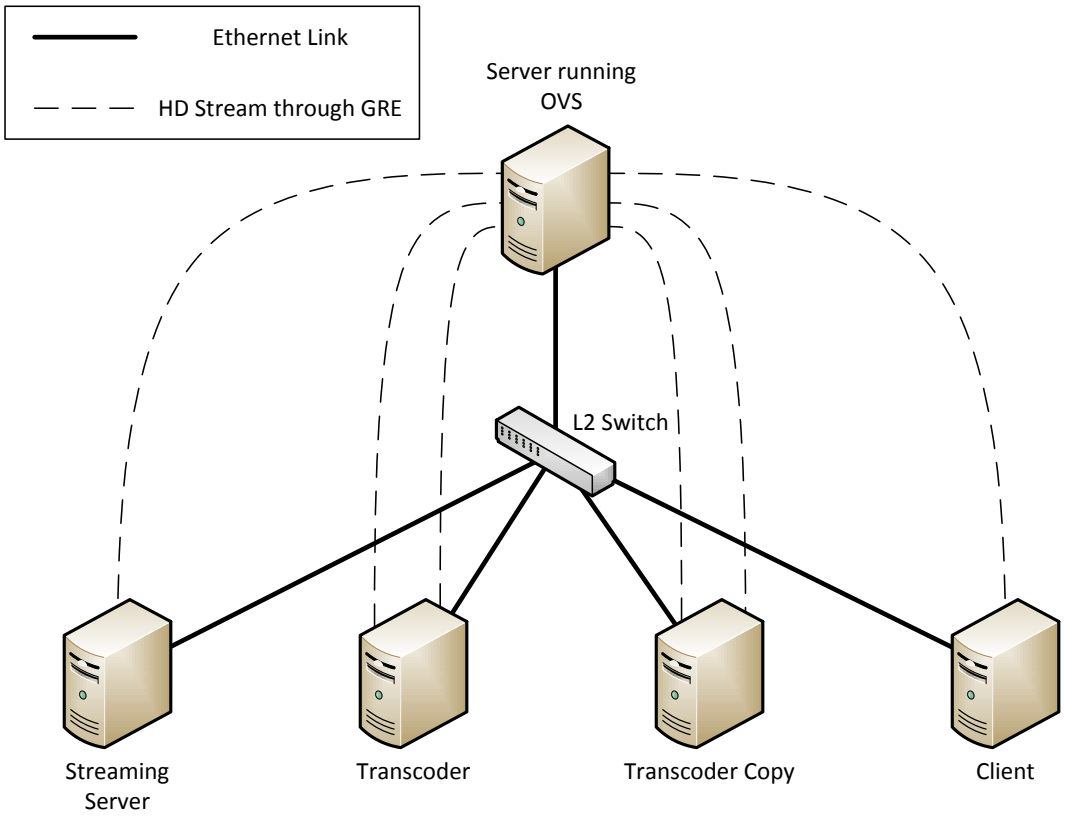


Figure 5.3: Overview of the bottleneck caused by the single link to the OVS machine.

single network interface; this limited the bitrate that could be used for the content. A related issue was the significant overhead caused by the processing of Generic Routing Encapsulation (GRE) packets on the machines; this processing overhead placed a further restriction on the bitrate of the video that could be used, since machines were not able to perform the GRE encapsulation and decapsulation process fast enough for high bitrate content. If the bitrate was configured too high, packets started to be queued for GRE processing at machines, which caused artefacts and stuttering video to appear at the client machine.

It is important to note that lost packets in the network can cause an extensive reduction in image quality, as well as playback stability, especially when using UDP with MPEG-TS. Additionally, although the packets in the current system do not travel a significant distance and only travel through a single (more than adequate) switch, packets arriving out of sequence is also a problem, as UDP does not have a re-ordering mechanism built in. Lost packets in these situations can be caused by a combination of issues, such as the transcoding machines not being able to receive and process the video content in time for real-time retransmission to the client. This is due to the significant steps that are required when the video content is received at the transcoders network interface. Firstly, the video packets need to be decapsulated to extract the packets from the GRE tunnelling headers; the video packets can then be processed using FFmpeg using the desired configuration. Finally, before re-transmission, the packets also need to be encapsulated with GRE tunnelling information again. The described delay in packet processing can cause UDP buffers in the machine to become congested, causing packets to be dropped when queue space is unavailable. A symptom of lost, or out of order packets, can include a corrupted video stream,

which can be further corrupted with the use of transcoding along its path. As previously discussed, this additional transcoding task can introduce further delays during streaming, especially if the bitrate overflows the UDP buffers; this can further reduce video quality, as well as amplifying any distortion artefacts already present in the video. Unfortunately, there is a significantly high probability of stream corruption, since UDP does not provide any delivery guarantees. There are several solutions that can be used to remedy this, including reducing the bitrate of the video to stop machines becoming overloaded; this will work to some extent, but will usually come at the cost of degrading the image quality of the content. This is not a desirable scenario, as a reduction of video quality opposes one of the main aims of the system, which is to provide an improved viewing experience to the client. There are alternative codecs available that can both reduce bitrate and improve resiliency to packet loss, while also still maintaining video quality. Their ability to achieve this relies on leveraging improved video compression techniques and including error correction mechanisms; this can achieve a reduction in artefacts and corruption when packets are lost. Although these codecs can provide some resilience to a small amount of packet loss, the impact of large numbers of lost packets will still produce a significant reduction in the quality experienced by the client. Furthermore, these more advanced codecs generally require increased processing time during both encoding or decoding to achieve the additional features, which would not be suitable for this real-time streaming system. There are codecs that have the ability to leverage certain hardware to increase encoding performance, but these are not investigated in further, as the focus is on software based transcoders that are able to be migrated.

Rather than attempt to remedy the packet loss issues at the application layer as discussed, it would be beneficial to target the issue at a lower level in the networking stack. In this case the use of the transport layer protocol User Datagram Protocol (UDP) can be seen as a possible improvement area, since by default, UDP does not feature a method of ensuring packet delivery; this provides a specific area that can be focussed on when attempting to address packet loss issues. Alternatives to UDP are numerous, but very little of these would be suitable for use in the proposed scenario. TCP is an obvious alternative to UDP with its reliable data transmission features, as it includes retransmission of lost packets; additionally, TCP also provided almost flawless video transmission between the server and client when tested. However, it remains unsuitable for the proposed scenario for multiple reasons. One such reason is that TCP requires establishing a connection with the destination machine before streaming, which does not allow the switching of transcoders in a feasible way, due to the break in connection. Additionally, with the current software platforms being used, TCP would make delivery of the content difficult to clients joining the video stream after it had begun. Similar to these reasons, just the use of a transcoder between the client and server creates a far more complex system than that of the UDP based version. TCP was known to be unsuitable before initial testing due to these reasons, but it was used to diagnose other video corruption issues that were being experienced during FFmpeg configuration. From research into streaming specific protocols, Real-time Transport Protocol (RTP) [213] is a possible candidate to replace pure UDP. RTP was designed to transport real-time video, audio and simulation data across networks and provides certain characteristics that allow for more efficient data streaming. RTP is most commonly run on top of the UDP protocol to augment its streaming capabilities, but it can be run on TCP if desired. An important feature of RTP that

should be highlighted is the inclusion of a sequence number in the RTP header, which allows for easy ordering at the receiving client [213].

It is, however, important to bring attention to the fact that RTP is still based on UDP unless configured otherwise, which means although it builds on top of the UDPs feature set, it does not solve all its issues. One of these issues that unfortunately still causes problems even when using RTP, is that the transcoders are still unable to receive, transcode and send out data fast enough to meet the streams demand. Furthermore, RTP adds to these delays with the inclusion of the extra processing required to handle the RTP header processing and the reordering of packets. Similar to that of using TCP, RTP does allow a direct stream from the server to the client to function without issue, but a stream being intercepted by the transcoder and forwarded on to the client would perform worse than when using standard UDP. The main reason for this is the reordering of packets being a lengthy process, which causes the network card buffers to overflow similar to the issues associated with standard UDP.

From analysis and testing the different configurations and protocols, it became apparent that the best solution for the current scenario would be a combination of UDP and RTP; with a UDP stream being sent from the initial server to the transcoder and then using RTP for the transmission between the transcoder and the client. Initial results from analysing the viewing experience and video quality at the client allowed this conclusion to be drawn. Furthermore, this combination allowed an increase in bitrate for the video without causing further lost packets, which resulted in a significantly increased video quality. This combination was also implemented due to the feasibility of its use with the migration system, which requires that the stream be diverted during transmission. The described system allows this to occur without any additional work, unlike if a connection based protocol was used. It is evident that the variation of protocols and methods of combining them, presents a significant number of combinations that could be attempted in order to provide marginally improved results; however, the UDP and RTP combination was deemed sufficient at this stage of development. Determining precise optimum codecs or designing others, are beyond the scope of this research, as it is the migration system that is being focused on and developed. It is possible to assume that the RTP client on the transcoder could be re-engineered to improve its performance within this specific system, allowing it to be used without the issues associated with processing packets at the transcoders; however, this concept is left for investigation as a possible area for future work, with further discussion found in Section 8.6.3.

5.5 Final System Design

The completed system design was constructed and tested using two separate testbeds; both utilised the Pica8 P-3290 OpenFlow switches detailed in Section 5.3.3 and both were configured to stream 4K content during the migration scenario. The first testbed involved the use of NetEm [214] to mimic WAN link characteristics on an Ethernet LAN link. The testbed was established during a collaboration project with the Poznan Supercomputing and Networking Center (PSNC) in Poland [215] and the International Center for Advanced Internet Research (iCAIR) at Northwestern University in Chicago [216]. The original plan for this collaboration project was to utilise the 1Gbit/s L2 WAN connection established between PSNC and the Starlight facility in Chicago (part of iCAIR at Northwestern University) to test the presented migration system. Unfortunately, after

some brief initial testing, the transatlantic undersea cable carrying the lightpath for the WAN link was damaged and remained non-operational throughout the assigned project time frame. This necessitated the use of WAN emulation for the testbed. However, emulation provided the benefit of investigation into the effects of varying WAN link latency on the migration process, including results for $<$ 1ms, 125ms and 250ms round trip latency values; further details for the reasoning behind these values can be found in Section 5.5.3. Additionally, using network emulation also provided an insight into the issues associated with using non-OpenFlow interconnecting devices within the system; these issues will be described later in Section 5.5.4. After the successful operation of the migration system using the emulated WAN link, it was determined that it would still be desirable to test the migration system using a real transatlantic WAN link. Further collaboration with iCAIR resulted in the establishment of a 1Gbit/s L2 transatlantic WAN link between Essex University and the Starlight facility in Chicago. This enabled the second testbed to be constructed, which allowed the successful testing of the migration system over a real world transatlantic WAN connection while streaming 4K content.

NetEm is a network emulation software which allows several characteristics of WAN network links to be introduced to a standard Ethernet link; this includes factors such as delay, packet loss and jitter. This is essential for testing performance on a system that will be used over WAN links, which is often difficult to achieve because of WAN availability issues. During testing, the only parameter varied using NetEm was packet delay, which was configured to mimic the latency found on WAN links such as the transatlantic link within HPDMnet [1]; this was assumed to be a similar environment to the architecture that would be found within a single operator network.

The system architecture implemented for both test systems, use a single client receiving 4K video content from a server. This is a simplified example of the proposed scenario, which is presented as a system to provide content to multiple clients at varying resolutions and bitrates. This simplified example was used due to restrictions with available resources, as well as time restrictions in completing the work. Initial results, however, have shown that the system functions successfully with multiple clients requesting the same content at different resolutions. In a multi-client scenario, the transcoder receives a single stream at the highest requested resolution; the transcoder then processes the content and uses application layer multicasting to distribute the content at multiple resolutions to the respective clients.

It should be noted that in both systems, compressed 4K content was streamed at a very high quality, with any visual loss of quality being minimal when compared to uncompressed versions. The compressed version was required due to restriction in both the hardware processing power and the network infrastructure, which is only capable of transporting content at up to 1Gbit/s; this is far below the required 9.6Gb/s required for uncompressed 4K content that has a reasonable frame rate and colour depth. The bitrate for the 4K content used during testing was measured at the client and was found to vary between 3 MB/s (24 Mbps) and 12 MB/s (96 Mbps), with an average rate of around 6.7 MB/s (53.6 Mbps); this bitrate is similar to some of the 4K content streams found on YouTube.

UDP streaming was utilised in both streaming systems for all transmissions, due to RTP causing issues with the streaming of 4K content. Some of the issues with RTP were found to be caused by the limitation of the network cards and the transcoding machines; these limitations

relate to the speed at which they were able to encapsulate the RTP streams, as they were not able to successfully process the large number of packets from the real-time 4K content stream. One of the RTP based issues observed during the use of the UDP and RTP combination mentioned in Section 5.4.2, was that packets arrived out of order and delayed; these delays caused a large number of playback issues at the client due to the reordering process, including video artefacts and stuttering occurring during playback. However, the previous issues with UDP packet loss in Section 5.4.2 were able to be overcome, which enabled its use at all stages in the system. A large improvement in UDP packet loss and packet ordering issues was achieved through the use of the Pica8 physical switch, rather than using OVS on a single machine; this was already known as an issue, but it was beneficial to see the large and direct improvement that could be achieved with this single change. Another additional improvement to the system was the use of improved transcoder machines; these were able to process the increased bitrates of 4K content in real-time, in the same way the previous machines were able to with HD content.

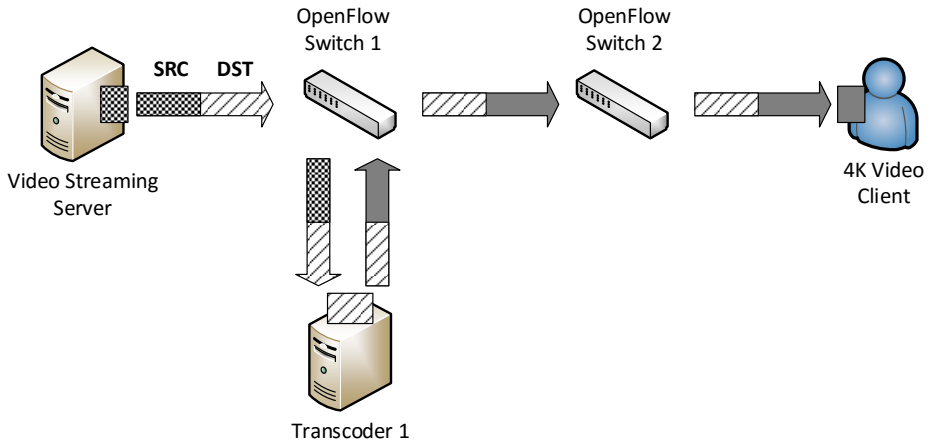
It should be noted that VLC player was configured to use up to a 200ms network cache, which helped to maintain playback during the switchover time. This is important as VLC by default will use a 1 second network cache, which would allow unwanted delays in the real-time playback scenarios that the system is designed for. However, during testing with the emulated WAN link with 250ms latency, VLC's network cache was required to be increased to 400ms to maintain smooth playback; this seems to be more an issue with the variation of packet delay rather than issues with the switchover process, since gathered results showed similar switchover times.

5.5.1 Migration Process

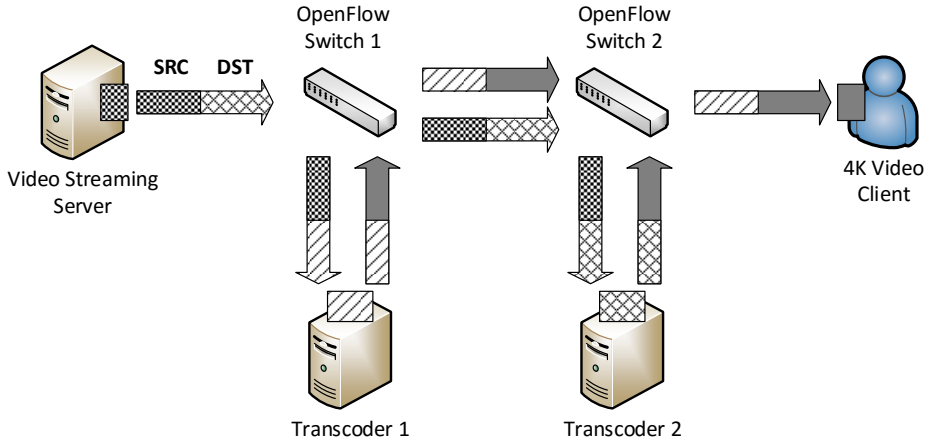
The migration process has many steps to ensure a near instant switchover of streams, which is required to reduce the interruption in the playback of the stream at the client. Although playback systems can handle some interruption/delay in the stream, if it extends more than the previously mentioned network cache, the stream will halt and require a longer time to be re-established. The longer re-establishment period is due to the fact that after a set time, playback systems tend to treat the delayed stream as a new transmission which requires content analysis to ensure correct playback.

The basic premise behind the final OpenFlow migration system is that it both replicates the stream at the switches, while also redirecting and rewriting packet headers to enable the two transcoders to process the stream in parallel; however, during this time of parallel processing, only a single stream from the initial transcoder reaches the client. The parallel design is essential to allow for the server to establish the new transcoders MAC address in its Address Resolution Protocol (ARP) cache table, without affecting the content stream to the original transcoder currently serving the client. Once the new MAC is established in the server ARP cache, the old transcoder can be disabled while simultaneously enabling the stream from the new transcoder; this process enables a near seamless switchover of the streams at the client.

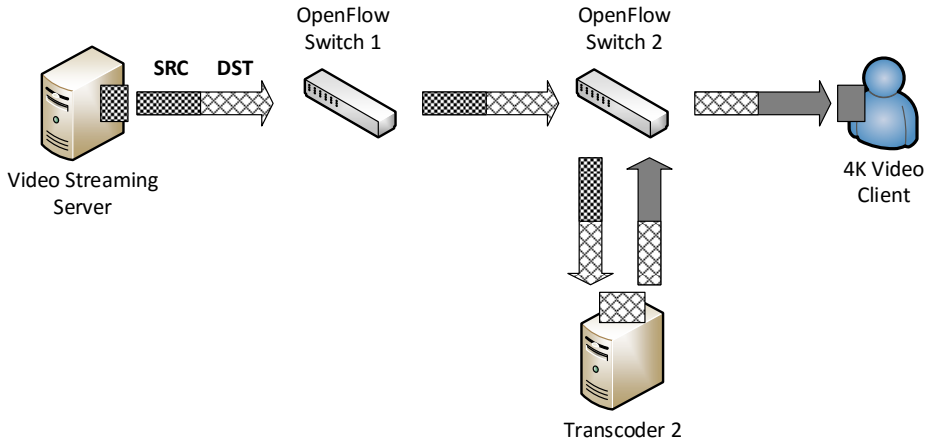
Figure 5.4 gives a simplified overview of the system operation at multiple stages of the migration process. However, there are many sub-stages involved between these stages, which aid the near seamless migration. Both the described testbed systems operate in this configuration, however, the difference between them is related to the WAN link between the two switches. The



(a) Before Migration



(b) During Migration



(c) After Migration

Figure 5.4: Traffic flow for the migration process.

difference relates to the WAN connection being either the emulated version in Section 5.5.3 or the transatlantic link version in Section 5.5.4. It should be noted, however, that the transatlantic based system differs very slightly in relation to the duplicate streaming, with this difference being detailed in Section 5.5.4. Figure 5.4 provides a clear view how the ARP entry in the server is replaced with the new transcoder MAC, even before the switchover takes place; this important aspect enables a much faster migration and also prevents any packet header modification being required after the migration event. Figure 5.4 also shows that during migration, the streams are duplicated and sent for processing at both transcoders. It is important to highlight that only the initial transcoder stream reaches the client during this time, as this enables the previously mentioned ARP entries to be updated; this period of both transcoders running, provides the time required for the transcoding software to analyse the content stream before it can begin transcoding. These are important steps in the system that allow the switchover to occur within a very short period of time; results for the observed migration times are presented in Section 5.6.

Figure 5.5 shows the process of flow rule configuration during the migration process; this occurs during the stages shown in Figure 5.4. The related flow rules can be found in Appendix C.3.

The wait times presented in Figure 5.5 were chosen as they presented the best performance in the current system, but they can be adapted as necessary for specific scenarios. These wait-times are not critical to the overall performance of the system, which is instead influenced by the gap in transmission at the receiver, which should be as small as possible. The first wait time is less crucial than the second, since the migration process can actually continue without the port open at that exact step in the migration process. However, the system reliability and performance is increased if this wait time is introduced, as it allows the second transcoder to establish its network connection fully before being sent the data heavy video content. The second wait time is more crucial to the system, as it allows transcoder 2 to begin its transcoding task, which involves populating the receiving buffer and analysing any time-dependent factors such as motion vectors. This buffering period can be adjusted in the transcoder, but if it is set too low, there will be issues identifying the content and it will fail to process the video stream. For this reason, the buffering time was left to its default setting in the transcoder and the wait time is used to obscure the start up delay in the transcoding process. With the current configuration, it is important to highlight that duplicate traffic will be streamed over the WAN link during the 4 second wait time; this short period allows both transcoders to operate in parallel at this stage of the migration process. It should also be noted that the second wait time also provides time for the server ARP table to be updated with the new transcoder MAC address, as this allows the correct MAC address to be used by the server at the time of switchover. As previously described, the updated ARP entry enables a smooth transition to the new transcoder, while also removing any requirement for packet header modification after the migration process completes.

A description of Figure 5.5 is as follows:

1. Enable both transcoder 2 and the port that it is connected to on Switch 2.
2. Wait a set time, e.g. 2 seconds, for the port to enable.
3. Then in parallel:
    - On Switch 1:

- ◇ Duplicate the stream from the server, sending the unedited stream across to the second switch, while sending another to transcoder 1 but with the MAC destination field rewritten to the transcoder 1 MAC.
- ◇ Send ARP requests for the transcoder IP address to Switch 2.

- On Switch 2:
  - ◇ Send packets destined for the transcoder 1 MAC to transcoder 2 with its MAC destination field set to the transcoder 2 MAC.
  - ◇ Send packets destined for the transcoder 2 MAC to transcoder 2.
  - ◇ Send ARP requested for the transcoder IP address to transcoder 2.
  - ◇ Drop any packets received from transcoder 2.

4. Wait a set time for the server to send out an ARP request to the transcoder, which will now be redirected to transcoder 2.

5. Then in parallel:

   - On Switch 1:
     - ◇ Disable transcoder 1 and the port it was connected to.
     - ◇ Send packets from the server to switch 2.
     - ◇ Delete old stream duplication flows that are no longer required.
   - On Switch 2:
     - ◇ Send packets from transcoder 2 to the client.
     - ◇ Delete old flows that drop packets as they are no longer required.

It is important to reiterate, that after the migration process has finished, there is no packet header manipulation being performed. This is essential to maintain the scalability of the system, as packet header manipulation places a considerable load on the switches. Furthermore, the also means that the integrity of the layer 2 architecture is not diminished, since there are no two devices with the same MAC on the network, even for a short amount of time. If this was not the case, duplicate MAC addresses could cause issues during migration, where the machine is cloned along with the MAC address; this can cause confusion to networking devices which are trying to identify locations of devices using source MAC learning. Additionally, if the system did rely on continual packet header manipulation, it would most likely cause issues when handling a large quantity of real-time parallel streams.

5.5.2 Control Software

The control software being used for these tests interacted directly with the OVS on the Pica8 switches to set up flow rules. In a production system, it would be integrated directly into an OpenFlow controller, such as the Floodlight controller currently being used to handle normal switching operation. Further to this, the software was designed in Java to allow easy integration with the Java based Floodlight at a later date. Once this integration is completed, the transcoder migration process could then be initiated and controlled directly from Floodlight, with the optimisation algorithm presented earlier in Chapter 4 being used to detect a migration condition.

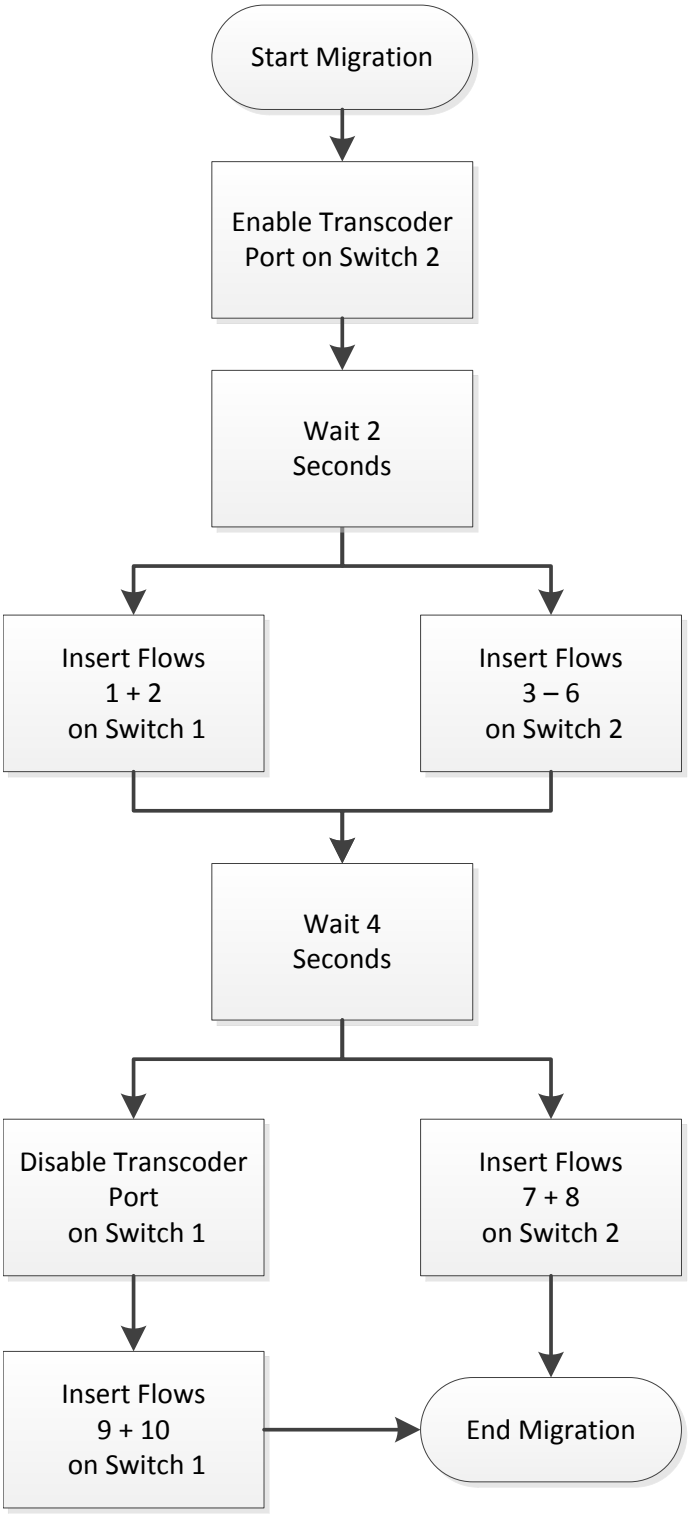


Figure 5.5: Optimised flow process for the migration using a network emulated link. See Appendix C.2 for detailed flow descriptions.

The current implementation of the control software allows concurrent control of multiple switches, using synchronised threads with direct SSH communication to ensure correct timing of flow rule insertion. Further information regarding the custom control application can be found in Appendix C.2.

5.5.3 Local Testing with NetEm

As previously described, this testbed system was implemented while visiting PSNC in Poland, using NetEm to emulate the latency of a WAN link so that it was possible to test the system with varying latency levels. All the required machines in the system were run on dedicated high performance physical machines, which allowed the best performance available, without issues of resources becoming the restricting factor. The client machine was connected to a 4K display for assessing the visual quality of the video during normal streaming, as well as studying artefacts and other video issues during migration. Figure 5.6 shows the configuration of these resources at the initial stage, before migration takes place. It can also be seen in Figure 5.6 that the link between the second switch and transcoder is currently disabled; this link is enabled as part of the migration process, along with the disabling of the original transcoder link at the correct stage. More detailed machine specifications for the system can be found in Appendix B.1.

The values of latency tested using this configuration were based on a private network infrastructure link, such as those used in HPDMnet. The link between PSNC in Poland and the Starlight facility in Chicago has a round-trip delay of 125ms, which presented a useful metric to test the system with. A 250ms delay was also tested to provide an insight into the performance of the system within high latency environments. These values were also compared against a local LAN link with negligible latency (< 1ms), to show how the system is not adversely affected by the link delay. This was an important constraint of the system to focus on, since the content being

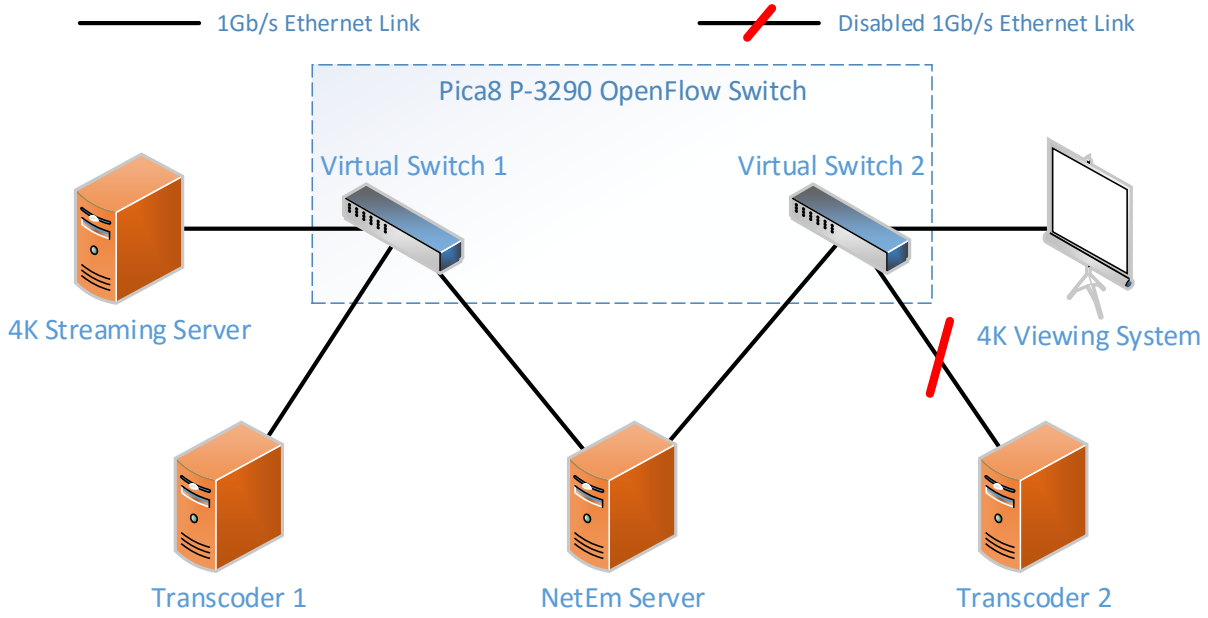


Figure 5.6: Initial state of NetEm testbed architecture.

streamed is envisioned as real-time content which is also heavily influenced by network latency.

5.5.4 Transatlantic Testbed

In this testbed architecture, the previously described Pica8 P-3290 switches were located at the University of Essex and the Starlight facility in Chicago. They were connected over a transatlantic WAN link with 1Gbit/s capacity, which was configured as a light path through HPDMnet; furthermore, the WAN link was measured and found to have an average measured round trip latency of 118ms, which is similar to that of the 125ms latency test within the NetEm testbed. The described WAN link is the focus for the reduction of traffic, as the migration operation intend to move the transcoder from one side of the link to the other. Figure 5.7 shows the configuration of these resources at the initial stage, before migration takes place. It can also be seen in Figure 5.7 that the link between the second switch and transcoder is currently disabled; this link is enabled as part of the migration process, along with the disabling of the original transcoder link at the correct stage. More detailed machine specifications for this testbed system can be found in Appendix B.2.

During initial testing of this system, it became clear that there were issues that had previously been seen in the network emulated testing. One of the main issues was the confusion of intermediary network devices along the path between the two switches; this was due to the packet header manipulation that occurs during the migration process. From the perspective of non-OpenFlow networking devices, it appears that there are duplicate IP addresses on the network, as well as the knowledge that Ethernet packets were being sent along incorrect links according to their MAC address. Network devices are able to determine this due to their inspection of packet header information, as well as the use of IP and MAC address caching which is used to improve switching performance. Unfortunately, this causes non-OpenFlow enabled devices to block the duplicated

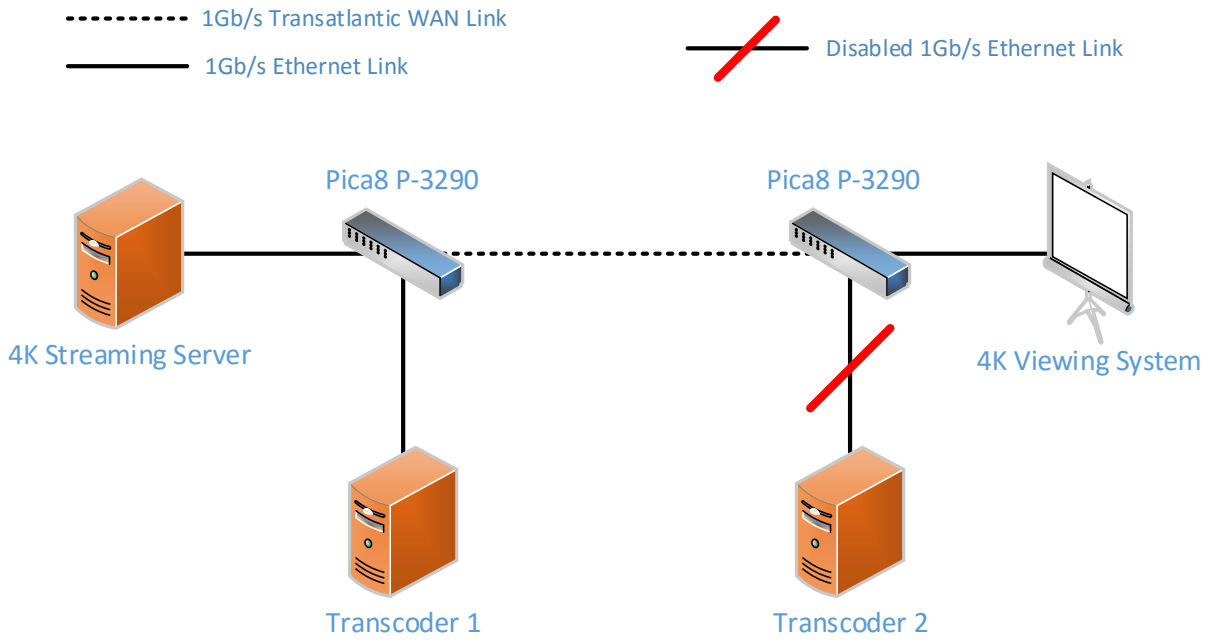


Figure 5.7: Initial state of transatlantic testbed architecture.

traffic along the WAN link while the initial transcoder continued transmitting. During the network emulated testing, it was possible to disable the checking of such aspects for the interconnecting bridge as direct access was available. However, due to the larger scale of this testing scenario, it was not possible to pinpoint any specific devices, or adapt any configuration on those that were known, as access was not available. It should be noted that packets travelling over the WAN link are encapsulated within VLAN frames, which should allow packets to be forwarded based on their VLAN tag value without inner packet header information causing problems. However, some devices can be configured to inspect the inner packet header details of a VLAN frame, which can cause the issues described. If these network devices forwarded using only the VLAN tag as intended, this issue would not be observed. It is not known whether this inner packet inspection is taking place along the link, however, it is more likely in this case that network devices located before the VLAN tunnelling has occurred are the cause of this issue. With improved control over both end-points of the WAN link, it is likely this problem could be avoided; however, the system is designed for use in a single operator OpenFlow enabled network, where the complete use of OpenFlow devices would remove the problem entirely.

Several attempted solutions were implemented to bypass the intermediary device issue stated above. The most promising solution seemed to be using OpenFlow to set the destination IP and MAC addresses of duplicated stream packets to unique values before they entered the transatlantic link. The original header information was then put back in place when the packets traversed the OpenFlow switch at the other end of the link. This solution did allow the duplicated stream to pass through the link, also confirming the hypothesis of why it was initially blocked; however, due to the added packet manipulation overheads, the video stream had many interruptions and appeared distorted at the client, making this solution unsuitable for the current scenario.

The solution that was found to allow the system to operate in its current state, was an increase in the second delay value during the migration process. Through a trial and error testing process, the optimal wait value was found to be 30 seconds, which significantly extended the previous wait time of 4 seconds shown in Figure 5.5. Increasing the wait time at this point in the migration process, allowed the server ARP table to be updated before the switchover occurred; this was an interesting observation, as it showed how the ARP requests were still able to traverse the transatlantic link, even when the repeated video stream could not. Although this does increase the total time taken to fully migrate the streams, this specific delay does not affect the quality of the switchover for the client. It should be noted that the described solution did introduce a small increase in the switchover delay when compared to the duplicate stream methods; however, as described in Section 5.6, this increase was negligible and did not affect the client viewing experience. With the maintained client switchover performance, this can be seen as an alternative strategy which does not require duplicated streams. This has the added benefit of conserving network capacity along the WAN link during migration, with only a minor reduction in performance.

5.6 Final Results

The reduction in the delay between packets arriving from the old transcoder and the new transcoder is the main metric that this system aims to improve, as this reduction improved the

viewing quality for the client during the migration process; therefore, the delay between these two streams was the focus for the collection and analysis of results. Other factors were also recorded for analysis, but were not essential for the migration to be deemed as a success; this is discussed further in Section 2.8.

The results presented for the emulated network scenario were collected over a span of 50 migrations for each migration type and were gathered by collecting packets using tshark [217] at the client; similar to this, the results from the transatlantic testbed were gathered in the same way over 25 migrations for both types of migration. Packet captures were analysed using an R script to determine the packet delay between the last packet from the original transcoder and the first new packet from the new transcoder. Since both transcoders utilise the same IP, the source MAC address was used to differentiate between the two transcoders. There were some instances of overlap between the two transcoder streams, where the packets had most likely been delayed at some point in the transmission process. However, these overlaps were rare and only consisted of about 10-20 packets, meaning VLC could handle them without issues during playback.

Table 5.1 provides the mean migration times for the various types of migration recorded. It can clearly be determined from these results that the OpenFlow aided migrations provide a significant improvement over standard stop and start migration techniques. Although the use of an ARP Flush on the server improves the performance of the non-OpenFlow migration, it still does not show sufficient improvement to match that of the OpenFlow aided migrations.

It is important to note, that the ARP flushed results are shown only to provide an insight into how the ARP timeout on the server is one of the largest contributing factors to the packet delay when using the non-OpenFlow aided migrations. This delay is caused by the server not detecting that the old transcoder is no longer active, meaning that the new transcoder with new MAC address takes a while to be detected and continue the stream. However, it would not be feasible to use an ARP flush like this in a real world system, as it would require access to the video server; whereas the system detailed here is being presented as a transparent system to the server and client, where apart from the transcoders physical migration, all the migration actions take place within the network, with no modification to the client or server.

Migration Type	**Mean Time**	**95% CI**	**Min**	**Max**
125ms OF Aided	0.0743	±0.0184	0.00001	0.2482
125ms Standard	18.2414	±2.0579	3.0056	37.7518
250ms OF Aided	0.0463	±0.0153	0.00001	0.1308
250ms Standard	20.0841	±5.3761	3.5516	42.9213
ARP Flush OF Aided	0.1231	±0.0201	0.000002	0.2928
ARP Flush Standard	4.2093	±0.1311	2.8771	7.1119
OF Aided	0.1058	±0.0191	0.00001	0.2580
Standard	16.2127	±2.3677	2.7545	37.3414
Transatlantic WAN OF Aided	0.1710	±0.0644	0.0340	0.5704
Transatlantic WAN Standard	24.3049	±4.7377	6.5502	51.9272

Table 5.1: The mean migration times in seconds for each scenario, including with and without the aid of OpenFlow.

Figure 5.8 shows the significant difference between the migration scenarios, with the OpenFlow aided migrations barely registering on the scale due to their sub-second delay times; these OpenFlow aided migrations can be more clearly seen in Figure 5.9. The non-OpenFlow migrations prevent a seamless switchover from occurring, with many reaching far beyond what would be considered acceptable interruption. It is clear from the presented results that the use of OpenFlow makes the idea of migration feasible, when before it would not have been attempted.

Additionally, from Table 5.1 and Figures 5.8 and 5.9, it can be observed that link delay does not adversely affect the performance of the system. However, as stated in Section 5.5, VLC required its network cache to be increased to 400ms during the 250ms scenarios in order to enable a seamless switchover; although, as previously discussed, this does not seem to be attributed to the migration process itself, as some artefacts appeared during normal playback when the network cache was set to 200ms. From looking more closely at the packet captures during the tests, variation in receiving packets seemed to be a cause of this instability; this variation in packet delay is believed to most likely be caused by the network emulation software introducing the latency, which may not always be consistent.

It is important to highlight that the system worked successfully over the transatlantic link, even with the complications discussed in Section 5.5.4. Although this system provided a slight increase in the switchover delay, it still achieved switchover on average within 200ms; this still allowed a seamless switchover to occur when viewing the content using VLC player with its 200ms network cache. Furthermore, as described in Section 5.5.4, this can be seen as an alternate strategy for the migration system. This is important to consider, as it simplifies the migration process and conserves link capacity with only a small increase in switchover delay.

Packet captures were further examined to gain an insight into the effects of migration on the packet stream as well as packet delay. Although a large number of the packet captures were analysed, only a subset of these are shown in Figures 5.10,5.11,5.12 and 5.13; this was to provide clarity in the results, as other packet captures for different latency values produced results similar to those shown, with no significant differences.

Figure 5.10 presents an insight into how packet delay is affected during the migrations, as it shows the inter-packet delay during the video transmission. It should be noted that although some results are displayed with a 0ms value, this does not show a situation with no delay between packets, it represents when no packets are being received. As can be seen from the results, the OpenFlow migrations present minimal disruption to the stream; this is in contrast to the non-OpenFlow migrations which significantly disrupt the stream, with extended periods where no packets are received.

Figure 5.11 presents a sample view of the packet arrival times at the client during a number of LAN based migrations, including 2 OpenFlow and 2 non-OpenFlow migrations. It can be seen that there is a significant improvement when OpenFlow is utilised to aid the migration process, as there is minimal disruption to the packet arrival times. In contrast, the standard migration produces significant disruption to the packet stream, with a large delay before the stream is re-established after the switchover. Similar results can be seen in Figure 5.12, which presents a closer and more detailed view of packet delivery during a LAN based migration.

Figure 5.13 presents the packet arrival results captured during the transatlantic WAN testing

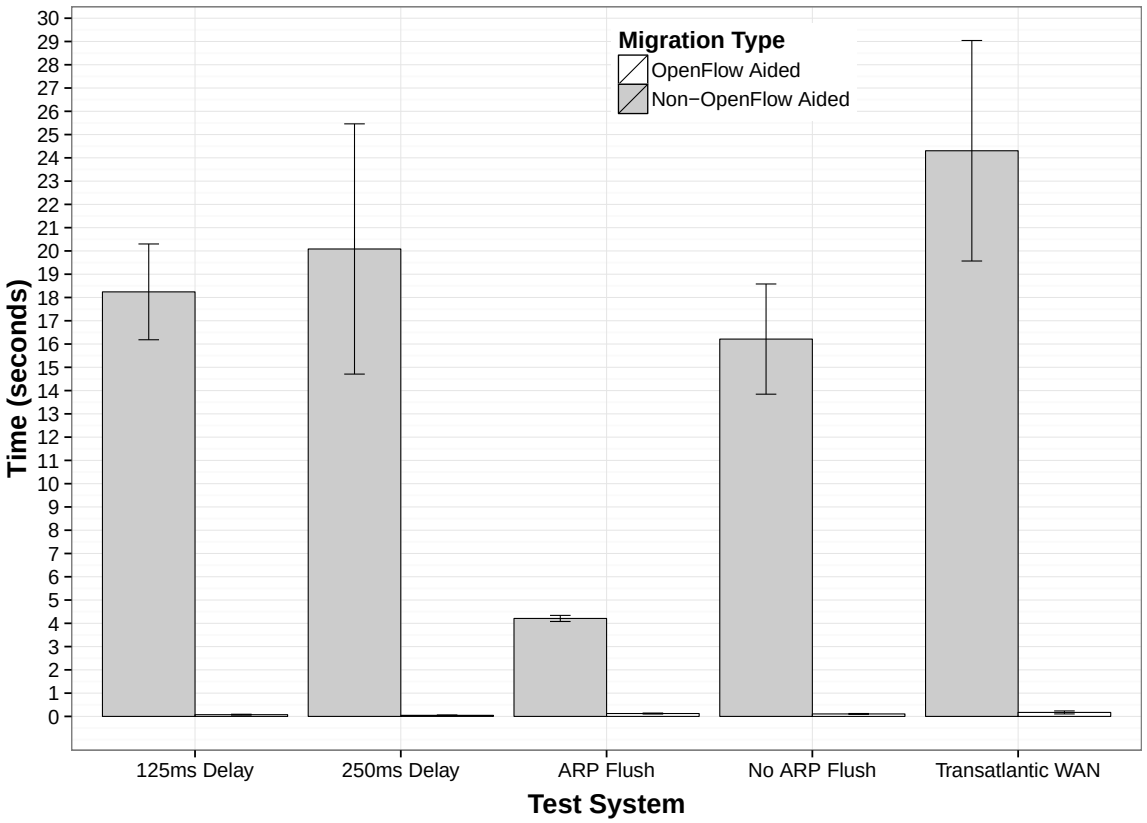


Figure 5.8: Packet delay in content stream caused by the migration process.

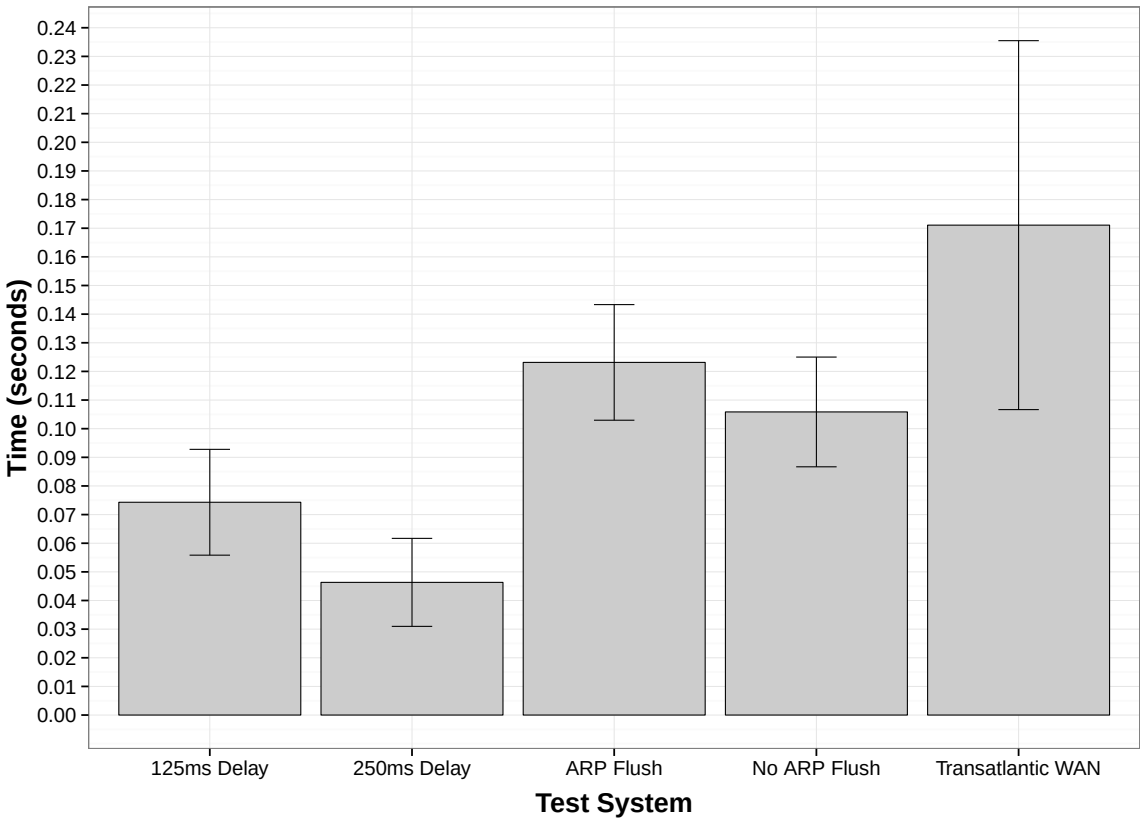


Figure 5.9: Packet delay in content stream caused by the OpenFlow migration process.

scenario. It displays the same characteristics as the LAN based tests shown in Figures 5.11 and 5.12, with the OpenFlow aided migrations not affecting the packet arrival rate and the standard migration showing significant interruption in the stream.

Further to the presented quantitative results, it is important at this stage to discuss the qualitative results that were observed during the streaming and migration operations; this is important due to one of the main goals of this system being a seamless migration for the viewer. A number of migrations were observed and a number of factors were assessed to gauge the performance of the switchover in terms of viewing quality. During non-OpenFlow migrations it became clear that a seamless migration would not be possible, with interruptions in the video spanning on average 10-20 seconds; this was perceived as not only a large pause in the video, but it also produced significant artefacts when the stream was re-established. In contrast, OpenFlow aided migrations showed only a brief stutter in the video on occasions, although during a number of migrations this was barely perceptible; this brief stutter in the video on occasion produced some

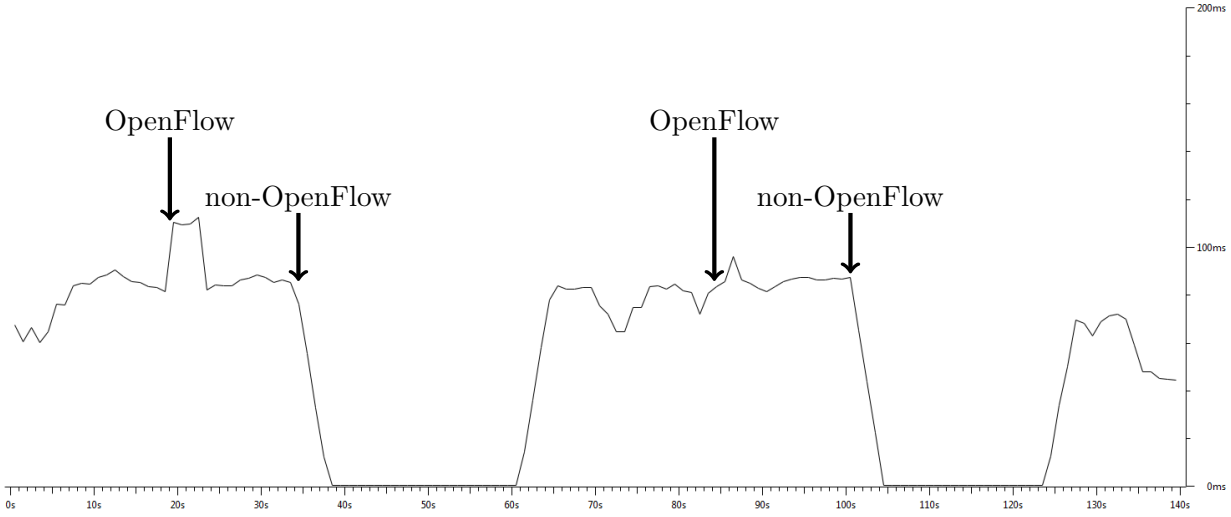


Figure 5.10: Time delta between packets received at the client, for LAN based scenario.

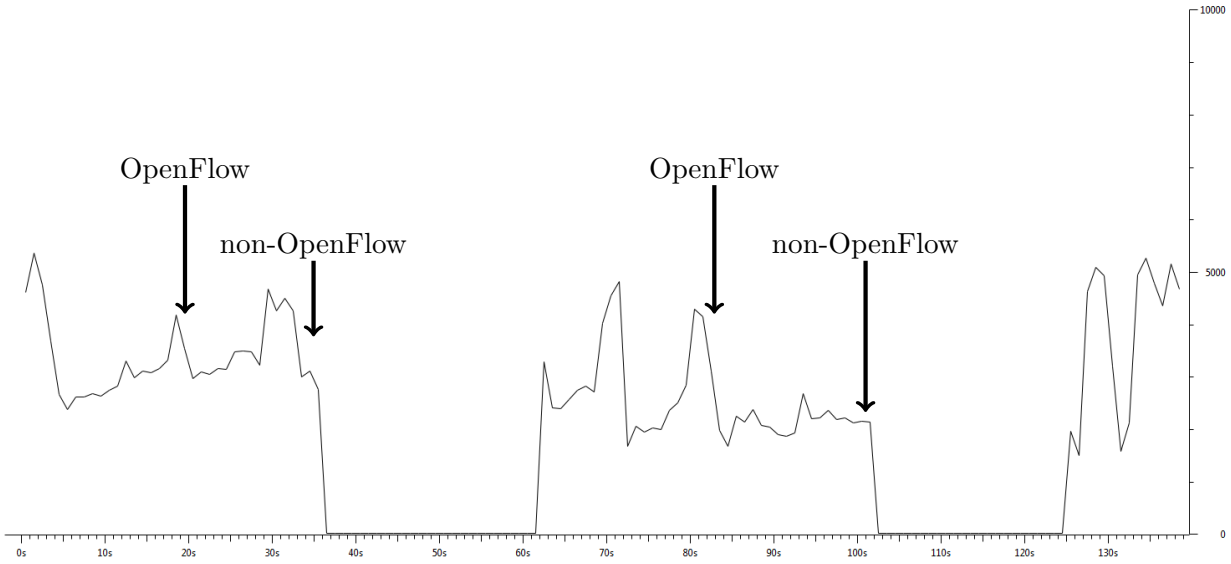


Figure 5.11: Packet arrival rate at the client, for LAN based scenario.

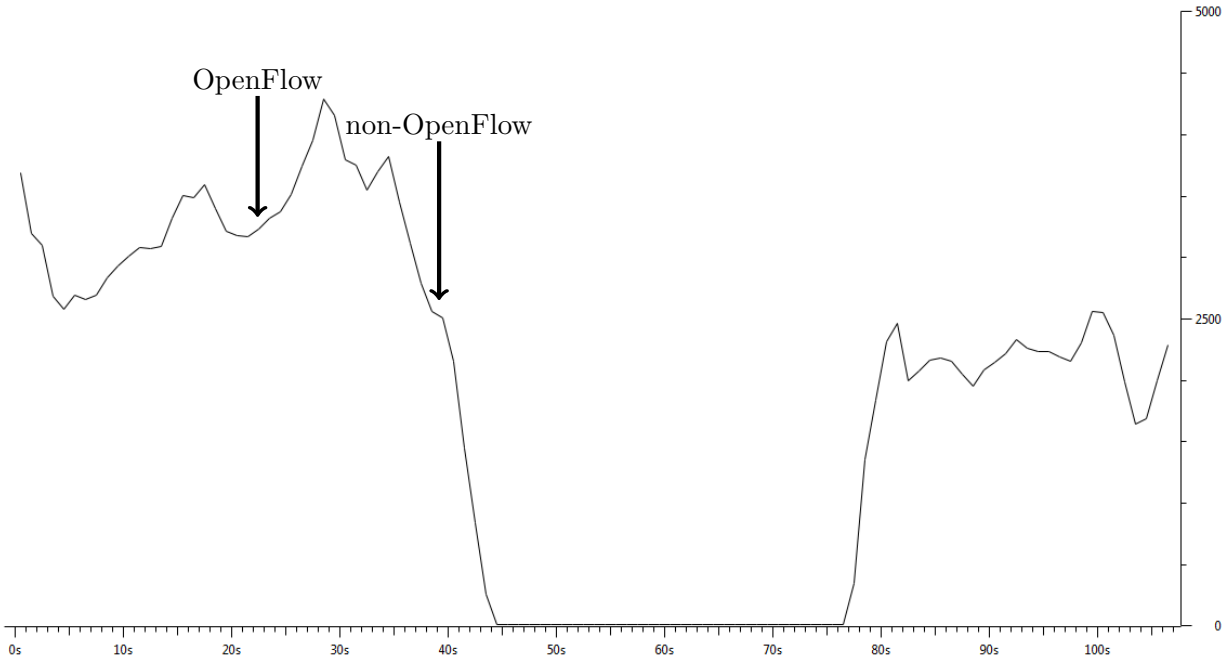


Figure 5.12: Detailed packet arrival rate at the client, for LAN based scenario.

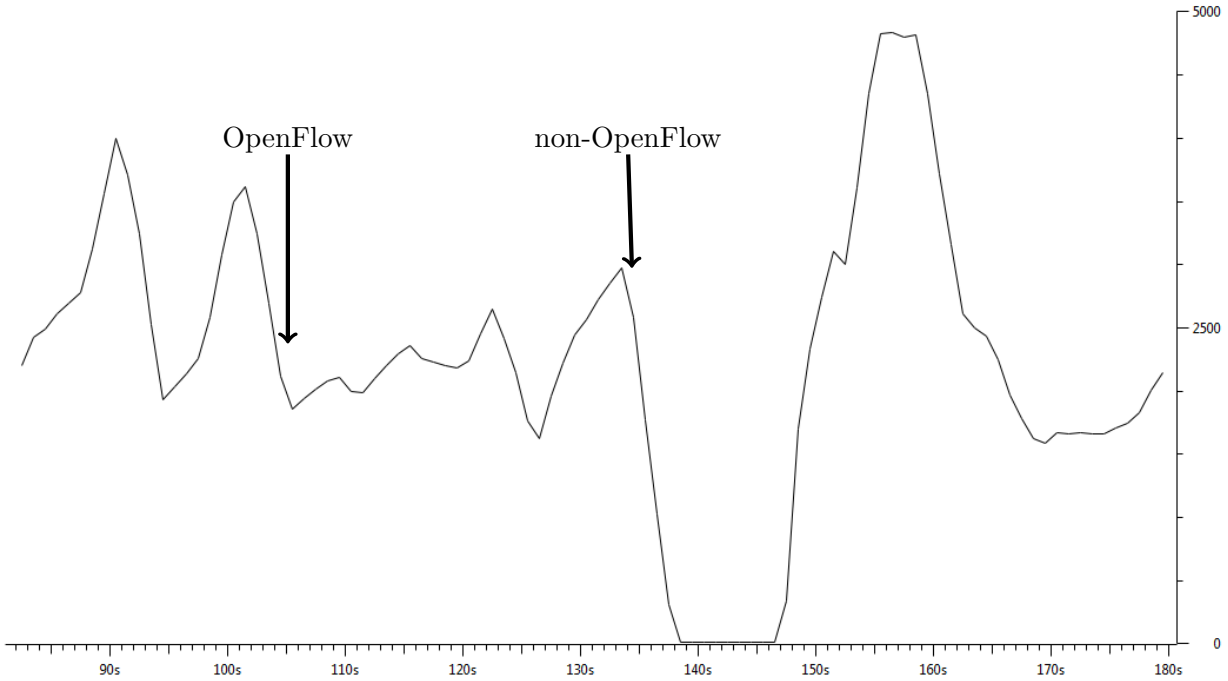


Figure 5.13: Detailed packet arrival rate at the client, for transatlantic based scenario.

artefacts, however, this was minimal and was significantly less than those found in non-OpenFlow migrations.

5.7 Summary

It has been shown in this chapter how the use of OpenFlow can aid the migration process of transcoders when they are streaming live UHD content. Using this type of dynamic transcoding resources is one possible solution to managing the increase in real-time content delivery; however, it is important to also investigate other delivery mechanisms that can improve content delivery further. Further to this, an OpenFlow content delivery network is presented in the next chapter, designed to work in parallel with the stream migration system, as well as providing other improved mechanisms for real-time content delivery.

Chapter 6

OpenFlow Smart CDN Architecture for Video Delivery

CDN architectures are used to distribute multimedia content to large groups of clients, with current implementations mainly focusing on stored content. However, there are systems that extend to providing real-time content with P2P, CDN and ALM approaches [166,193,199], but most do not utilise the increased flexibility and efficiency that SDN can provide. OpenFlow enables networks to separate themselves from normal routing and switching mechanisms, allowing them to operate in a way that was previously not possible. This enables research into network control beyond that which standard IP and MAC routing policies allow. The OpenFlow CDN architecture presented in this chapter utilises this concept of bypassing standard networking policies, in order to improve the delivery of both stored and live video content to large numbers of clients.

Using the capabilities of the OpenFlow platform, it is possible to design a system that replaces several aspects of a standard CDN architecture. This new OpenFlow based CDN will provide improved resource utilisation, by harnessing OpenFlow's ability to quickly modify network behaviour; this can be used to both allow optimisation of network traffic, as well as routing content to and from the ideal servers. The proposed system is designed to focus on providing video content to large numbers of clients, with both stored content and real-time streaming scenarios being considered; however, the system could also be utilised for other content if slight modifications were made. The proposed system will also focus on a new user scenario, which differs slightly from those used in previous chapters; however, the system is designed to be complementary to the earlier scenarios, with the assumption that they would work in parallel to provide the best service to a number of different applications. This chapter will present the architecture for the proposed system, while Chapter 7 presents a novel max-flow algorithm that aims to optimise traffic placement within the proposed system, with the goal of reducing the probability of blocking.

It is important to reiterate that the architecture presented in this chapter is proposed to complement and operate concurrently with the already presented migration system in Chapter 5, it is not designed to supersede it. Additionally, it needs to be highlighted that the systems presented have not been fully implemented and tested together, so further work is still required to enable a fully functioning system; however, initial research through testing and modelling has shown each part of the proposed system to be both feasible and working as described.

6.1 Application Scenarios

As discussed in previous chapters, the recent development of streaming applications being presented as social media platforms has had a dramatic effect on the way people both produce and consume video content; these types of applications provide an ideal use case for the proposed system. Before the introduction of these live streaming applications, the most efficient and accessible way to share your personal content such as your gaming sessions, was to record the session and then upload the video to an online video host such as YouTube [170]. YouTube would then handle the replication and transcoding tasks for the users content, so that it could be viewed by large numbers of clients at a later time on multiple platforms; this would be achieved by storing multiple copies of the video in multiple formats throughout a typical CDN. As stated in Section 5.2, this method has recently been overshadowed by services such as Twitch [168], which provide people with the ability to stream their gaming session live straight from their console. Other user broadcasting applications such as Meerkat [172], Periscope [171], Kamcord [173] and Mirrativ [174], have also seen a recent rise in popularity [175], allowing users to stream live to a large audience from their mobile device. These applications are very closely integrated with social networks such as Twitter and Facebook, allowing users to reach large audiences with their broadcasts. These applications differ slightly from those such as Twitch, as they utilise a more restrictive device in terms of both processing power and network connectivity; these restrictions present a large number of issues that need to be addressed, in order to allow a reliable and consistent stream. Current methods of achieving this involve using HLS or MPEG-DASH for distributing the stream, however, this inherently adds a large 10-20 second delay to the live stream. Further description of these issues and other possible use cases can be found in Sections 2.8 and 5.2, while description of HLS and MPEG-DASH can be found in Section 2.1.3.1.

A user being able to broadcast to a large audience using their relatively low powered device through a low capacity link, is a significant advancement from previous streaming systems. This shift in live broadcasting from the usual large corporations is important to discuss, as live streaming previously utilised large scale broadcasting equipment to distribute content to large groups of users; this new type of live streaming presents an important influence on the development of current live streaming technology. If consumer based live streaming adoption continues to grow, new systems will need to be developed to handle the increased demand. It is unknown how, or if, this new type of broadcasting will influence current broadcasting services, but its rapid large scale adoption seems to indicate that it is most likely going to continue to gain further interest from users.

One of the aims of the system presented in this chapter is to address the lack of real-time streaming architectures, specifically designed to distribute consumer produced live content; furthermore, the proposed system will be able to achieve this without having introducing the 10-20 seconds of delay that current HLS based systems currently require. Details of the real-time streaming aspect of the proposed system are presented in Section 6.4.

6.2 OpenFlow Smart CDN Architecture

This section details the proposed CDN architecture, with a focus on its utilisation of the OpenFlow platform to achieve improved flexibility and performance. The proposed CDN architecture

is designed primarily for distributing multimedia content, including UHD video; however, it is feasible that the system can also be utilised for providing other content types. It should be noted that the term "node" is used in this section to identify an OpenFlow enabled networking device, located within the CDN system; these network nodes are to be assumed as the attachment points for the other devices located within the CDN, such as the streaming servers and transcoders.

The flexible control that is gained using an SDN technology such as OpenFlow, allows systems such as the stream migration mechanism in Chapter 5 to be applied within the network. Furthermore, by using OpenFlow, the traffic can be routed intelligently through the CDN infrastructure towards exit nodes that bridge the privately addressed CDN with the public Internet. This added control over traffic in the network, provides the flexibility required to control client requests for content; this enables the redirection of streams in the network to avoid congestion, as well as providing the ability to redirect client requests to alternate servers to aid load balancing operations.

The centralised control aspect of OpenFlow is one of the many advantages over existing CDN architectures; the OpenFlow controller would be at the core of the presented CDN system, as it would be tasked with managing the entire network aspect of the system. This includes the more advanced tasks such as redirecting streams away from congested servers. The ability for the controller to redirect content away from congested servers would require various statistics and device information to be passed to the controller, many of which relating to the network would already be available using the statistics reporting built into the OpenFlow system. However, additional statistics such as server load would need to be gathered using an alternate means, such as with a statistics gathering tools like the previously described sFlow. The combination of network statistics and server resource utilisation would provide an extensive knowledge of the system, enabling efficient optimisation throughout the systems operation; these optimisations include traffic placement optimisation, server utilisation optimisation and stream replication optimisation, which will be detailed in Section 6.4. Although it is assumed this optimisation functionality would be built into a controller such as Floodlight using one or more modules, it is also feasible that these optimisation and redirection tasks could be performed by other individual applications; these could perform the required optimisation calculations and then pass the required control information to the controller, using an Application Program Interface (API) interface such as the static flow pusher API found in Floodlight. Regardless of where the specific functionality would reside, it is important to note that the system would still rely on the OpenFlow controller to configure the network accordingly.

With any centralised system it is important to discuss scalability concerns, as this can be an inherent scalability weaknesses usually found in such systems; however, as shown extensively in Section 2.6.3, OpenFlow controllers are able to scale sufficiently well and there are also several systems developed to extend this scalability further with the use of a distributed design. Therefore, scalability issues relating to the OpenFlow controller will not be considered further; however, scalability relating to other aspects of the system will be detailed where relevant.

Another benefit of using OpenFlow within the system is the fine granular control of traffic in the network; this precise control is gained through the flexibility of flow rules that can be configured within the network devices. Using this precise control over traffic it is also possible to

provide improved security within the system, allowing parts of the network to be masked from view; this can hide internal network structure and server resources from both internal and external hosts, which may be using network probing to try and ascertain the architecture for malicious purposes. The improved security aspect of this system could also be extended further to include enhanced authentication for users, with the ability to mitigate unauthorised users at the edge of the network before entering the internal infrastructure; in current systems the unauthorised traffic would still be able to traverse the internal CDN infrastructure, before being rejected once it reaches its destination server. This security aspect is discussed further in Appendix D.1, as although it is relevant, it goes beyond the main focus of developing a system to deliver video content.

The proposed OpenFlow CDN can be partitioned into two specific scenarios, each with different requirements that need to be optimised in order to achieve improved performance. The first of these scenarios investigates the use of the OpenFlow CDN for providing real-time streaming content, this is detailed in Section 6.4. The second scenario focuses on providing stored video content, similar to that of many existing CDN systems; this scenario is detailed in Section 6.5.

It should be noted that all aspects of the proposed system presented in this chapter operate using network traffic manipulation, as it has the added benefit that no software modification is required on either the servers or client machines; this is an important factor, as it allows the system to function transparently without issues with software or machine compatibility.

The overview of the system described here can be seen in Figure 6.1, which shows a simplified view of the architecture design. The displayed WAN links are assumed to be dedicated light paths between data centre sites, however, the system would also be able to operate with tunnelled connections through the Internet with an expected loss in performance. With the partial mesh design, it would be usual to question the connectivity of L2 switching based on a large number of loops within the network; however, it is important to remember that OpenFlow controllers such as Floodlight, handle the task of topology management as well as using L3 routing for suitable traffic. For this reason, loop-free network mechanisms such as Spanning Tree Protocol (STP) are not required.

The aspects of the proposed system that are presented later in this chapter have been tested under the assumption of a UDP based streaming system, due to it connectionless design. This allows easy redirection and migration of traffic, without having to handle TCP based session and connection issues. Certain aspects of the system would not be affected by these types of issues, but in order for all the stated features to operate together, it is important to utilise UDP based transmissions. This is considered a reasonable assumption, given a lot of real-time video protocols such as RTP utilise UDP as the basis of their design, due to the need for fast transmission instead of reliability. Further description of the differences between video transmission protocols can be found in Section 5.4.2.

6.3 OpenFlow based NAT

As mentioned in the previous section, it is possible to utilise the flexibility that OpenFlow provides to improve several aspects of a CDN system. This section presents an improvement to the current NAT systems utilised in some CDN architectures [218, 219], but can also be used outside

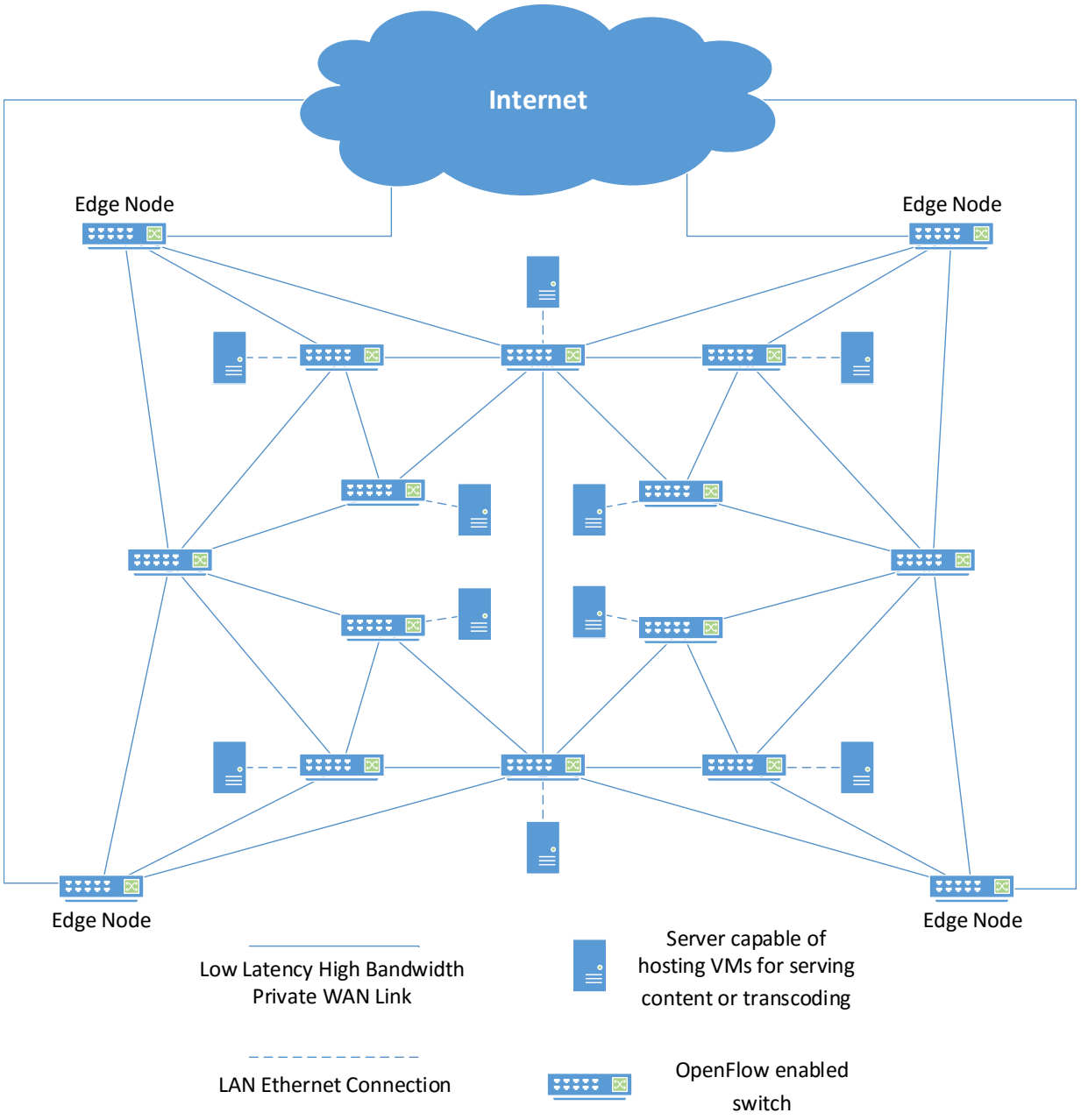


Figure 6.1: Overview of the proposed OpenFlow CDN architecture.

the context of a CDN system as well. Yeh and Chiu [218] present an interesting approach for distributing NAT operation between a number of servers, for future work it would be interesting to compare this with the architecture presented in this section to see how well they compare in terms of performance.

Within the proposed OpenFlow CDN system, a private IP addressing scheme is used to allow greater control over traffic engineering and system architecture. For the privately addressed CDN devices to interact and provide content to clients in the public Internet, the private addresses need to be converted to those within a public address range; this mechanism is termed Network Address Translation (NAT), with current systems mostly using the port number and source IP address to identify connections [220–222]. Within the proposed system, this task is performed at the exit/entry nodes positioned on the edges of the CDN architecture; these nodes use OpenFlow's

packet header manipulation to modify the source and destination IP addresses as they transition between the public and private networks. From this point, the exit/entry nodes will be referred to solely as edge nodes and are considered to be any node which maintains connections to both the internal network and the Internet. In order for these edge nodes to bridge the two networks and perform the translation, each edge node is assigned both a public IP address accessible to Internet based clients, as well as a private IP address accessible from within the CDN architecture. With this system in place, current active network streams can then be tracked and managed inside the CDN by keeping a record of requesting client's source port, as well as the specific edge nodes internal IP address that the request arrived on. Using these identifiers, the system is able to determine the correct packet header information to write to the packet as it exits the CDN onto the Internet.

It should be noted that the system presented here is not designed to be an improvement of the system presented by Liu and Jin [75], detailed in Section 2.6.6, as its design was not in any way influenced by this. The presented system was designed from its initial foundations specifically for the purpose of use within the OpenFlow CDN proposed in this chapter, with the goal to improve the delivery of content to clients.

Aside from the standard NAT services described, the edge nodes will also be required to handle ARP requests, so that correct MAC addresses can be obtained; this can either be performed using a similar approach to the translation system proposed below, or using a separate implementation on the nodes to reply to the ARP request and then convert the MAC address accordingly when the packets traverse the node. This will not be discussed further, as ARP relay and proxy systems are already well researched. However, it is realised that further work into selecting the most efficient mechanism for this specific scenario is still required, in order to ensure the best performance within the system.

The system is proposed to operate as follows:

- A client sends a request to the IP address gained from a DNS query for a specific content's URL.
- The request packet reaches the edge node and is forwarded to the OpenFlow controller, the controller then performs several actions depending on the content being requested. This may involve Deep Packet Inspection (DPI) for content identification, depending on the application being requested.
    - ◇ The controller inspects the packet to determine what content is being requested.
    - ◇ It then chooses the correct server for the content, based either on content availability in the scenario of real-time video content, or the least utilised server when providing stored content replicated across multiple servers.
    - ◇ It then stores a record of this connection in a connection database, including the client's source port, the edge node that received the request and also the selected server. This database can then be used to help with packet translation operations, but additionally, it can also improve optimisation mechanisms since it will have a record of all current traffic in the network.

- Flows are then inserted into the edge node providing several instructions:
  - ◇ For packets arriving from the Internet from the client:
    - ○ Replace the IP address with the private IP address of the chosen content server. **This is one of the key novel components that allows access to the CDN, while redirecting the client to the selected server.**
    - ○ Replace the source PORT with a mapped private PORT, which could be the same if no conflict is detected.
    - ○ Replace the source IP address with the private IP address of the current edge node.
  - ◇ For packets arriving from the internal network from the content server to the mapped private PORT of the client:
    - ○ Replace the source IP address with the public IP address of the edge node.
    - ○ Replace the destination PORT with the source PORT of the client.
    - ○ Replace the destination IP address with the client IP address.

This is a simplified operation scenario and some smaller stages in the process are omitted for clarity.

Using the described NAT process, the system would be able to keep track of traffic in and out of the network, as well as performing optimisations both with server selection and traffic assignment. It should be noted that DPI is not always required for content identification, as packet header fields such as destination port number could be utilised for this; but it would depend on the current application of the system, as DPI may be beneficial for other scenarios that require more detailed request information. Furthermore, although DPI may at first seem like a bottleneck to the system, it is important to remember that only the first initial request from the client would be inspected in order to configure the flow rules; after this initial request, no DPI mechanism would be required for that stream. With regards to any scalability concerns relating to this part of the system, it is useful to look at the current Internet Service Provider (ISP) network caching performed for streaming services, such as Netflix [223, 224] or YouTube [170, 225]. These provide valuable insights into real world systems that track large numbers of video stream requests, in order to redirect them to local caches; this shows that the proposed system is feasible if implemented correctly.

The described NAT process is shown more clearly in Figure 6.2, which provides a visual representation on how the OpenFlow NAT system processes a packet entering the network; for clarity, certain packet header fields have been omitted or shortened in the diagram. Although the reverse of the NAT process is not shown, it can be assumed as just the reverse of the incoming packet.

Depending on the systems being implemented outside of the CDN, the functionality of the proposed CDN could be further extended in relation to the bridging between networks. Further to this, with the availability of a number of edge nodes connecting to the Internet in various locations, it is desirable for content to exit the CDN at the edge node closest to the requesting client. This can reduce latency and also reduce the risk of congestion issues that could be encountered within the Internet, as it minimises the distance travelled within the public Internet's shared infrastructure. In current systems this can be achieved by routing the client to the closest edge node using

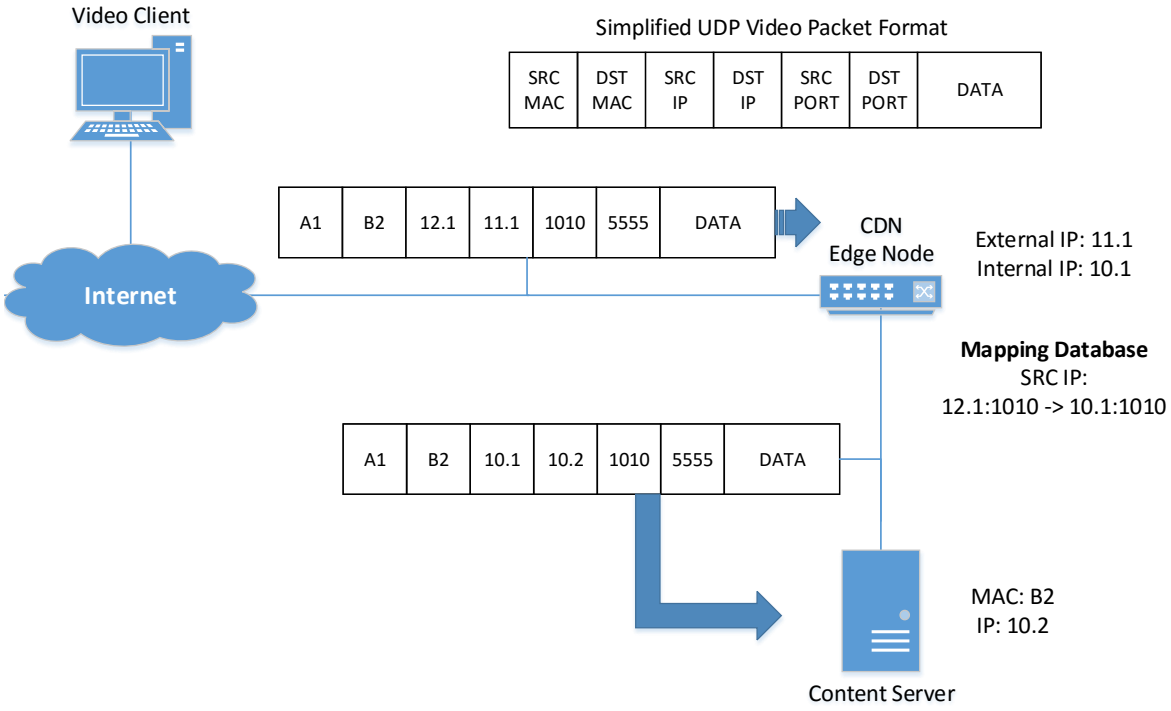


Figure 6.2: Overview of the OpenFlow NAT system handling an incoming UDP packet.

DNS redirection techniques, however, with the proposed system this can be both extended and improved. Although the NAT system presented above currently assumes the content is to exit the CDN using the same edge node that the request entered, this does not have to be the case. Using OpenFlow redirection techniques, it is possible to manipulate the traffic to exit the CDN from a different edge node than the one it entered on. This can mean that if a client request arrives at an edge node that is not their closest node, this can be changed without interaction from the client. Additionally, current DNS redirection systems are not able to change their configuration quickly on demand, whereas the proposed system would be able to rapidly modify the exit node being used in order to bypass any current congestion issues; this, of course, will not aid latency in any way, but maintaining a stream with increased latency is preferred in many scenarios over dropping the stream because of congestion.

For the system to function with an exit node different from the entry node, the controller would need to configure flow rules that differ from those presented on Page 105. The controller would need to split the flow rules between the entry and exit edge nodes as follows:

- For packets arriving from the Internet from the client on the entry node:
  - ◇ Replace the IP address with the private IP address of the chosen content server.
  - ◇ Replace the source PORT with a mapped private PORT, which could be the same if no conflict is detected.
  - ◇ Replace the source IP address with the private IP address of the newly selected edge node.
- For packets arriving from the internal network to the new exit node, from the content server to the mapped private PORT of the client:

- ◇ Replace the source IP address with the public IP address of the original edge node.
- ◇ Replace the destination PORT with the source PORT of the client.
- ◇ Replace the destination IP address with the client IP address.

Furthermore, an additional flow rule would be configured on the switch acting as the attachment point for the content server, whereby it would replace the destination MAC address of any relevant packet for the current stream with the MAC address of the new exit node. This would then allow the packet to be forwarded as required to the new exit node without other modifications to normal L2 switching functionality. It should be noted that although certain headers are configured with the switch IP address, this is a virtual IP address acting only as an identification for the switch for routing and translation purposes. It is interesting to note that this form of NAT would be difficult to achieve using standard router enabled NAT.

Scalability is always a concern when dealing with any large scale content delivery system, so it is important to discuss apparent issues with each part of the system. This section relies on the use of the OpenFlow controller, so it is again important to highlight the research discussed in Section 2.6.3 detailing its ability to scale well. Furthermore, it should be reminded that the controller is only required for the initial request from the client, after this point the flow rules are inserted into the relevant switches to offload all further management for that stream; this significantly improves the scalability of the system. Further investigation into the most efficient architecture may necessitate the use of multiple controllers in a hierarchical architecture, rather than a single high performance controller; however, the concept of the presented system will not need to be modified as long as they are configured correctly.

Another important aspect related to scalability, is the identifier length which will limit the number of streams that are able to be tracked and managed. The identifier will be reusable given a certain timeout value, which will need to be selected based on the specific application that the CDN is providing. It has been decided in the proposed system to utilise one of the most widely used identifiers within current NAT systems, this identifier is the application port numbers. These provide a 16 bit address field to track traffic streams, giving a theoretical 65,536 identifiers that can be used. Although this theoretical total number is available, most systems choose to restrict the port numbers to the range that they were originally from. Available ranges include 0-1023 for well-known ports, 1024-49151 for registered ports and 49152-65535 for dynamic/private usage. This can restrict the scalability of the system, but it is performed for compatibility reasons when dealing with internal requests. However, these compatibility issues can be overcome by modifying ports in the event of a conflict [220,221]. The port numbers can also be used alongside an incoming IP address when used in a situation where the NAT gateway has multiple public IP addresses. In the case of the presented system, the entry nodes internal IP address is used in conjunction with the application port numbers as an identifier within the network. The external public IP address could also have been used in this case, but was considered less flexible than the easily modified virtual internal IP address. However, it is important to note that if the edge nodes were configured with multiple public IP addresses, this could then be used to extend the number of internal identifiers by using one of the public IP addresses of the edge nodes.

6.4 Real-time Video Delivery using OpenFlow CDN

One of the intended scenarios for the proposed OpenFlow based CDN is a real-time video streaming architecture; with this scenario, OpenFlow is used within the network to route traffic from an initial video source server to the relevant edge node. The initial source server is selected when the first request from the recording client is received, it can then be selected based on either latency requirements or current server utilisation conditions. The initial server can then receive the real-time stream direct from the recording video source, where it can distribute the stream to a large number of clients through the proposed architecture.

When a client on the public Internet requests an active real-time video stream, the request would be routed using normal IP routing to the closest edge (entry) node for the CDN. Redirection to the closest edge node would usually be achieved using standard DNS redirection practices, which provide the client with the appropriate IP address based on the requested content URL. However, this system could also be augmented if the client was also located on a private OpenFlow enabled network; this will be discussed further in Section 6.6. The client request would then be directed to the correct server within the CDN, based on the requested content. DPI techniques may be required at this stage for content identification; however, the use of DPI will depend heavily on the current application being served. For example, when using the real-time video streaming scenario, DPI would not be required, as packet header fields such as destination port number could be used as content identifiers. Once the packet traverses the CDN edge node, it would be converted to the private addressing system using the NAT system presented in Section 6.3.

The server providing the stream is then able to begin transmission of the stream to the requesting client. If no conversion of the content is required, the content stream can be sent directly to the respective edge node. If content adaptation is required, the server would then send the content to be processed at a transcoder within the CDN architecture, which would then forward the stream to the required edge node. With the use of a transcoder within the OpenFlow network, the migration processes presented in Chapters 4 and 5 would allow stream migration to optimise transcoder placement and utilisation.

One of the most important benefits of the presented system in relation to delivering real-time video content, is the ability to duplicate streams on the network devices themselves. Duplicating streams at the switches, coupled with the ability to manage traffic from the centralised controller, enables multicasting at the switch level within the proposed architecture. Utilising switch level multicasting would not only reduce the overall network load within the architecture, but also reduce the load on the content servers and transcoders. The reduction in load can be attributed to the fact that if multiple clients request identical content, the network can replicate a single stream from the server and/or transcoder and disseminate it to the respective clients. To achieve the stream replication process, the packet header manipulation features of OpenFlow would need to be utilised. Specifically, the replication process would involve modifying the destination IP address and port numbers of packets when traversing a specified switch; these IP addresses and port numbers would need to be configured to match each request from a specific user, in order for the duplicated streams to successfully reach their respective clients. The switch level multicast functionality would rely heavily on the centralised nature of OpenFlow, as it requires the knowledge of each client request entering the network. Although client request information would already

available from the NAT system proposed in Section 6.3, the content servers and transcoders would also need to contribute to the centralised flow database of the controller. The information that would be required from the servers and transcoders is the specific stream characteristics, such as codec, resolution, bitrate and some form of content identifier. These stream characteristics ensure content is only duplicated when all characteristics of a stream are considered identical, which would make it suitable for switch level multicasting. It is important to note that the proposed multicasting system would operate most efficiently if it was performed at the edge nodes of the network. This would minimise the traffic flowing within the CDN infrastructure, ensuring only unique streams are routed to each edge node. Moving the multicasting process to the edge nodes would also simplify traffic management, as there would be less content streams to route and manage within the network. However, due to current limitations on packet header manipulation performance within switches, it may be necessary to perform the header manipulation on multiple switches throughout the network, in order to distribute the overheads involved. The overhead issue described would need to be investigated further, in order to determine the most efficient configuration, as well as any upper limits of the system; this is one area of future work.

A further benefit of the proposed video delivery architecture being located on a privately connected network, is the reduction in latency that the system can provide for delivering video to clients. The reduction in latency is a result of minimising the content streams exposure to the unpredictable and unreliable public Internet's shared infrastructure. In order to achieve this, content is routed through the CDNs private low latency links to the edge node closest to the requesting client, reducing the number of hops the stream traverses within the Internet. By reducing the streams exposure to the public Internet, reliability is also improved, as it can bypass any issues with the Internet's shared infrastructure; some examples of issues that can occur within the Internet infrastructure include: routing loops, link failures, congested links and dropped packets. It is important to note that the presented system also maintains a partial mesh network, with redundant links available for use in the event of link failure or congestion. The redundancy within the presented partial mesh network is possible due to the OpenFlow controller monitoring the network architecture for link issues and congestion, which can then be mitigated using flow rules to bypass any affected links.

The proposed live streaming system would present an ideal infrastructure for the consumer live streaming applications discussed earlier, as it would provide the live stream to a large user base without the current 10-20 second delay found with HLS and MPEG-DASH based systems. This would be beneficial for applications such as Twitch, Meerkat and Periscope (see Sections 2.8 and 5.2 for more details on these), where live user interaction aims to bring a social aspect to the broadcast. In order for these types of applications to utilise the proposed CDN system, the user broadcasting their content would send a single live stream to a selected content server within the CDN infrastructure through their closest edge node. The selected server would then act as the initial streaming server within the CDN, handling the delivery of the content using the mechanisms described above. It should be noted that the proposed system would still add a certain amount of delay to the transmission, since the stream would still need to be received, processed and retransmitted to clients; however, with the capability to transcode content in real-time within the system, the delay would be significantly less than that of HLS based systems. It is estimated the

delay associated with the proposed system would be in the order of around 1-2 seconds based on initial testing, which would be more than adequate for the application scenarios mentioned earlier. For example, game session streaming through services such as Twitch, involve interaction with viewers through text chat and other text based social media; since this interaction is performed through an indirect communication, it can accommodate a second or two delay with the video stream without adversely affecting the viewer experience. However, scenarios that involve live broadcasts that require instant feedback, such as a remote surgical team operating using robotic operators, or offering consultation [226, 227], might not be suitable for use with the proposed system; this is due to the requirement of a latency far lower than other types of broadcasts. The remote surgical team scenario would most likely benefit from the standalone transcoder system presented in Chapter 5, as this would provide much lower latency and is designed with performance as one of its main objectives. Further investigation would need to be performed to gain an insight into acceptable levels of latency for different types of broadcasts, as well as the levels of latency possible using the proposed system on a large scale.

6.5 Stored content OpenFlow CDN

A further application of the proposed OpenFlow CDN system would be for delivering stored video content to a large volume of geographically diverse clients; this would closely mimic the functionality of standard CDN systems. As with a standard CDN, the stored content is replicated on multiple servers, so that content requests can be distributed across the infrastructure in order to provide an increased number of users. CDN systems that operate in this way are able to scale far beyond what is possible using a single server. In addition to this, CDNs can also be used to improve latency when serving content, due to the possibility of an available server in close proximity to the client. However, in the case of the proposed system, this would be influenced more by the proximity of the client to an edge node of the CDN and not the internal servers.

This architecture would be similar to that of the previous architecture discussed in Section 6.4, but would add the additional optimisation of server selection for each incoming client request. In contrast, the real-time system only has to optimally select the initial server for the real-time recorded stream, not for every client, as they are all effectively served from the same initial server. This server selection optimisation would be dependant on the requirements of the content being hosted, as it could be configured to prioritise certain characteristics such as latency or server utilisation.

The proposed system would allow for a high degree of optimisation, as it would be able to adapt more rapidly to client demand than existing systems; this would be due mainly to OpenFlow being able to redirect streams on demand, according to the in-depth knowledge it possesses through its centralised control. In contrast to the real-time scenario, it would not be required to send content for processing at real-time transcoders during transmission, as the content would be stored in all the required formats. However, multicasting at the OpenFlow switches as described in Section 6.4, would still be beneficial in some cases to reduce traffic and server utilisation within the infrastructure. It should be highlighted though, that switch level multicasting would not be as dominant in the stored content scenario, as requests for content at various intervals would not be suitable for multicasting; only identical content requests received within a short period of time of

each other would provide the conditions required for the multicasting to be successful.

6.6 OpenFlow DNS Redirection Alternative

The aspect of the proposed OpenFlow CDN system presented in this section would only be feasible in specific scenarios; it involves the application of OpenFlow's centralised control to the problem of DNS based client redirection. DNS redirection is the process used to route requests to resources located on a number of servers, when they are addressed using the same URL and hostname identifiers; this is performed for many reasons, some of which include: load balancing, latency reduction, geographic specific content provisioning and network capacity conservation [228–231].

As previously stated, the proposed DNS redirection alternative would only be able to function within specific network environments, which in this case would be a complete OpenFlow enabled network. The network would need to not only include the content servers, but also the client attachment nodes e.g. in the broadband remote access servers. It is possible that the system could function in other scenarios; however, it would not be able to fully utilise the OpenFlow capabilities and would limit the benefits that could be achieved.

An example of a standard DNS request from a user requesting a resource using its URL operates as follows:

- User requests a website using a unique memorable URL.
- The URL then begins the process of filtering through the DNS system.
- The client machines configured DNS server is queried for the specified URL.
- The DNS request is then filtered through the complex DNS hierarchy until it finds an authoritative record that it trusts; this may even be a cached result at the initial DNS server. If the IP address is not cached at some point in the DNS hierarchy, the authoritative record would then be obtained from the DNS server responsible for the CDN. The CDNs DNS server is often capable of querying various characteristics of the CDN, including server load and latency. Furthermore, it also has the option to perform geolocation of the client's DNS request using the source IP address. Using all these characteristics, it is able to return the IP address of the most appropriate server.
- The returned record will provide a link from the given URL to the IP address of the server hosting the requested content.
- The client machine then uses the returned IP address to establish a connection with the server in order to obtain the requested content.

This example is a simplified view of the DNS mechanism and includes the assumption that the DNS record is not already cached within the client machine. Using a hierarchy of DNS servers allows clients to be redirected to different IP addresses, by manipulating the records at specific DNS servers. This is the basic concept of DNS redirection and as mentioned earlier, it is used to improve factors such as server utilisation and latency. Improving these specific factors involves distributing client requests across multiple servers, while also optimising content delivery

performance by assigning servers to clients based on location. This generally involves manipulating records to redirect clients in a specific geographical area, to a server in close proximity to them.

There are many issues involved with the current method of DNS redirection, most of these problems originate from the concept of DNS caches providing stale record information, or, in contrast to this, short DNS cache times presenting scalability issues [42,228–231]; some of these issues are addressed with the use of the proposed OpenFlow redirection system. The main objective of the proposed system is to address the issue of out of date records being held at DNS servers, especially in relation to load balancing purposes. Stale DNS records can cause a multitude of issues, including blocked requests and congestion; these can be the result of some servers being overloaded while others remain unutilised. This can be partly addressed in current systems by optimising timeout values for DNS entries; however, this can introduce scalability issues when short timeout values are used. It should be noted, that optimising the timeout values in this way does not provide the level of improvement that the proposed system provides, as the OpenFlow DNS system allows both flexible and instant control over content requests.

With the proposed system implemented on an OpenFlow based network, the DNS requests would retrieve a DNS entry using standard processes, but all records throughout the network would contain a single IP address. This IP address would be registered with the OpenFlow controller as an identifier for a specific collection of servers, containing the replicated content for the URL of the DNS record. Packets destined for this IP address can then be redirected at any number of switches in the network to an appropriate server. The redirection decisions can be influenced by the specific requirements of the given network, either optimising by selecting the least utilised servers, or optimising for reduced latency by selecting a server closest to the client. The redirection process would be achieved using a system similar to the OpenFlow NAT system described in Section 6.3, whereby the OpenFlow controller would insert flow rules to modify packet headers; these flow rules would be inserted into the attachment node switch that the client was connected to, with the task of converting the IP destination address to the IP address of the appropriate server. Additionally, the controller would also configure flow rules to convert the source IP address of the server back to the original value for returning packets. Once this modification has been made, the packet is able to pass through the network using standard routing mechanisms to reach the chosen server. The benefit of this system is its ability to change, on demand, the server that any given client is redirected to; this is in addition to the insertion of generic rules for servers in a proactive approach. This is unlike standard DNS redirection solutions that require DNS records to timeout and propagate through the network. The proposed system can allow additional servers to be introduced and utilised instantly during peak times, without requiring DNS caches to expire; this also improves the reverse of this scenario, by allowing the reduction of active servers when demand is low. The described optimisation of server numbers assumes the use of virtual machines, as described in Section 2.3.1.

For the application of this system within a partial OpenFlow architecture, it would function as the system described within Section 6.4; allowing standard DNS redirection to be utilised on the public Internet, with OpenFlow redirections taking place within the OpenFlow enabled CDN. It is important to note that standard DNS redirection would send client's requests to their closest edge node of the CDN system, which as previously described, could then be manipulated

within the CDN to reach the chosen server based on optimisation requirements. Although this has already been detailed in Section 6.4, it is important to highlight that this internal redirection would otherwise be handled by DNS redirection within standard CDN systems.

The proposed DNS system would present a cost effective solution to load balancing, without having to purchase dedicated load balancing hardware. This fact does rely on the assumption of a network with existing OpenFlow support, but with large corporate providers such as Google and Akamai adding SDN to their networks [232, 233], this becomes less of an issue. It should be noted that the use of this system within the OpenFlow CDN proposed in this chapter, would also provide a more flexible and efficient solution than existing load balancing techniques. This can be achieved due to the wealth of knowledge and statistics that the OpenFlow controller has available, which allows more detailed optimisation calculations to be used. Furthermore, the scalability of the system relies mainly on the controller managing the large quantity of information from all the relevant resources, as well as maintaining control of flow information. It should be again mentioned that the scalability of controllers has been discussed thoroughly in Section 2.6.3, where it was shown that controllers are able to scale considerably well, especially when designed with a distributed architecture.

As previously discussed in Section 2.6.6, the proposed DNS redirection system is similar in concept to the system presented by Wichtlhuber *et al.* [160]; however, as described in Section 2.6.6, it differs in its implementation, which should improve scalability. This fact, coupled with its integration with the proposed OpenFlow CDN, extends the presented DNS redirection system far beyond what Wichtlhuber *et al.* [160] propose.

6.7 Functional Testing of the Smart CDN System

To show that the smart CDN concept was feasible with current technology, several aspects needed to be tested; this involved investigating systems utilised in previous chapters, as well as setting up some small testbed systems in order to validate some of the mechanisms proposed.

Initial results were collected while utilising the same Pica8 P-3290 OpenFlow enabled switch as used in Chapter 5, as it was important to use a dedicated hardware switch to assess the performance of certain aspects of the system. It was also important to specifically utilise the Pica8 switches, as OVS is part of its implementation and was required to support OpenFlow v1.4. This version of OpenFlow allows a large array of packet header manipulation options, which are generally not supported on other devices yet. In some cases, OVS was also used on a standard server machine, but the tests performed using this configuration were not resource intensive and were not focusing on performance.

Initial testing began with various packet header manipulation testing, these included combinations of source and destination IP and MAC address modifications. Flow rules were implemented to redirect traffic to alternate machines, with negligible added latency to packets[1]. This functionality is the basis for the systems proposed in the previous sections.

Initial results from this stream redirection can be seen from the results presented in Chapter 5, where the results provide an in-depth look at the performance of both stream redirection

[1]Latency was determined to be below 1ms, determining latency below this in a software system is not straightforward.

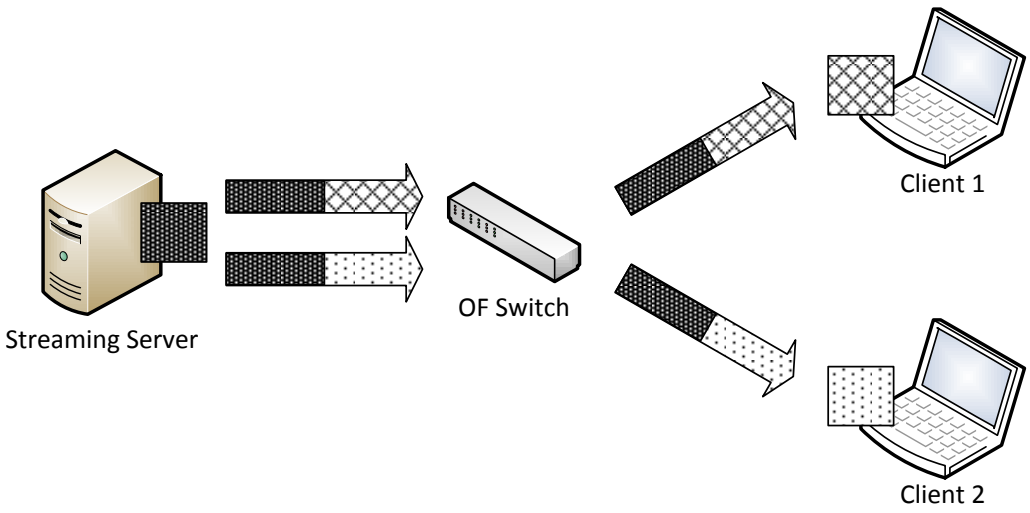
and stream migration. It can be seen from these results that traffic redirection and migration is not only feasible, but capable during transmissions of live video content without significant interruption to clients. This allows traffic to not only be engineered at the initial entry to the CDN, but also allows dynamic stream placement after streams are already active and in place. Having this functionality means optimisation can occur throughout the operation of the network, on both new and existing traffic streams. This is in contrast to existing systems which only rely on optimising the placement of new traffic entering the network. The possibility of performing optimisation during system operation enables improved and efficient resource utilisation. Another benefit of dynamic stream migration is the improved resilience against link and transcoder failure; this allows rapid migration of streams when link failure is detected, as well as redirecting to alternate servers in the event of a server failure. It is important to note that in situations such as this, the performance of the migration would not be as fast as those presented in Section 5.6, as the flow rules would not have been present before the resource failure event.

Stream duplication is another important aspect of the proposed system, with many benefits relating to the reduction in network and server load. The duplication system was tested by sending a single real-time UDP video stream to a client through the Pica8 switch, where after transmission had begun, an additional client was added to the switch. Figure 6.3 presents an overview of this scenario, along with a comparison of a system without the aid of OpenFlow duplication. In the non-OpenFlow scenario shown in Figure 6.3(a), it can be seen how two UDP streams are required to be sent from the server to the respective clients, as would be expected. However, for the OpenFlow duplication scenario shown in Figure 6.3(b), only a single UDP stream is required to be sent from the server. This is because the initial UDP stream destined for the IP address of Client 1 is duplicated at the network switch, in order to serve the Client 2 without the server providing an additional stream. For the duplication to be successful, the following header manipulations need to be performed on duplicated packets:

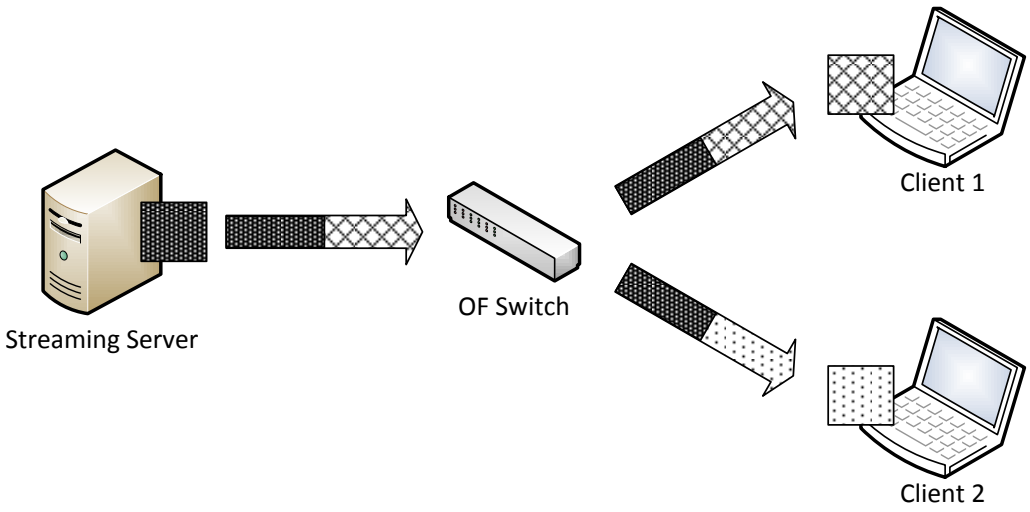
- Replace the IP Destination address with the IP address of the new client.
- Replace the MAC Destination address with the MAC address of the new client.
- Application Destination and Source Ports might also need to be modified if different from the original client.

The duplication shown in Figure 6.3(b) is only a simplified example of how the proposed system would function; however, even at this small scale, its benefits can clearly be seen. One of the main benefits is the reduction in link capacity consumed between the server and the OpenFlow switch, which is halved in the presented example. This is already a significant improvement, but with the introduction of additional clients and network links, the reduction of traffic will become even more pronounced. This achieves a similar effect to that of using IP or application layer multicasting, which is detailed by Hosseini *et al.* [163]. Furthermore, the benefits of application layer multicasting deployment can also be seen in Section 4.6, with Figure 4.3 showing a large reduction in network load; this shows how placing application layer multicasting nodes even at random locations, can provide reductions in network load when delivering real-time video content.

A further benefit of stream replication that is not immediately obvious from Figure 6.3(b), is the reduction in server utilisation that is achieved. This is due to the server effectively only



(a) Standard Streaming Scenario



(b) OpenFlow Switch Duplicated Stream Scenario

Figure 6.3: Streaming scenarios showing the benefits of stream duplication at switches.

serving a single client, which reduces both CPU utilisation, network IO and memory usage. Each additional client will require a small amount of computing resource at the server and controller, but this would just be for the initial process of setting up the required flow rules in the switches and keeping a record of the active streams.

Stream redirection results and analysis can be observed in Chapter 5, where OpenFlow was utilised to redirect a UDP video stream as well as ARP requests to an alternate transcoder. Stream

duplication results can also be found within Chapter 5, although it was only used for a limited time during the migration process. Despite its limited use within the system discussed in Chapter 5, it did function without issue and provided the content to multiple transcoders, providing a valuable insight into both its performance and capabilities.

Although the proposed systems functionality has been tested and demonstrated to work successfully, the scalability of such a system would need to be investigated further to determine the limits of the system. OpenFlow switch and controller scalability has already been discussed in Section 2.6.1, but further work is required to investigate the performance in the context of the proposed scenario to prove reliability. This would mainly involve determining the number of concurrent streams that could be served using duplication, as well as determining the overheads involved when manipulating packet headers. A possible limit for UHD packet manipulation can be seen in Section 5.5.4, where the combination of packet manipulation from the migration process and the pseudo tunnelling through the transatlantic link, caused disruption to the video stream. These tasks did involve a lot of header manipulation on the high bitrate stream, so it would be desirable to investigate this further. However, regardless of limits such as these, they could most likely be overcome using additional OpenFlow switches within the proposed system to distribute the packet manipulation tasks. Furthermore, it is likely that packet manipulation performance will be improved in future OpenFlow switches, since OpenFlow is still a relatively new technology with ongoing development.

6.8 Summary

This chapter has presented a novel OpenFlow based CDN, which uses the idea of a centralised controller and OpenFlow packet manipulation to improve on several existing CDN mechanisms. This idea is then followed with initial results to support the proposed system, including references to the relevant work performed in previous chapters. Using these novel advancements alongside the systems presented in previous chapters, allows new types of content delivery mechanisms that are able to support the increase in both UHD content and real-time streaming applications. As an extension to the proposed OpenFlow CDN, a novel algorithm to disperse traffic within the CDN architecture is presented in the next chapter; the algorithm focuses on reducing the probability of blocking, in order to improve the scalability of the video delivery system.

Chapter 7

Optimising Traffic In A Smart CDN Architecture

With a large volume of traffic flowing through the proposed CDN architecture presented in Chapter 6, especially during peak times, it is important to manage the flow of traffic within the network. This presents an NP-complete problem which can not be easily solved [234]. There are many different approaches to traffic optimisation, based on the demands and requirements for the specific network. One possible optimisation technique would be to aim to reduce the overall traffic in the network, as this would reduce overall link utilisation for the network, allowing more content to be served to additional clients. However, this approach can be further improved, by not reducing the overall link utilisation, but trying to maximise the free capacity on all links, so that the traffic is distributed evenly across the network. This chapter presents a variation of a max-flow algorithm, with the goal to optimise traffic flows within a CDN architecture such as the one presented in Chapter 6.

By maximising free capacity on links, it provides a good defence against blocking in the network, as just a single fully saturated link could in effect produce blocking. It is important to note, however, that this method of optimisation can increase the total overall load in the network, since traffic can be routed along longer paths to use lower utilised links. To further explain this, it can be seen that although a network may have a higher overall network load, as long as each link has free capacity to carry additional traffic, no blocking will occur. This makes overall network load less important in this optimisation scenario, as it is not a good metric for determining solution performance. Furthermore, since this is based on a scenario of a private CDN infrastructure, where the cost of a link is fixed and not based on capacity used, it would also not affect costs by optimising traffic in this way. The described optimisation method can only be fully realised in this case, because of the OpenFlow CDN systems ability to influence all the traffic within the network; whereas if the system only had the ability to route a subset of the traffic, it could produce worse results by interfering with other routing methods currently being used, such as shortest path routing.

The proposed network optimisation is targeted at OpenFlow enabled systems, such as the proposed OpenFlow CDN architecture presented in Chapter 6. This is due to OpenFlow enabled networks being able to rapidly configure network traffic paths, as well as controller applications such as Floodlight maintaining a complete network overview. This overview is required to enable

the optimisation algorithm to obtain topology information used in its calculations. Within the proposed OpenFlow CDN architecture in Chapter 6, the algorithm solution would be configured within the network using flow rules set up by the central controller, enabling the quick redirection and migration of traffic when required; this rapid configuration allows the network optimisation operations to be performed frequently during normal system operation, in order to maintain network efficiency. The presented max-flow network optimisation can also be used in conjunction with the transcoder migration system discussed in Section 5, which could further improve traffic usage by moving transcoding resources within the network; however, this is not considered during testing and would need to be investigated further to merge the two concepts.

7.1 Max-Flow Optimisation Problem Statement and Solution

The presented max-flow algorithm is designed to distribute demand traffic over the network, in order to maximise the free capacity available on each link. This provides a reduction in the probability of blocking, as it tries to ensure that each link has sufficient capacity to meet additional, future demands. The basis of the presented algorithm for solving the NP-complete max-flow problem was inspired from existing algorithms that solve the linear problem in polynomial time; the described linear problem that those algorithms aim to solve is a fractional flow network optimisation for use with non-IP based network scenarios. The linear problem solution is first described, as this forms the method for the novel heuristic solution to the integer case.

7.1.1 Description of the Traffic Engineering Problem

The goal of the linear multicommodity traffic engineering problem is to maximise the residual capacity remaining on each link in the network; this problem will now be formally stated following the description presented by Reed [235].

Given a graph $G(E, V)$ with a set of demands D, describing the requested traffic between pairs of vertices in V that will flow along the edges E, the Multicommodity Flow with Minimum Congestion (MFMC) problem can be formulated by determining a set of flows X, that ensures all flows traversing an edge e leave at least a proportion Ω_L of spare capacity within that edge. This can be stated as the maximisation problem:

$$\max(\Omega_L) \quad \text{s.t.}$$

$$\sum_{P_e : e \in p} x(p) \leq u(e)(1 - \Omega_L) \quad \forall\, e \in E \tag{7.1}$$

$$\sum_{p \in P_d} x(p) \geq d \quad \forall\, d \in D \tag{7.2}$$

where $x(p)$ is the flow along a path $p \in P_a$, P_a is the set of all possible paths in the network, $P_d \subset P_a$ is the set of all possible paths associated with demand d and $P_e \subset P_a$ is the set of paths that pass through edge e with capacity $u(e)$. Furthermore, Equation 7.2 confirms that demand d is successfully fulfilled. The MFMC is a linear programming problem that makes the assumption

of being able to distribute a single flow for a demand, d, across an infinite set of paths between the source and destination of the demand.

Unfortunately, the split flow design of MFMC limits its application within standard communication networks, as it is often not feasible to send traffic for specific demands across multiple paths. This can be observed when using transport layer protocols such as TCP, as their performance suffers when their path is modified [236]. For this reason, network devices which perform load-balancing over multiple paths, usually ensure that the paths taken by individual TCP sessions (or destination addresses) remain consistent throughout transmission [237]. Therefore, in current communication networks, it is generally assumed splitting flows along multiple paths is not possible; this introduces an integer MFMC (IMFMC) problem. This Integer Multicommodity Flow with Minimum Congestion (IMFMC) problem can be described as for each demand $d \subset D$, a single path p_d must be found to carry the flow $x(p_d)$, such that:

$$\max(\Omega_I) \quad \text{s.t.}$$

$$\sum_{p_d \subset P_I : e \in p_d} x(p_d) \leq u(e)(1 - \Omega_I) \quad \forall\, e \in E \tag{7.3}$$

$$x(p_k) \geq d \quad \forall\, d \in D \tag{7.4}$$

where $P_I \subset P_a$ is the set of all paths, but only including one for each demand. The IMFMC is known to be NP-complete [234], asserting that there is currently no known algorithm available which can find an optimal solution to non-trivial problems in a reasonable amount of time.

7.1.2 Methodology for Solving the Flow Optimisation Problem

In order to solve the IMFMC problem, the MFMC problem needs to be discussed. The MFMC problem has been described in various literature, however, it is rarely considered when solving traffic optimisation problems. Reed [235] discusses using the MFMC algorithm to optimise traffic placement within an Information Centric Network (ICN) architecture. It is important to highlight the algorithms intended use with an ICN, as ICNs based systems do not suffer the degradation of performance that TCP based system do when splitting streams along multiple paths. Further relevant details of the MFMC problem will be included where necessary with the description of the IMFMC, however, a detailed description of the MFMC solution presented by Reed can be found in Appendix E.1.

7.1.2.1 Solving the IMFMC

The IMFMC is an NP-complete integer problem [234], hence the aim of the presented algorithm is to provide a good, rather than truly optimal, solution. Indeed, because of the nature of the problem, it is not even possible to determine how close to optimal it is likely to be. Consequently, the presented IMFMC algorithm solution quality will be compared against the solution given by CPLEX [34], an alternative black-box solver described in Section 7.2.

The heuristic was devised by observing, closely, the operation of Algorithm E.1. From these

observations, it was discovered that the algorithm already calculated possible solutions to the IMFMC problem within each phase of its search function; however, the algorithm generally disregarded these, since fractional flow solutions almost always present a better solution to the MFMC problem. Therefore, it was realised that the solution to the IMFMC problem could be obtained with only minor additions to Algorithm E.1. This involved collecting suitable solutions within the MFMC search that could be suitable for the IMFMC problem, meaning only solutions that assigned demands to single paths were stored; this ensured that fractional flow solutions were not used for the IMFMC problem. Additionally, the minor modifications made to the algorithm do not modify the existing functionality, so the solution to the MFMC problem is still calculated alongside that of the IMFMC.

The modified algorithm can be seen in Algorithm 7.1, where it can produce solutions for both the MFMC and IMFMC problems. There were a number of modifications and extensions made to the original MFMC algorithm presented in Algorithm E.1, which are described closely in Appendix E.2; within this description, line by line changes are detailed in order to clearly show the extensions to the solution gathering process, which gathers integer solutions during the MFMC optimisation search process. The IMFMC optimisation process can also be seen in Figure 7.1, which highlights where the IMFMC branches from the original MFMC search; it should be highlighted that Figure 7.1 only shows a simplified view of the optimisation process, as the complete IMFMC algorithm shown in 7.1 has many other steps and stages during its operation. The main branch point where the IMFMC deviates from the MFMC is shown in Figure 7.1 within the main demand loop process, however, as Figure 7.1 focuses on describing the IMFMC, it does not show additional specifics of the MFMC optimisation process. As previously described, Figure 7.1 shows how the IMFMC optimisation process involves selecting integer flows from the pool of available MFMC paths through the network; the addition of this selection process is one of the main extensions made to the MFMC. This added integer selection process enables the use of the well tested MFMC fractional flow optimisation mechanisms with the optimisation of integer flow problems.

7.2 Brief Introduction to the CPLEX Solver

CPLEX [34, 35] is a black box solver which is used to provide solutions to problems specified within its own defined language. The problem is specified in mathematical form and it attempts to find an optimal solution. It does this using a wide range of complex solving techniques which are hidden from the user, although it does include specialised techniques to address graph optimisation problems making it more efficient in this domain. Although CPLEX is considered a black box solver, it is shown to produce reasonable solutions to a wide range of problems [238, 239]. This presents a suitable comparison solution to be tested alongside the presented max-flow algorithm.

The traffic optimisation problem that the max-flow algorithm attempts to solve was formulated in the CPLEX syntax so that comparison results could be gathered. Results were collected over a number of repetitions, along with variations in network and demand characteristics to ensure their reliability.

Algorithm 7.1 Adaptation of Algorithm E.1 to search for solutions to the IMFMC problem

Function MFMC$(G(E,V), D, \epsilon)$**:**
$n \leftarrow 1$; $Y \leftarrow \emptyset$; $\delta \leftarrow (|E|/(1-\epsilon))^{-1/\epsilon}$
$l_1(e) \leftarrow \delta/u(e) \quad \forall e \in E$; $\Omega' \leftarrow 0$
while $\Delta(l) \leq 1$ **do**
  $\Gamma \leftarrow \emptyset$
  for $d \in D$ **do**
    while $\Delta(l) < 1$ **and** $d > 0$ **do**
      $p_m(d) \leftarrow F(l_n)$
      $\mu \leftarrow \min(u(e) : e \in p_m(d))$
      if $d > \mu$ **then**
        $c \leftarrow \mu$
      else
        $c \leftarrow d$
        create new flow $\gamma \in \Gamma$ **where:**
        $p(\gamma) \leftarrow p_m(d)$**;** $r(\gamma) \leftarrow c$
      end if
      if $y \in Y$ **where** $p(y) == p_m(d)$ **then**
        $f(y) \leftarrow f(y) + c$
      else
        create new flow $y \in Y$ **where:**
        $p(y) \leftarrow p_m(d)$**;** $f(y) \leftarrow c$
      end if
      $l_{n+1}(e) \leftarrow l_n(e) + \epsilon c/u(e) \quad \forall e \in p_m(d)$
      $d \leftarrow d - c$; $n \leftarrow n + 1$
    end while
  end for
  if all flows in Γ **meet demands then**
    $\Omega \leftarrow \min(u(e) - r(\gamma))/u(e) : e \in p(\gamma); \gamma \in \Gamma)$
    if $\Omega > \Omega'$ **then** $\Omega' \leftarrow \Omega$; $\Gamma' \leftarrow \Gamma$ **end if**
  end if
end while
$f(y) \leftarrow f(y) \log_{1+\epsilon} 1/\delta \quad \forall y \in Y$
$f_d \leftarrow \sum_{p \in P_d} y(p) \quad \forall d \in D$
$\theta \leftarrow \min(f_d/d) \quad \forall d \in D$
$p(x) \leftarrow p(y)$; $f(x) \leftarrow f(y)/\theta \quad \forall y \in Y$
return Γ', Ω', X

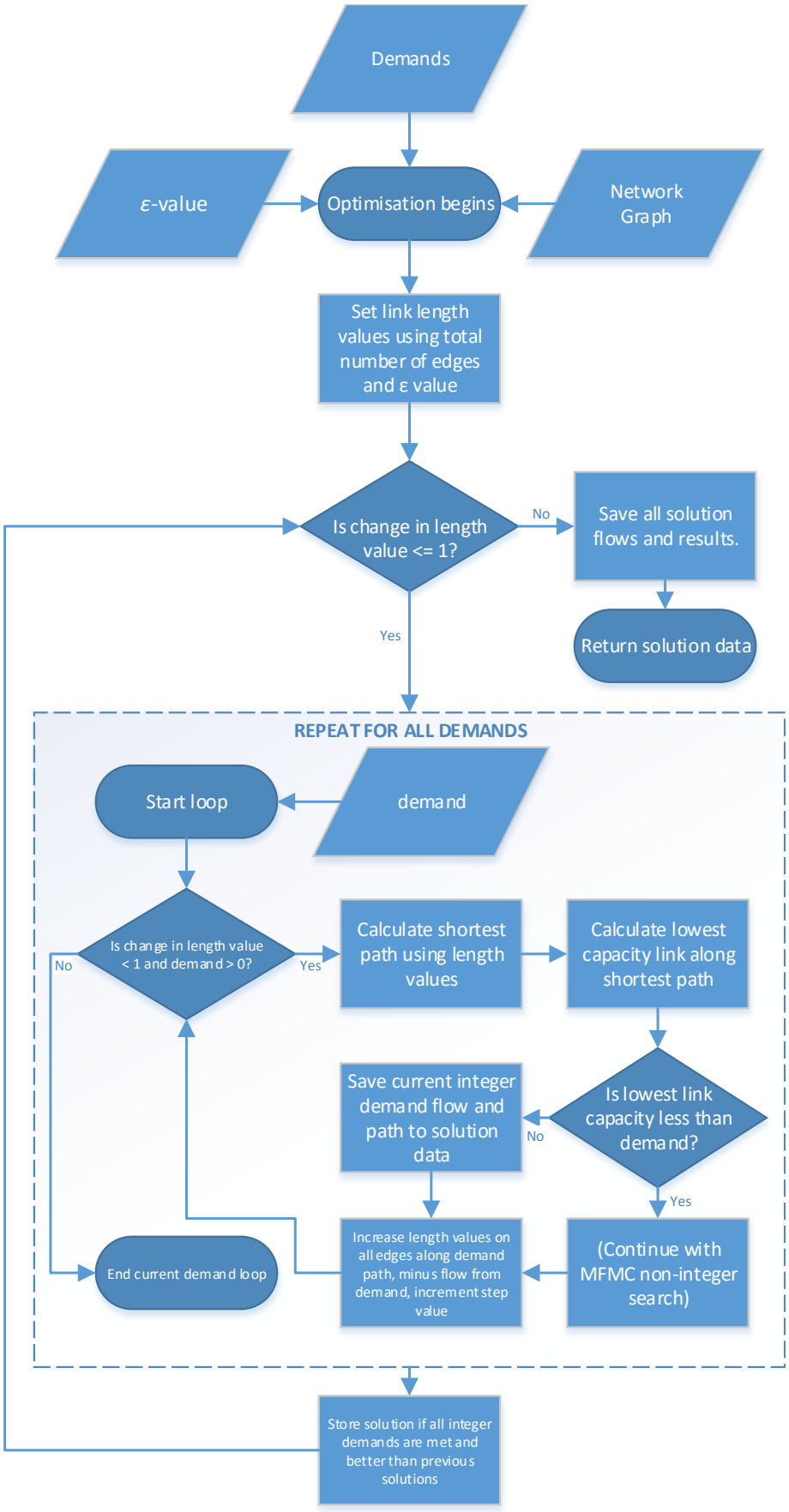


Figure 7.1: IMFMC flow diagram.

7.3 Comparing Solution Quality

Results were obtained from a modified version of the R model used in Chapters 3 and 4, with the addition of using the CPLEX solving environment. The model was modified to replicate the design of the OpenFlow CDN described in Chapter 6, due to it being one of the proposed applications for the max-flow algorithm; however, the OpenFlow CDN architecture is also very similar to other CDN architectures, so it is expected the results would produce similar, if not the same, when tested with those scenarios. Furthermore, with the design of the OpenFlow CDN, the scenario used for results collection was the delivery of real-time content to clients outside of the CDN network. Content for clients would exit the CDN using one of the edge nodes, with the concept of the exit node used being the closest to the client on the external network; for simplicity, the external network envisioned as the public Internet was not modelled, with the selection of the optimal exit node being left to random assignment. With this scenario, only the routing of traffic within the CDN was considered, as it would not be feasible to influence external traffic in a real network environment. It should be noted that although the scenario of the OpenFlow CDN architecture was used, capabilities such as switch level multicasting were not modelled; this was due to the focus being on analysing traffic placement, not the performance of the OpenFlow CDN components.

The results were all gathered over sets of 40 repetitions to ensure confidence in their accuracy. As previously stated, the model used randomly generated synthetic network topologies of varying size, which had restrictions to ensure they were fully connected graphs. The topologies were created with a node degree following a Weibull distribution with shape parameter 0.42, conforming to the results of a survey of ISP router level topologies [201]. It should be noted that the generated networks were then given other attributes, in order to match the design of the OpenFlow CDN architecture; these added attributes will be discussed later in more detail. Additionally, as with previous models, the server capacity was assumed to be infinite, with reasons for this detailed in Section 3.4. A number of network parameters were recorded during modelling to provide results based on the optimisation goals that the presented max-flow algorithm aims to achieve. It should be clarified that results presented in this section use "position dodging" to better display results in close proximity to one another. "Position dodging" is a feature within plotting software that allows similar results from variable result sets to be displayed clearly on the same plot in close proximity. This may appear as if different x-axis values were used, but this is not the case as closely grouped results relate to the closest x-axis marker. To further clarify this, x-axis values for all results are integer values ranging from 20-200 in steps of 20. Additionally, all presented results use error bars to show 95% confidence intervals, however, in some cases the intervals are too small to see.

Additional model parameters were scaled as the network increased in size, in order to maintain a moderately congested network that produced small amounts of dropped traffic. Entry nodes into the network were maintained at 10% of the total number of nodes within the network, with their selection being based on the number of connected links they maintained with the rest of the network; nodes with the most links were chosen over others, as they would provide improved load distribution across links. The number of demands was also scaled, with the exact number of demands being calculated using the formula $35+(42\times(networkSize/20))$; this specific formula was

found to maintain a congested network with small amounts of blocking, throughout the increase in network size. Furthermore, the number of demands were also divided between 4 types of video content, termed: 1080p, 2k, 4k and 8k. These content types represented the varying resolutions that were being requested by clients, with each of them requiring a certain link capacity to be available, in order for the demand to be successfully met. The required capacity of these were as follows: 1080p=40, 2k=60, 4k=100, 8k=200; with these values being related to the arbitrary units used for link capacity. The demands were divided up between these resolutions as follows: 30% as 2k, 20% as 4k, 10% as 8k, with the remaining demands being set as 1080p. All the presented values for demand provisioning were selected based on extensive observation of network model behaviour, with trial and error based techniques being used.

For an improved comparison, two Dijkstra based algorithms were also tested in addition to the proposed max-flow algorithm and CPLEX. This comparison is important as it provides an insight into how the proposed max-flow algorithm performs against existing shortest path solutions, which are utilised in many current networking environments. The first standard Dijkstra algorithm uses a basic shortest path method to produce a best effort placement of demands, whereby if the first calculated shortest path has insufficient capacity to fulfil the demand it is dropped. The first-fit Dijkstra algorithm extends on this standard shortest path algorithm, by attempting to route around congested links; thus, demands will only be dropped if no possible route to the destination is available. It should be noted that in both Dijkstra based algorithms the demands are placed sequentially, so there is no attempt to remove already placed demands in order to improve the returned solution.

Although the time factor is a measured statistic in the presented results, it is not unreasonable to assume that there would be some difference based on the language each algorithm is implemented with. CPLEX is a commercial, complex, model solver, where it can be assumed that many steps have been taken to achieve high efficiency and the smallest possible run times. The Dijkstra algorithms were implemented in C/C++ and called from R, so these could be made more efficient by implementing the whole algorithm in a more efficient compiled language. Although the max-flow algorithm is also called within the R model, the optimisation algorithm code is implemented using C++ through the Rcpp package. Rcpp is an R package which can seamlessly call C++ code from R script while handling many of the issues that are associated with switching languages, including handling object and datatype conversion.

Following on from the concept of measuring run time of the algorithms, it should be clear from the description of the max-flow algorithm in Appendix E.1, that the ϵ value can influence both the run time and solution performance. The lower this value, the smaller the search steps within the algorithm, which translates into a more accurate solution but with increased run time. For this reason, it is important to use a value which not only presents a good solution, but also does so in a reasonable time frame. After analysis of possible ϵ values, it was determined that a value of 0.1 produced acceptable results in a reasonable amount of time. One important factor that significantly affects the results is whether the network is under blocking or non-blocking conditions; consequently, the performance was analysed under both conditions. This is important due to operators aiming to maintain non-blocking networks, especially for architectures such as the proposed OpenFlow CDN in Chapter 6; however, these networks will occasionally present

blocking conditions in certain high traffic volume scenarios. The results presented in Section 7.3.1 are taken from a blocking based network, where network link capacity was configured at 1000 arbitrary units. In contrast to this, the results presented in Section 7.3.2 are taken from a non-blocking based network, with their link capacity set to infinite. Apart from the difference in link capacity, the two network model configurations were identical.

As with the previous models, network load is still defined as follows:

$$L = \sum_{d \in D} d_{s,t} |E(d_{s,t})| \tag{7.5}$$

where $d_{s,t}$ is the bitrate of the video demand d, defined earlier, and $E(d_{s,t})$ is the set of edges within the path taken by d.

7.3.1 Blocking Results

Figure 7.2 shows the performance of the presented max-flow algorithm alongside the comparison methods. This specific plot aims to demonstrate the ability of the max-flow algorithm to provide a solution that minimises the number of dropped streams within the network. It outperforms both Dijkstra based methods in almost all cases, as well as outperforming the standard Dijkstra algorithm by a significant margin. CPLEX does at times provide a marginally better solution than the max-flow algorithm; however, there are other factors that are important to this comparison, which are discussed later in this section. As both CPLEX and the proposed max-flow solution give very similar results, it might be, tentatively, concluded that they are achieving optimal, or near-optimal solutions.

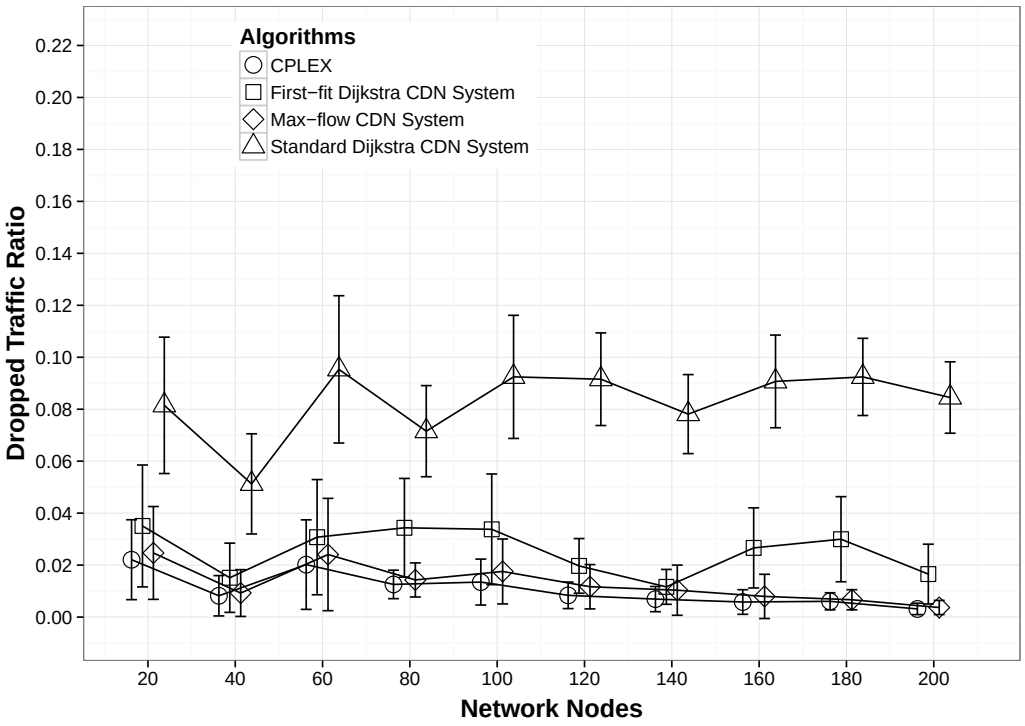


Figure 7.2: Dropped streams for optimisation methods.

Figure 7.3 compares the mean path length of demands as they pass over the network. This is an important metric, as hop count can both increase total link utilisation, as well as increasing latency for the stream. From Figure 7.3, it can be observed, that as expected, the Dijkstra's shortest path algorithms perform the best in this comparison, since their main design specification is to route traffic along the shortest path. It should be highlighted, however, that the Dijkstra shortest path algorithm shown in Figure 7.3 has a significantly higher dropped stream count than the max-flow and CPLEX scenarios, as can be seen in Figure 7.2. Although this variation in blocking makes comparison increasingly complex, it is important to show this within a blocking network to allow a more realistic performance analysis; this is because the main aim of the max-flow algorithm is to reduce blocking within the network. The comparison between CPLEX and the max-flow is also viable in this case as their dropped stream counts are very similar.

An additional result that needs to be analysed is the mean link utilisation within the network. This is an important metric not only because it is related to the main optimisation goal for the max-flow algorithm, but also as it can heavily influence the probability of blocking occurring when additional traffic is added to the network. As can be seen from Figure 7.4, the mean link utilisation when using the max-flow algorithm is significantly reduced when compared with CPLEX. As before, it needs to be highlighted that the results in Figure 7.4 were not collected from a non-blocking network, so when comparing the results, the dropped streams shown in Figure 7.2 need to be considered. In the case of comparing the max-flow and CPLEX scenarios, this is less of an issue, as the number of dropped streams are minimal in both these scenarios and generally

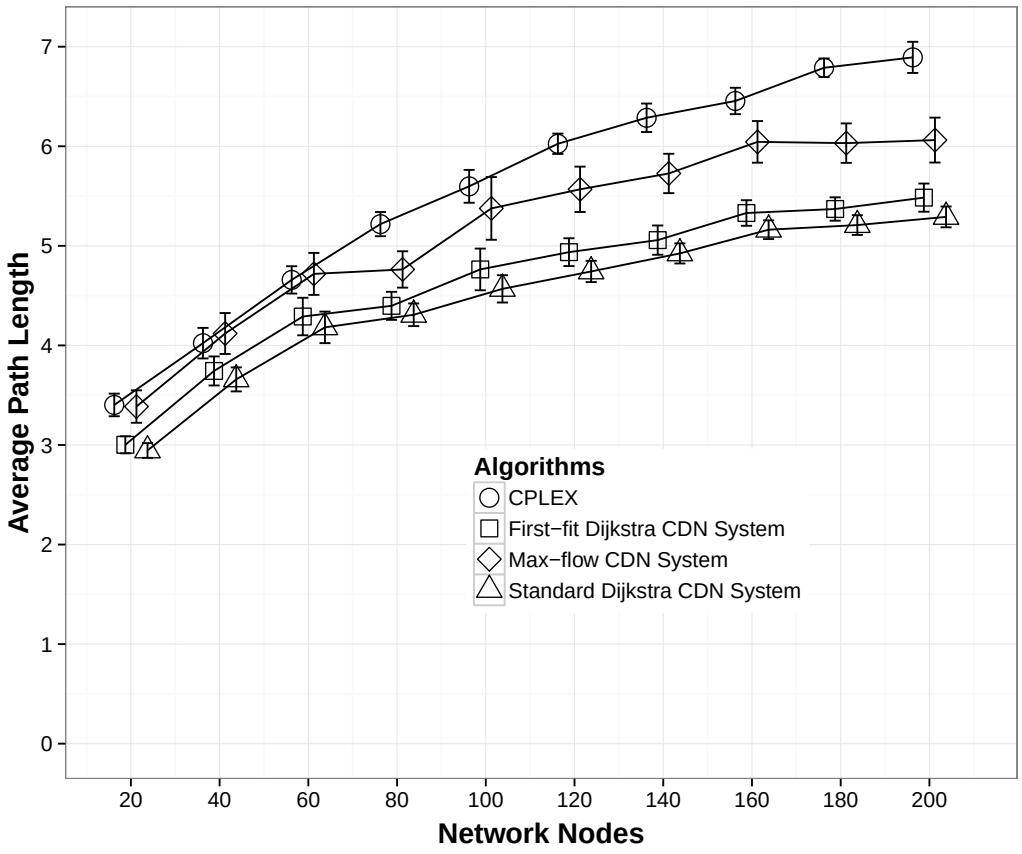


Figure 7.3: Mean path length of demands for optimisation methods.

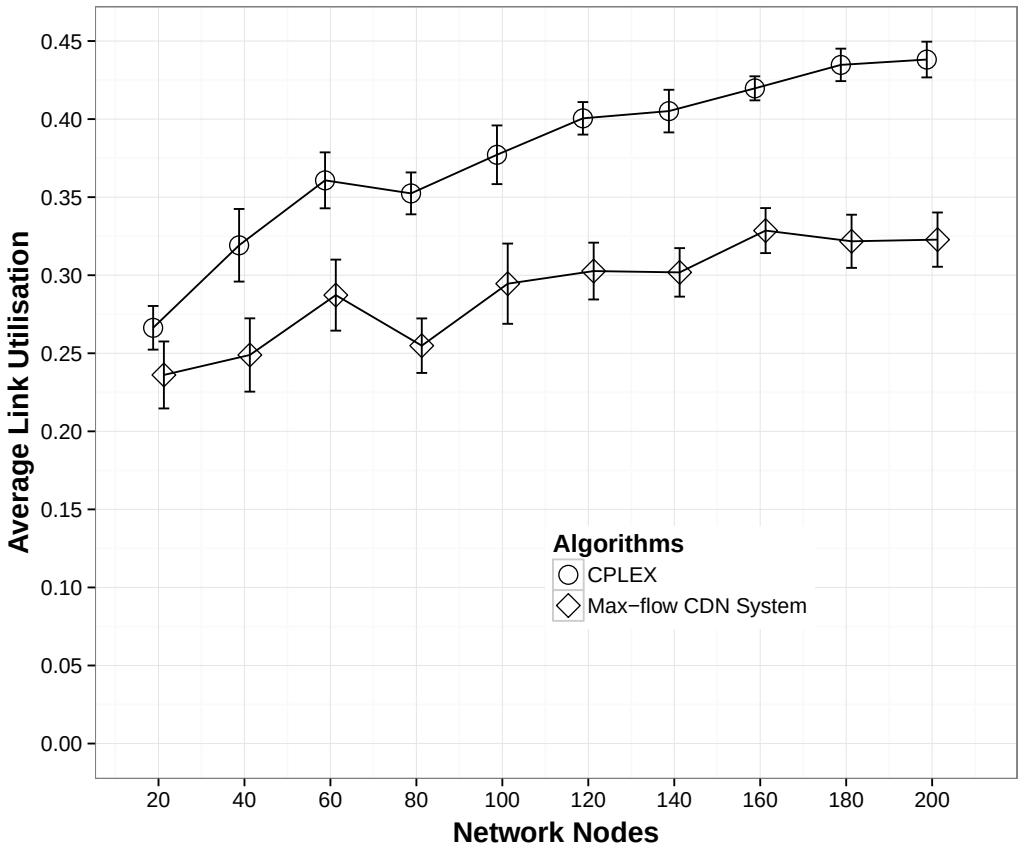


Figure 7.4: Mean link utilisation for optimisation methods.

they produce a similar number of dropped streams. However, since both Dijkstra based algorithms produce high levels of dropped streams which vary significantly, they have been omitted for clarity, as the results may appear confusing. It should be noted, however, that as expected with a high number of dropped streams, they produce an average utilisation less than both the max-flow and CPLEX optimisation scenarios.

Network load also provides insights on the optimisation performance of the different scenarios; however, it should be made clear that the optimisation goal of the max-flow algorithm does not aim to reduce network load. Figure 7.5 shows these results, but again it should be highlighted that results were gathered from a blocking based network, so dropped streams shown in Figure 7.2 need to be considered. As expected the Dijkstra based algorithms show network load below that of the other algorithms, due to their significant dropped traffic. One interesting characteristic shown, is the large separation between the max-flow and CPLEX results, considering their dropped streams are relatively equal; this is an important observation, as it demonstrates that although in each case nearly all demands are met, the max-flow algorithm does this more efficiently than CPLEX by conserving network load. This can also be observed from results in Figure 7.4, where CPLEX link utilisation is much higher than that of the max-flow scenario. With this reduced network load and link utilisation, the max-flow algorithm should theoretically allow increased numbers of additional demands to be met when compared to CPLEX.

Further to the analysis of the solution performance, it is also important to discuss the run time of the presented algorithms. Figure 7.6 presents this comparison, which shows that as the network

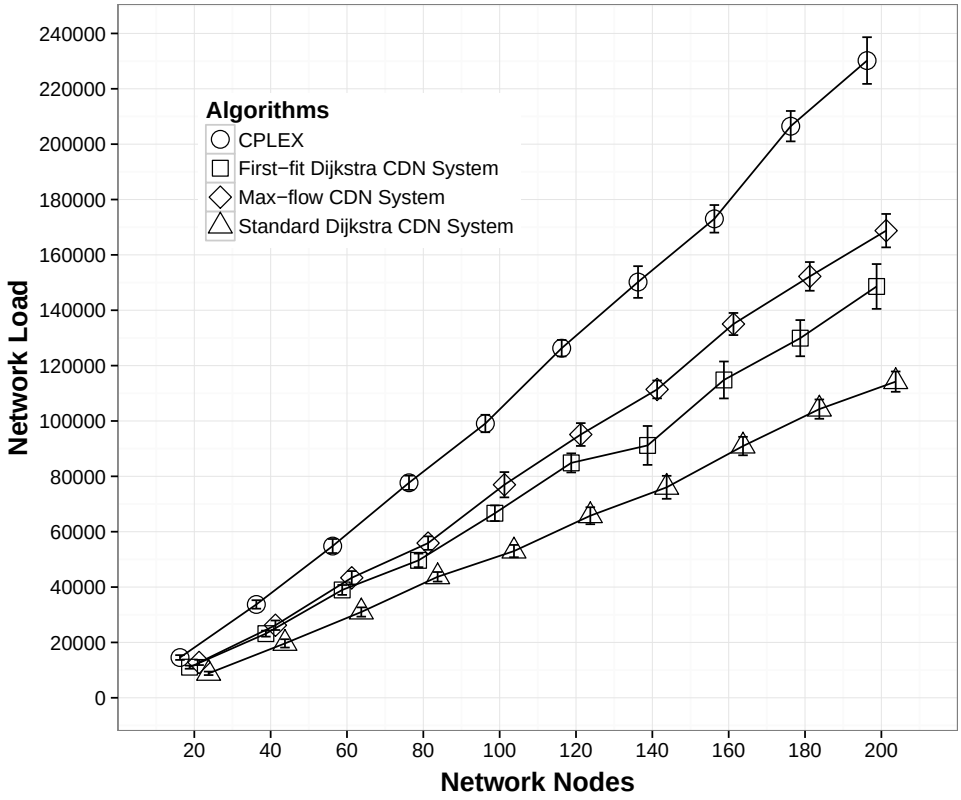


Figure 7.5: Network load for optimisation methods.

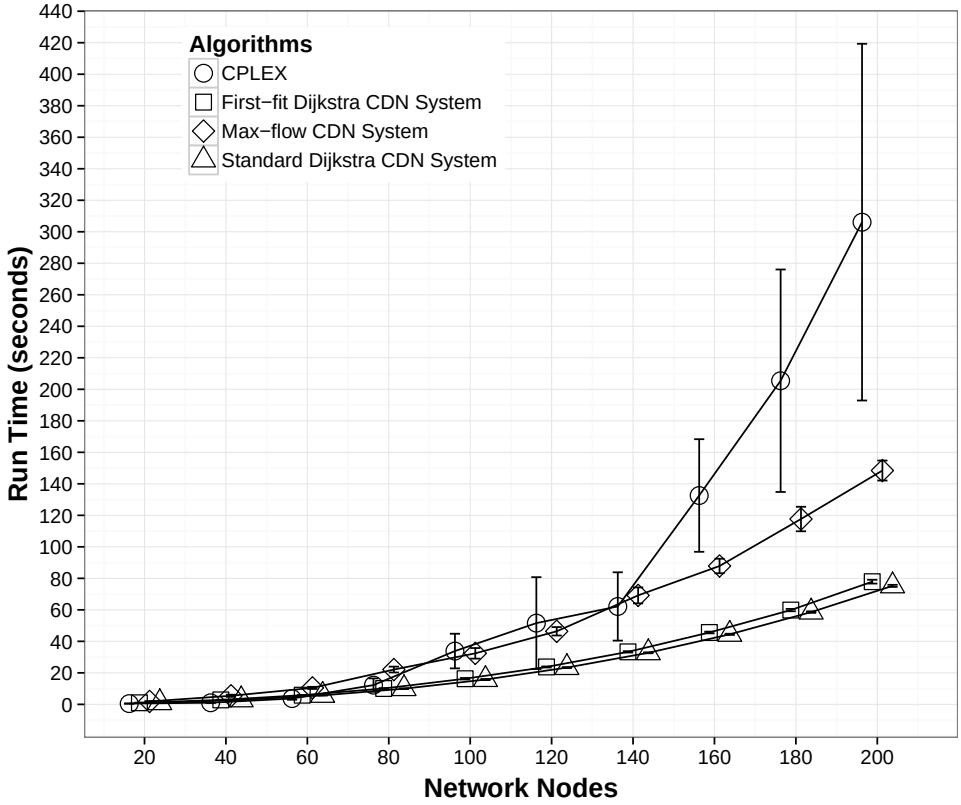


Figure 7.6: Run time for optimisation methods.

complexity increases, the max-flow algorithm provides solutions in a relatively short amount of time in comparison to the CPLEX scenario. As expected the Dijkstra algorithms complete in a faster time, as their calculation involves little more than finding a possible shortest path route. It should be noted, however, that as mentioned before, it is realised that the language each algorithm is implemented with does affect the results of this. For this reason a more conclusive study of run time will need to be investigated for reliable results; however, these initial results do give a reasonably good indication relating to CPLEX and the max-flow comparison, as they are both computed in preferred languages. As previously mentioned, the ϵ value used within the max-flow algorithm significantly influences the run time and quality of the algorithm, so further investigation into varying this value beyond what has already been performed is still required.

7.3.2 Non-Blocking Results

The results presented in this section provide insights into how each optimisation mechanism perform in a theoretical non-blocking environment. To accommodate this new scenario, network link capacity was made large enough to ensure that no blocking would occur, while keeping all other network and demand characteristics as described in the previous section. Dropped demands and mean link utilisation results are omitted here as they do not apply to a non-blocking environment and would not contribute to the analysis. Furthermore, it should be mentioned that in Figures 7.7 and 7.8, the Dijkstra based algorithms can be seen as close to, if not equal to the optimum in these metrics; this is due to the lack of possible blocking on links meaning the shortest path is always available, which is generally the most efficient way to send the traffic through a non-blocking network such as this.

Firstly, the mean path length of demands is considered for the non-blocking scenario, which helps display how the algorithms perform when there are, in effect, almost no restrictions on capacity across links. The results for this are shown in Figure 7.7 where it can be seen that as expected, the Dijkstra algorithms perform almost identically, as there are no overloaded links in the network to block traffic and cause redirection. They also provide the best results for this metric, which is also as expected based on their shortest path objective. An important observation from these results is the significant difference between CPLEX and the max-flow scenarios, which show how the max-flow provides a significant improvement over CPLEX. This can be explained from the optimisation criteria given to CPLEX, which provided the optimisation objective to place all demands in the network without regard to optimising path length. This observation provides additional confirmation of the reliability of results shown in Figure 7.3, showing that the max-flow places traffic in the network in a more efficient way than CPLEX.

Network load is also affected by path length, so the results shown in Figure 7.8 present expected results in relation to those presented in Figure 7.7. It shows how the large path length found within the CPLEX solution, causes significant load to be placed on the network; in contrast to this, the Dijkstra based algorithms achieve the lowest network load, due to the use of shortest path routing. The max-flow algorithm achieves an adequate solution with respect to network load, providing results close to the Dijkstra based methods. This shows the efficient routing of demands without capacity restrictions on links, showing that the max-flow algorithm can still distribute load across the network in non-blocking scenarios.

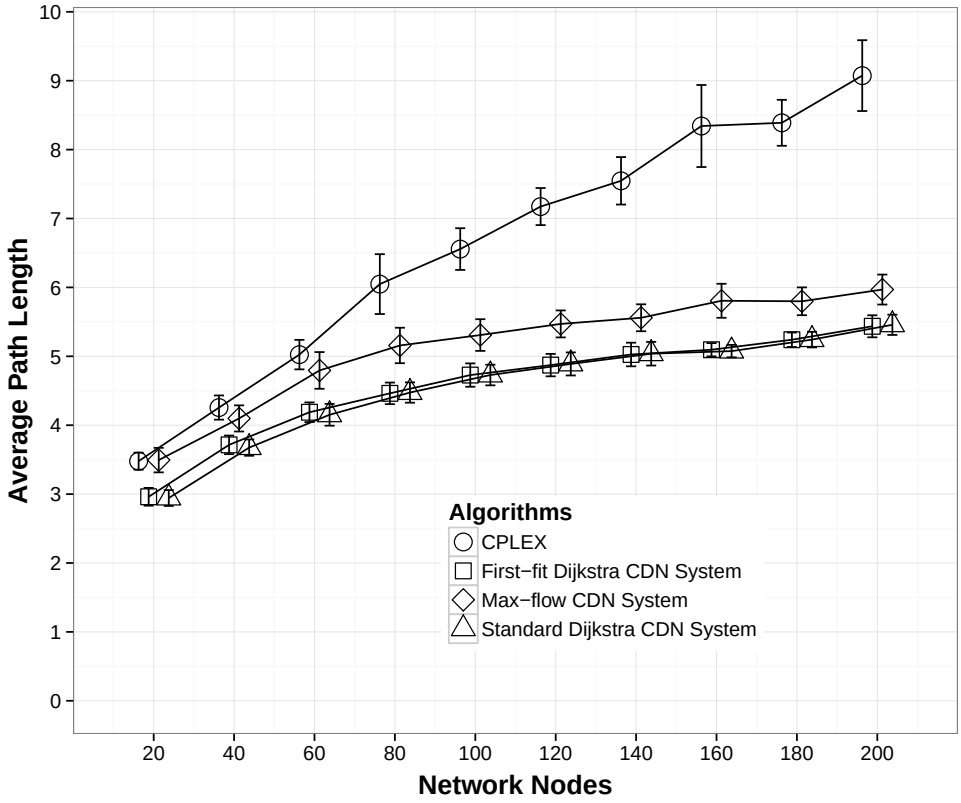


Figure 7.7: Mean path length of demands for optimisation methods in a non-blocking network.

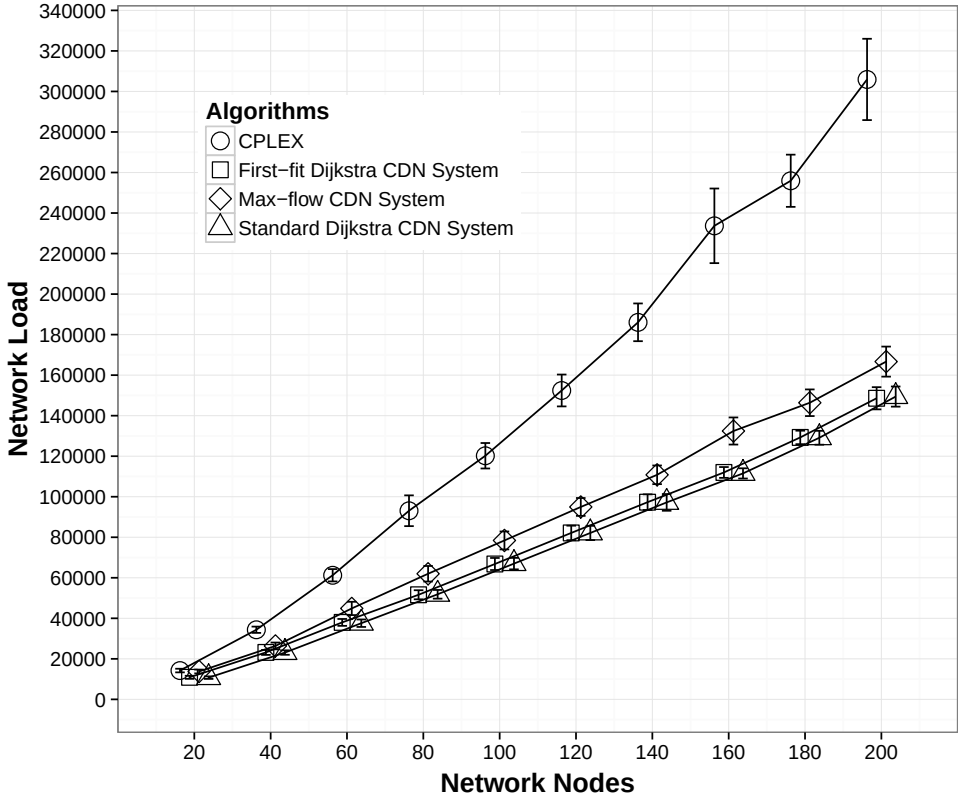


Figure 7.8: Network load of optimisation methods in a non-blocking network.

Further to the analysis of run time presented in Figure 7.6, it is important to investigate this aspect within a non-blocking scenario; this can be seen in Figure 7.9. From these results, it is clear that the max-flow algorithm is quicker in this scenario than the blocking scenario, which can be attributed to the relaxed restrictions on network capacity. It can also be observed that CPLEX performs significantly faster than the other optimisation methods, outperforming even the Dijkstra based algorithms; however, this can be expected based on its inefficient placement of demands shown in the previous non-blocking results. This reinforces the statement that CPLEX aims only to place all the demands, without optimisation of path or link utilisation.

By observing the results presented in this section, it is clear that the max-flow algorithm places demands considerably better than the CPLEX based approach. The run time of the max-flow algorithm may be larger than that of CPLEX, but it is still within an acceptable time frame. Furthermore, using CPLEX within this type of scenario would not be advisable even with its efficient run time, since this would not only increase latency for communications with its large path lengths, but also present a highly inefficient use of network resources. It should be noted that neither the max-flow or CPLEX optimisation methods were optimised specifically for non-blocking usage, so it is reasonable to assume their solution performance and run time could be improved with adaptations to their design. Further research into improving these algorithms for non-blocking environments is not discussed further, due to the planned application of working within a system where blocking is probable in certain scenarios; however, it may be an interesting area for future work when looking to apply them to other applications.

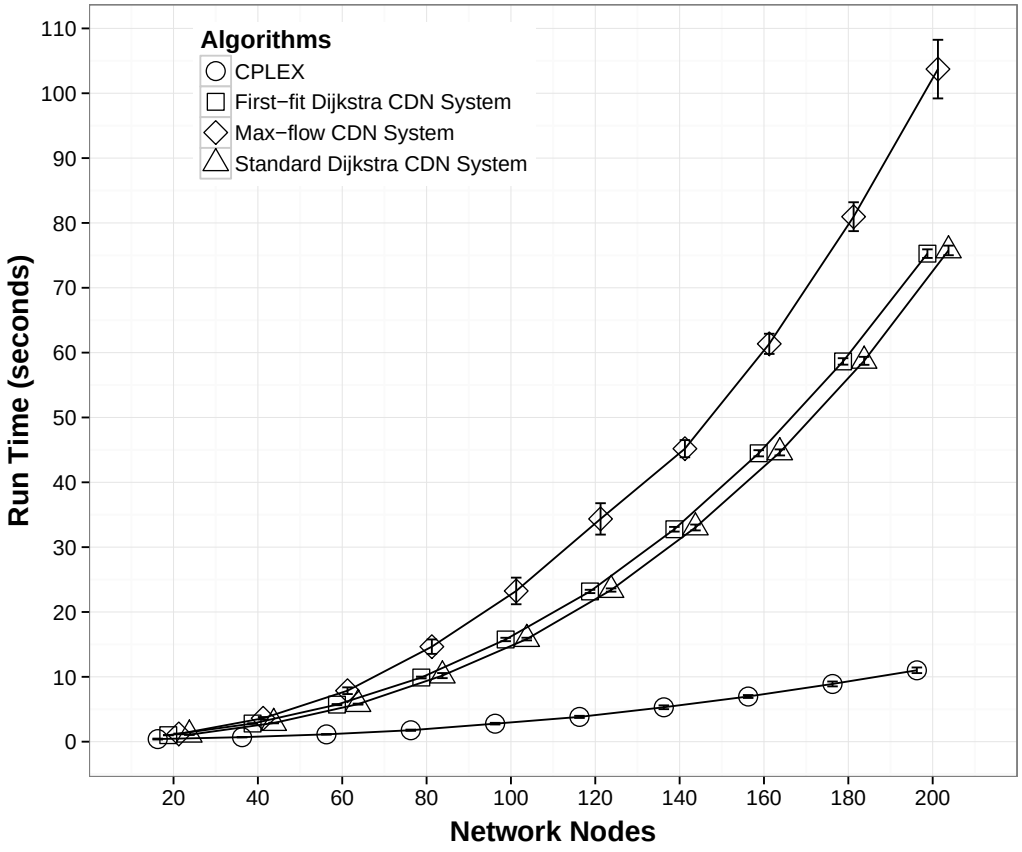


Figure 7.9: Run time of optimisation methods in a non-blocking network.

7.4 Summary

This chapter presented a novel max-flow algorithm to disperse traffic throughout the proposed OpenFlow CDN architecture presented in Chapter 6. The algorithm reduces the probability of blocking, by minimising congested links in the network. The use of the max-flow algorithm within the proposed OpenFlow CDN system would provide efficient use of network resources, enabling more demands to be met compared to standard shortest path routing techniques. Although the presented max-flow algorithm is tested using the scenario of delivering video content within the OpenFlow CDN, the algorithm is not restricted to this scenario and could be used to optimise other network traffic.

Chapter 8

Conclusion

Within the results section of each chapter, it can be seen that certain conclusions have already been stated. This chapter will provide an overview of these conclusions, followed by a critical analysis, highlighting the novel contributions of the presented work. Only brief analysis of relevant results will be given in this chapter, as detailed discussion and descriptions of results have already been provided within the relevant chapters. Conclusions for each chapter will first be presented individually, followed by a discussion of possible future work in Section 8.6. Finally, the contributions and conclusions of each chapter will be considered together in Section 8.7.

8.1 P2P-CDN

It can be observed from the results presented in Chapter 3 that the use of additional distribution points of content, such as in the case P2P, is beneficial to reduce both network congestion and the probability of blocking. The CDN architecture also displays results to this effect, with additional content distribution points reducing dropped streams occurring within the network. In order to increase the distribution points for the content, while also ensuring the network remains resilient and financially viable, a fusion of specific aspects of both CDN and P2P is desirable. Although direct testing was not performed on the amalgamation of P2P and CDN technology, it is important to highlight that this has been shown to be effective in the referenced literature. Both fusion methods described show improvements over the individual technologies, showing that on their own, P2P and CDN have several issues that can be overcome.

The results presented in Chapter 3 show a clear relationship between the number of content distribution points and network congestion; furthermore, it is known from previous literature that the placement of these distribution points heavily influences the performance of the system. With these concepts in mind, it can be concluded that in order to improve video delivery mechanisms, both the number of distribution points and their placement need to be optimised to produce the best solution. Following on from this conclusion, the remainder of the thesis aims to address the placement optimisation problem, with the focus on the distribution of real-time content; although real-time content can not be cached in the same manner as the stored content found in many CDN and P2P system, it is possible to use transcoders working as application layer multicasting nodes to gain similar benefits.

Chapter 3 also brings attention to the cost/performance balance that is found within many

content delivery systems. In the case of CDN architectures, the number of servers maintained requires close analysis and discussion according to performance and financial requirements. The aim will generally be to maintain the least amount of servers required, in order to provide the content at an acceptable quality to all users; however, this objective is not always possible due to financial constraints, meaning a P2P solution might sometimes be considered as an alternate option. It is important to highlight how the results show a decline in performance gains from adding additional servers; adding additional servers past a certain number is unnecessary as all demands would be met regardless. Determining the number of servers required to meet all demands is a difficult task, as at its design stage, it will be unknown how many requests will be made during peak times. Having a system that can add additional servers during operation is an improvement, which is why Chapter 5 focuses on allowing dynamic live migration of transcoding resources.

8.2 Transcoder Placement

It has been determined in Chapter 4 that positioning transcoders within a given network using the developed novel heuristic, is superior to random placement in terms of network load and blocking reduction. It is also shown to provide similar solutions to a GA based approach, but in a fraction of the processing time. The improvement in solution calculation time is important, as it allows the possibility of its use within an online system for live content; this is in contrast to the GA, which would only be suitable in an offline scenario or within a network that is small in scale. Furthermore, the GA's run time is observed to increase rapidly as the complexity of the network grows, making the GA unsuitable for use in larger scale scenarios.

The presented placement algorithm provides a foundation for the next body of research, involving the physical migration process of the transcoding resources. Together, they provide a system that can determine optimised positioning for transcoding resources, then migrate these resources when required during normal system operation. The fast run time of the heuristic allows optimisation to occur frequently during system operation, allowing additional traffic to be served when new clients join the stream. This is in contrast to existing systems that would be restricted by the static nature of the transcoding resources; this static nature can cause new client requests to be dropped, when existing transcoder placements become insufficient to supply the current demand.

It is important to note that the presented heuristic algorithm could be utilised for scenarios other than the optimisation of transcoding resources, such as the optimisation of application layer multicasting positioning. This becomes desirable in a scenario such as the one mentioned in Section 5.1, allowing efficient placement of multicast enabling nodes when scalable video coding mechanisms are used. It should be highlighted that the reason for this dual functionality, is due in part to the transcoders also performing application layer multicasting after the transcoding process.

8.3 Transcoder Migration

It is clear that the novel migration system presented in Chapter 5, provides a significant advancement over migration techniques such as "stop and copy". The system allows a seamless switchover of a live real-time UHD transmission, moving the transcoder from one location to another, without significantly affecting the clients viewing experience. One of the main contributions presented is the improvement of the migration system to the point of a < 0.1 second switchover of streams, which enables a near seamless viewing experience to clients. The presented migration system provides the second part of a real-time UHD video delivery system, which would also utilise the heuristic presented in Chapter 4 to optimally place and relocate transcoding resources throughout the network, during normal system operation. The proposed system would enable efficient delivery of UHD content to a large number of clients. It is important to note that the migration system operates within the network, requiring no modification to the client or server; the system can, therefore, be easily integrated with existing systems at a low cost. The only additional equipment that might be required for the migration system to function correctly, are the OpenFlow enabled switches; however, with the increasing adoption of SDN technology, especially in cloud environments, OpenFlow devices may already be available in current architectures. Further to this, some switch manufacturers are also beginning to offer OpenFlow enabled operating systems for their network devices, further improving the possibility that OpenFlow enabled devices may already be available within a given network.

Although the final system was implemented using UDP, it is shown in Section 5.4 that other network protocols were found to function in certain scenarios. It would be beneficial for some of these protocols, such as RTP, to be implemented within the final system to improve streaming quality. Additional work would be required to overcome some of the issues encountered with using these protocols, such as delay and packet reordering problems. The work required to achieve this is described in Section 8.6.3, which describes modifying the RTP client at the transcoder to improve its handling of high bitrate RTP streams. The research into adapting the system to use other protocols is considered an area of future work, which in addition to improving the video streaming system, may also enable the migration technique to be applied to other applications.

8.4 OpenFlow Smart CDN

The proposed architecture in Chapter 6 describes several novel aspects that need to be highlighted; these aspects work together to create an improved CDN system, that can deliver live and stored video content to large volumes of clients. The first aspect of this system is the novel OpenFlow based NAT system, which is described in Section 6.3; this OpenFlow NAT system bridges the internal CDN network with external networks, such as the public Internet. The proposed mechanism improves on existing systems by allowing the centralised controller to optimise server selection and exit node location for incoming content requests; further to this, the proposed OpenFlow NAT system also provides the functionality to track all traffic streams within the CDN. The use of a central controller that has knowledge of both internal machine and network statistics, allows improved optimisation and traffic control over what existing standard router enabled NAT systems can provide. The second novel aspect of the OpenFlow CDN is the concept of using

switch level multicasting, which is initially described in Section 6.4. The multicast functionality is achieved by utilising the packet duplication and header manipulation features of OpenFlow. This stream duplication feature allows both server and network load to be significantly reduced, as it moves a large volume of load to the edge nodes, while also mitigating replicated content travelling over the network. As described in Section 6.4, the multicast system would be more prevalent in the real-time streaming scenarios, however, as discussed in Section 6.5, it could still be utilised in certain cases for stored content scenarios. The final novel aspect of the OpenFlow CDN, is the proposed OpenFlow DNS redirection system; which is detailed in Section 6.6. Although the DNS redirection is similar to that of the work presented by Wichtlhuber *et al.* [160], the system presented in Section 6.6 uses a proactive flow insertion approach, while also retaining the ability to operate on a per request basis.

Each aspect of the OpenFlow CDN has been shown to be feasible, with testing being performed mainly with OVS based OpenFlow switches working with a Floodlight controller. Although these initial results do support each aspect of the proposed architecture, additional work is still required to implement each aspect within a complete working system; the concept of which is discussed later in Section 8.6.4.

8.5 Max-flow Traffic Optimisation

It has been shown that the presented max-flow algorithm in Chapter 7, provides an optimised solution for traffic placement. The solutions it produces compares closely with the results gathered from CPLEX, especially in relation to reducing dropped demands in the network. As CPLEX and the presented max-flow algorithm provide results with only marginal differences, it could possibly be concluded that they are achieving close to, or even optimal solutions; however, in order to fully confirm this, further testing would need to be completed. The presented max-flow algorithm provides solutions with many improvements over those produced by CPLEX in a blocking environment, these include: reduced link utilisation, network load and run time. The max-flow algorithm achieves these improvements by aiming to minimise the probability of blocking, by distributing traffic across all network links. Consequently, this means no small subset of links become significantly more congested than others in the network. Considering it might only require a single congested link to cause a significant amount blocking within a network, the presented max-flow algorithm significantly reduces the risk of any blocking occurring, depending on its input parameters. An additional benefit of the max-flow algorithm distributing traffic across the network, is the network's ability to support the supply of additional future demands without the requirement of re-optimisation.

In order to utilise the max-flow optimisation within the described OpenFlow CDN system, the optimised paths would need to be implemented with flow rules configured by the central controller. The use of OpenFlow in this scenario provides the flexibility required to rapidly configure traffic paths in the network; this would not be feasible using standard non-SDN based routing techniques, which either have many restrictions on the paths that are configured, or require complex device configuration. The OpenFlow controller would also have all the necessary information required to perform the optimisation calculations, enabling quick redirection and migration of traffic flows within the network. Additionally, the rapid modification of flow rules within the OpenFlow system,

would allow the max-flow optimisation to occur frequently in order to maintain network efficiency throughout normal system operation.

8.6 Future Work

Some future areas of research for specific topics have been discussed briefly in their relevant chapters; however, there are a number of areas of research that can be explored from extending the work presented as a whole. One area that would be an interesting area of research is the implementation of the presented migration system on a large scale streaming environment. This would involve testing the system with a large volume of streams, as well as utilising the placement algorithm within an OpenFlow controller to automate the migration mechanism. This would provide important insights into the system scalability, as well as being able to observe any complications that would occur. Following the combination of these systems in a large scale environment, an extension to this would be the study of how the migration system might perform when used alongside the presented max-flow algorithm; the combination of these two systems would be an interesting research area, as it would be beneficial to have the traffic flow optimisations working in parallel with the transcoder placement optimisation.

8.6.1 CDN/P2P

The work presented in Chapter 3, presented a brief look at the fusion of CDN and P2P technologies. The results, however, only went so far as to test the individual capabilities of each technology to provide content. An interesting future area of research would be to investigate the performance of the different implementations of CDN-P2P fusion architectures. This may give an insight into improvements that could be achieved when delivering real-time video content, beyond those found with analysis of the CDN and P2P results. Only simple manipulation of the previously used network model would be required, with the possibility of comparing multiple content delivery architectures across the same given network.

8.6.2 Transcoder Placement Optimisation

The fact that the algorithm focuses on the placement of a network accessible resource, means it is highly likely that its uses could extend past the optimisation of application layer multicast nodes and transcoders. It may be possible to utilise the heuristic to help solve additional optimisation problems, especially those that focus on placement on network resources. The investigation into this concept would be an interesting area of future work, as it could broaden the use cases that the heuristic could be utilised. One possible optimisation problem that might benefit from the presented heuristic would be the placement of network caches. Analysis of the heuristic performance within a cache placement problem would provide an important insight into how the heuristic would optimise the placement of stored content, rather than the designed real-time streaming scenario.

8.6.3 OpenFlow Transcoder Migration System

The migration system presented in Chapter 5, provides a number of areas of future work. One of these areas would be the integration of the migration control system within the OpenFlow

controller application, which would remove the current dependency on direct SSH based control of OVS; this concept is discussed briefly in Section 5.3.4.1. By removing the dependency on SSH based OVS control, it would enable the migration system to be utilised on any OpenFlow enabled system, rather than the limited range of devices that utilise OVS. Improving the compatibility of the migration system in this way would be an important step to integrating the system within existing OpenFlow architectures, which may not utilise OVS or provide direct SSH access. Integrating the migration system could be done in a number of ways if the Floodlight controller was utilised; one of the easier methods for achieving this would be to use the "Static Flow Pusher" REST API of Floodlight to replace the SSH based system calls. The "Static Flow Pusher" API is able to receive OpenFlow instructions in a standardised format, which then forwards the instructions to the relevant OpenFlow devices using the OpenFlow protocol. Additionally, the described API can also be used to gather and return traffic and switch statistics, providing the ability to determine when to migrate using either the heuristic presented in Chapter 4, or another means. Unfortunately, the fact that the "Static Flow Pusher" API is accessed as a simple web based SOAP request from an IP address, means the system would still require an additional application to generate these requests and manage the migration.

In order to completely remove the requirement of additional software to handle the migration, the migration control mechanism needs to be included within the existing Floodlight controller. Luckily, the Floodlight controller was developed to make integrating custom control systems a simpler task, by keeping a very modular design. The modular design would allow the migration mechanism logic to be packaged within a "Floodlight module", directly interacting with the controllers standard operations. With the migration module developed, it could then be coupled with a module containing the heuristic presented in Chapter 4. The combination of these two modules would provide a fully automated system, enabling efficient and dynamic real-time content transcoding.

Another area of future work is the re-engineering of the RTP client within the FFmpeg transcoding software, as there were various issues handling high bitrate traffic during the testing presented in Section 5.4.2. At current, the RTP header overheads and reordering process make it unsuitable for FFmpeg to handle both incoming and outgoing packets of high bitrate content using RTP, as it causes delays in the streaming process. These delays have effects that include artefacts appearing in the video, stuttering of playback and even complete stream failure. Having observed the playback issues caused by this in multiple scenarios, it was determined that although the transcoder could handle streaming RTP to the client, receiving the stream as an RTP stream from the server at the same time was not possible. In order to overcome the delays and overheads caused by RTP, it would be desirable to re-engineer the RTP client at the transcoder, in order to speed up content processing. One possible modification which could be made to improve performance, is to the operation of reordering packets, as this adds considerable delay to the stream processing operations. The modification would involve dropping or forwarding incoming packets that were out of order, rather than the usual reordering process found in RTP. With the described RTP client modification in place, it is possible that the system could utilise RTP throughout, without the delays caused by the transcoder processing. Although the described RTP issue was observed on a number of machines, it is also possible that an improved machine with multiple

network cards would be able to handle the increased load without RTP client modification; this scenario would also need to be investigated.

One final area of future interest includes the possibility of utilising the migration system with other types of applications. The concept of using the system with other applications also provides further interest in developing synchronisation between the two parallel streams at migration time, which would ensure that no packets were lost or duplicated during the switchover. This would not provide much benefit to the current video streaming scenarios, as they are able to cope with small amounts of duplicate or lost packets; however, synchronisation would be crucial in adapting the system to support applications which depend on completely lossless transitions, such as those that may rely on transactions. One such scenario that would require transaction support, is a business server migrating while handling sensitive financial operations transmitted over the network.

8.6.4 OpenFlow CDN

One area of future work that would be an important step in the development of the OpenFlow CDN, would be the design, analysis and implementation of the security enhancements discussed in Appendix D.1. These would be a worthwhile addition to the OpenFlow CDN system, presenting a secure network for service providers to distribute their content. The security enhancements would be especially beneficial for shared hosting scenarios, where the OpenFlow CDN infrastructure was being utilised by multiple entities for content delivery. In the case of a shared hosting environment, the vulnerability of a single provider would usually degrade the security of the entire network; however, the use of the security enhancements in Appendix D.1 would help mitigate these types of security issues. Although current implementations can achieve similar traffic segregation using VLANs, this is only the small part of the wider security improvements that can be offered by an OpenFlow based security system within the CDN.

A further area of research that would be beneficial would be an in-depth performance analysis of the OpenFlow CDN, as this would be an important characteristic to determine before further work into its development was begun. The analysis would need to focus on several of the independent systems to assess their performance. One of the systems that would need to be investigated, was the OpenFlow smart NAT system, which would be required to handle a large number of requests if used in a large production system; it has been previously shown that this type of system is scalable to meet the needs of a large content delivery network, since only initial requests would need to be handled by the controller application. Packets arriving following these initial requests would be handled directly within the switch flow tables, helping to mitigate controller scalability issues. Another controller based scalability concern that will need to be investigated further, is the capabilities of the controller to handle placing traffic within the CDN, as this is one of the major components of the proposed system; however, it has been shown in various literature how controller applications can scale very well, especially if the controller is built in a hierarchical design. Following on from the scalability of the controller to handle incoming requests and traffic management tasks, the performance of the OpenFlow switches ability to manipulate traffic will need to be more closely investigated. This is due to a potential bottleneck with packet header manipulation being identified during testing in Chapter 5, which highlighted how high bitrate traffic, such as UHD streams, would be delayed if increasingly complex header manipulation was

used. However, as previously discussed in Chapter 5, the limit on manipulation operations could possibly be overcome by splitting the manipulation tasks across multiple switches within the system. A further performance aspect of the system that could be investigated is the acceptable levels of latency that are required for streaming certain content; this is an important issue to study, as it will determine the system's viability for certain applications, other than the described real-time video streaming scenario.

Following on from the scalability and performance analysis of the OpenFlow NAT system, it would be an interesting area of study to compare the proposed system with the previously mentioned distributed NAT system presented by Yeh and Chiu [218]. The comparison would help provide an insight into how an OpenFlow based system compares with distributed designs of standard NAT systems.

A final area of future work would be the full implementation of all the presented components of the OpenFlow CDN system, as at its current stage, each aspect of the system has only been tested individually and not as a complete system. Implementing the complete OpenFlow CDN would not be an easy task, as it would require in-depth development of control applications, or development of controller modifications like the previously described Floodlight modules; this is in addition to the large costs that would likely be involved in order to obtain the necessary equipment.

8.6.5 Max-Flow Traffic Optimisation

An area of future work for the work presented in Chapter 7, is the investigation of the proposed algorithm in delay focused scenarios. The modified algorithm would focus on not only reducing blocking in the network to meet all demands, but also reducing the path length of given solutions. To ensure path lengths of traffic were maintained at a desired value, the presented algorithm would need to be modified, as it does not currently consider this metric during its optimisation process. It would be a simple enough task to extend the presented algorithm to include path length as an optimisation parameter, forcing the current solution search to discard solutions with higher than desired path length. However, the search process would obviously need to be more complex, with the consideration that it may be necessary to extend certain paths beyond the desired lengths to meet demands. The addition of this path length optimisation would be a worthwhile area of future work, especially for deployment in systems where minimal delay is required. It should be noted, however, that the described extension to the algorithm, would most likely degrade the original solution optimisation goal of maximising the free capacity on network links.

An additional area of interest which requires further investigation would be the analysis of how other ϵ values affect the results produced by the max-flow algorithm. Although brief testing of ϵ variation has been presented, a more in-depth study is required to provide a detailed analysis of how it affects each aspect of the solution process. The study would only require additional modelling using a wider range of values then previously used, in order to provide a more reliable indication of the ϵ value effects.

A final area of future work relating to the CPLEX comparison system, would be the improvement of its solution generation within non-blocking or underutilised network environments. As shown in Chapter 7, this would be an important improvement, as CPLEX produces an inefficient solution in relation to path length and network load. It does, however, achieve a solution faster

than the other presented algorithms, so there is a high probability that its solution process could be improved while still maintaining an adequate run time. Improving its solutions could involve either adding restrictions on demand path length as previously mentioned or adding additional optimisation criteria.

8.7 Collective Summary

With the increasing demand for UHD video content, it is reasonable to assume traffic within networks will also continue to increase. Furthermore, with video content already being one of the largest contributors to network traffic, conserving the capacity consumed by video content is becoming an increasingly important research area. Video content providers, in particular, are actively researching ways to reduce their traffic and financial costs, while maintaining or improving the quality of service for their users.

The results that have been presented can be seen to show how the use of additional content locations, or transcoder and application layer multicasting use, can be beneficial to reducing network load in a video delivery system. Furthermore, it can also be concluded from the presented research that utilising transcoders as application layer multicasting nodes, can provide a significant reduction in traffic along constrained links, such as transatlantic cables. The achieved reduction in traffic reduces the probability of blocking in the network, allowing further demands to be met. As previously mentioned, the reduction in traffic is essential to supply the increased demand for UHD content, particularly when considering real-time scenarios where using cached content is not possible.

Using a combination of the placement algorithm and the OpenFlow migration mechanism presented in chapters 4 and 5, a viable real-time UHD video distribution system can be offered to service providers. The proposed system would be ideal for the previously mentioned live showings of sporting events at cinema venues, due to the real-time viewing requirement. The proposed system would continue to optimise transcoding resources during the transmission, relocating transcoding resources near seamlessly as additional venues joined the stream. The relocation event could be decided using the presented heuristic algorithm or based on financial cost, as some data centres provide cheaper compute resources during certain periods of the day. Once the optimised position of transcoder resources has been determined, the presented OpenFlow migration system would be utilised to relocate transcoders without significantly affecting the clients viewing experience. It is important to note that current systems allow this optimisation to be performed before the stream has begun, however, the presented system allows for optimisation and migration frequently throughout content transmission. The ability to migrate resources during live transmission provides improved efficiency, by enabling dynamic positioning of resources according to current demands; this is in contrast to the existing static systems that place resources based on predicted demand calculations before the transmission. It should be highlighted that although the system does aim to reduce network traffic, during the short period of migration, network traffic increases briefly due to the duplicated streams. In the currently proposed system in Chapter 5, the duplicated streams are only active for 4 seconds, which in comparison to the length of a live sporting event such as a football match is relatively insignificant. However, the concept of dynamically moving resources multiple times throughout transmission needs to be considerate to

this small traffic increase, as the frequency of migration events could inflate the short duration of duplicated streams to a point where the benefits of the system, are being mitigated by the migration mechanism itself. The frequency at which the optimisation takes place would need to be carefully chosen based on the current streaming scenario and network characteristics.

Other scenarios that this combination of transcoder placement and migration mechanism can help improve, includes the live streaming applications described earlier in Section 5.2, which are currently rising in popularity. Additionally, as previously described, the proposed migration system can also be applied to the concept of migrating application layer multicasting nodes, in order to provide a viable content delivery system when scalable video coding is being utilised.

Following on from the transcoder migration system, the OpenFlow smart CDN addresses several other usage scenarios that are increasing in demand. One of these scenarios includes the use of the mobile and game live streaming applications described in Section 2.8, which are currently showing large scale adoption. The proposed OpenFlow CDN enables these types of applications to live stream their content, without the current 10-20 second delay which is seen with HLS and MPEG-DASH based implementations. The reduction in delay provides improved interaction functionality for both users and publishers, such as the reduction in delay from a text based chat request causing an effect in the live stream. In addition to the delay improvements, the proposed OpenFlow CDN would also be able to serve multiple applications simultaneously, as it is capable of distributing stored video content in parallel with real-time content. Furthermore, the proposed OpenFlow CDN also has the benefit of efficient utilisation of resources, which is achieved using the OpenFlow NAT system to optimise server selection for incoming requests. A further benefit of the OpenFlow NAT system is that it has the additional capability of routing traffic through an edge node closest to the requesting client; this can improve the client viewing experience and also minimise any Internet infrastructure related issues.

By utilising the presented max-flow algorithm within the proposed OpenFlow CDN, the system should scale beyond existing CDN systems, while also providing an improved level of service. The improved scalability is in part due to the presented switch based multicasting system, but is enhanced further by reducing the probability of blocking within the CDN using the max-flow traffic optimisation. To confirm the scalability assessment, a direct comparison between the proposed system and existing mechanisms will need to be performed, in order to closely determine any limits or bottlenecks in the system; performance analysis of the proposed OpenFlow CDN is left as future work and is described in more detail in Section 8.6.4.

There are a number of novel contributions that have been highlighted throughout the presented research and conclusions, these include:

- A novel heuristic algorithm to provide optimised transcoder placement in a given network, with characteristics that enable online dynamic optimisation.
- A novel migration mechanism to provide a near seamless live migration of transcoders, while streaming UHD content.
- A novel OpenFlow smart CDN system, capable of providing improved delivery of both real-time and stored video content to large volumes of users. The OpenFlow CDN includes a number of novel aspects, including:

  - ◇ A novel OpenFlow based NAT system that bridges the private CDN with public networks, in order to track, manage and optimise server selection and content requests.
  - ◇ A novel switch level OpenFlow multicasting mechanism for use with distributing real-time streaming video.
  - ◇ A novel alternative to DNS redirection using OpenFlow, which improves on existing DNS redirection techniques. The system enables access to replicated content found within the network while routing content requests to the CDN entry point closest to the client.

- A novel max-flow algorithm, that is capable of efficiently routing traffic throughout a network, such as the proposed OpenFlow CDN. The presented algorithm minimises the average link utilisation throughout the network, in order to reduce the probability of blocking.

The systems presented in this thesis aim to provide proof of concept designs, which could be utilised in a production system with some additional work; most of the additional work would be focussed on implementing the migration control software within an OpenFlow controller such as Floodlight, as this would consolidate the system into a single application that could be deployed in an OpenFlow environment. Further analysis of the scalability would still need to be performed, however, it is envisioned the systems would scale to a larger scale without major issues based on initial results. This would then allow content providers to utilise the presented systems to improve the efficiency of their content delivery platforms; the improvements would provide a reduction in both cost and resource requirements, while also maintaining or improving the quality of service to clients.

Appendix A

Migration System Development

This appendix will contain details of the many proof of concept prototypes that were crucial in the development of the final migration system presented in Chapter 5. Each test phase will detail the limitations of its naive design, as well as the improvements that were required to overcome issues recognised at that stage of development. Additionally, a number of technologies and protocols that were initially implemented within the migration system will be discussed; however, after further development, these were seen to be unsuitable for the final UHD migration system.

As previously described in Chapter 5, the Floodlight controller was used in all prototype systems to provide standard switching functionality to the OpenFlow switches.

A.1 Initial Testing Using Software Platforms

To test and prove the concept of migrating content streams from one location to another, a simple test bed system was implemented using VMs on a single host machine. The VMs were configured using Oracle VirtualBox virtualisation software, along with OVS to connect the machines with an OpenFlow implementation.

Figure A.1 provides an overview of the virtualised system, including switches and computers. In this system, the two relay machines are given the same IP address, as the relay copy is envisaged as a migrated copy of the original relay; furthermore, it should be noted that the two relays are not used simultaneously during normal operation, only one of them is switched on and active, the other is in a paused state. However, it is important to highlight that versions of this system discussed later, include a brief time during the migration process that both transcoders are being used; this would normally cause IP address conflict issues given enough time, but blocks put in place using OpenFlow are able to avoid this issue. The link between the two virtual OVS bridges can be visualised as a constrained link, such as a transatlantic link between the UK and US; it is on this link that we would be trying to reduce congestion in a fully working example.

Initial testing involved the development of three basic Java applications:

1. The first application acted as a server and provided a simple packet transmission operation, sending a repeating message using a UDP based socket when requested. The message itself was just a text based message encapsulated in a custom container, which provided header information with the requested IP address and Port information of the requesting client.

2. The second application operated as a relay for the text based message sent from the first application. Its operation was to listen on a UDP socket for incoming messages and then once received, it would examine the custom header information from the message to obtain the relevant client IP address and port that would be required to forward the message. The message was then stripped of its custom encapsulation and then sent to the third application using the gathered IP address and port information using another UDP socket.

3. The third application acted as a client application and allowed a request to be made to the first application to send messages. It would then display any message it received on its UDP socket to the user.

These applications were then placed on the relevant VMs that had been set up.

- Application 1 on the server machine.
- Application 2 on both the relay and the relay copy machines.
- Application 3 on the client machine.

The relay application was designed to simulate the functionality of a transcoder machine, as a transcoder would both receive the input from the server and then forward it to the client once any content adaptation was performed; it is important to highlight the fact that this is also displaying the basic functionality of application layer multicasting, which can be applied to the proposed system.

The rate at which the server application sent messages could be configured so that migration performance could be crudely measured. It was initially set to 1 message/second, but this was

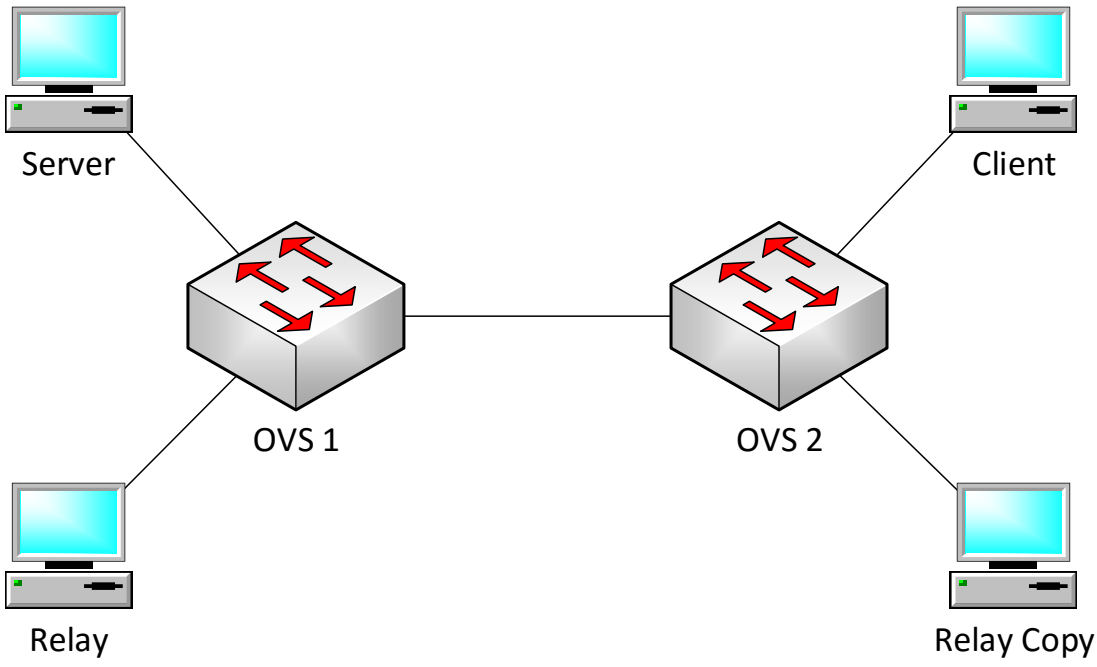


Figure A.1: Initial testbed architecture.

increased to 10 messages/second after successful operation was confirmed. The initial slow message rate was required to provide an easy visual confirmation that the system was working, as faster message rates proved difficult to view during migration.

Figure A.2 provides an insight into the system operation for the test scenario, including the processes involved during the migration; however, the timings are not correctly represented and is only intended to give a more detailed view of the stages during migration.

Direct system calls were used to control and insert flows into the OVS bridges during the migration process. For the migration to occur successfully, the relay copy was started and then the following flow rules were added to the OVS instances:

- Any traffic arriving on OVS 1 destined for the IP address of the relay was to be forwarded across the link to OVS 2.
- Any traffic arriving on OVS 2 destined for the IP address of the relay should have its

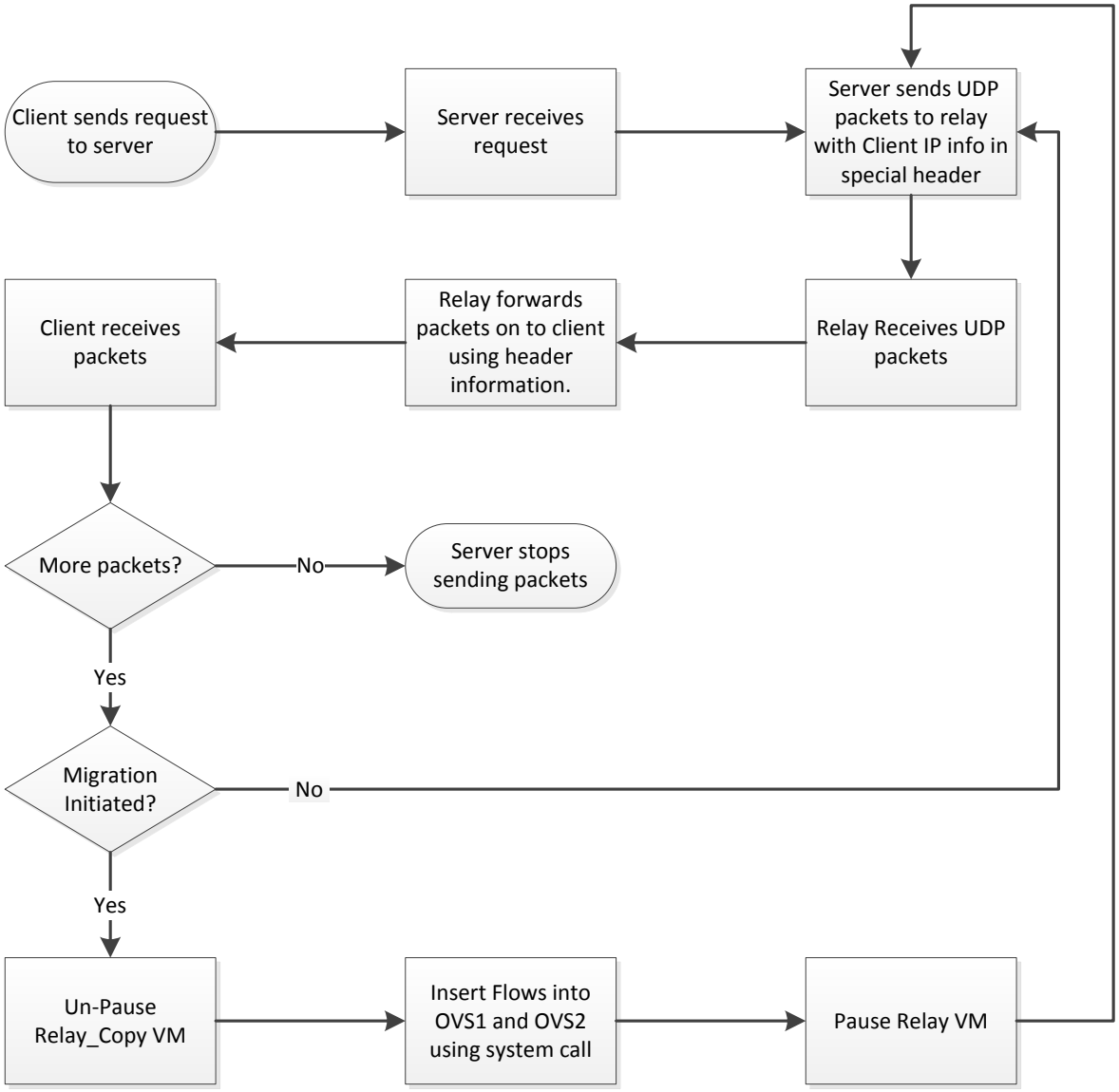


Figure A.2: A simplified view of events for the test setup.

destination MAC modified to read the MAC of the relay copy, this should then be forwarded out the port leading to the relay copy.

After these flow rules had been successfully input into the switches, the initial relay machine was placed into a paused state.

Using these simple OpenFlow rules inserted into the OVS instances, the packets destined for the relay were redirected to the relay copy machine; this mechanism occurs completely in the network, making its operation transparent to the server and client machines.

From initial testing of this system, it became clear that migrating the relay during transmission was not only possible, but also efficient, as no messages were lost during transmissions at a rate of 10 messages/second. This is in contrast to a standard networking scenario, which utilised standard networking functionality to recover after disconnecting one relay and then connecting another. In the standard networking scenario, a significant number of messages were lost during the switchover of the relays, although it did eventually recover given enough time.

This initial testing provided confirmation that the theorised migration system could be achieved using OpenFlow. However, to further investigate the feasibility of the proposed migration system, testing using real world video traffic was required; the next prototype system described in the next section addressed this deficiency.

A.2 VM Machine Migration with SD Content

This section implements a system architecture similar to the one presented in Section A.1; however, this system improves on the previous implementation, by using actual video traffic in place of the application generated content previously used.

FFmpeg [15] was utilised in this system in place of the relays to provide the transcoding functionality; while VLC player was chosen to perform both the initial streaming of the content at the server, as well as the receiving software at the client. As stated in Section 2.1.2, FFmpeg is an ideal candidate for providing transcoding functionality, as it has a diverse feature set and can both receive and send video streams through UDP socket connections. VLC was chosen for similar reasons, with its high customisability and ability to send and receive UDP content. It should be mentioned that a later prototype of the system presented in Section A.3, concluded that it was necessary to replace VLC as the initial streaming application with FFmpeg; this was due to compatibility and configuration issues relating to working with the FFmpeg transcoders. However, VLC worked correctly in this prototype system, showing that it may be a viable application in certain scenarios.

For this testing scenario, standard definition video content was used, due to resource constraints caused by running OVS and VM resources on a single host machine. This was in part due to the initial prototype system being built on a single machine, with that system being adapted and used for this implementation. Another complication that necessitated the use of a single machine was the issue of connecting the VMs to the OVS instances. This was a problem as at this stage in development, as research into possible connection technologies such as GRE tunnelling had not fully been completed; however, tunnelling technologies are utilised and described in some of the later prototypes.

As previously stated, the system was set up and configured as in Section A.1, with the described modifications. Other characteristics of the system remained unchanged, such as the process of migration with pausing and starting the VMs, as well as the flow rules that were inserted during the migration.

One of the things to be considered with this prototype system is the resources required to run both OVS instances and all the VMs hosting the transcoding and streaming software. This placed a considerable load on the host system, which at this stage was a high end laptop. Due to this, the performance of the system was constrained, causing the video stream to lag and pause at times. The unstable playback made collecting reliable quantitative results a difficult task, so this testing stage only served to provide a limited feasibility assessment. However, the testing was inherently successful with the video being streamed to the client and the migration successfully migrating the stream to the copied transcoder; the only issue was that both the streaming and migration tasks took longer than expected to complete.

One factor that initially seemed solely due to the lack of available resources, was the slow analysis time of FFmpeg on the copied transcoder; this meant a large delay between the copied transcoder receiving the content stream and FFmpeg starting the transcoding process. After analysis of this issue, it was found that although it was affected by the lack of resources, it was more significantly influenced by the analysis time of the video stream configured within FFmpeg. This initial delay can be reduced to a minimum, by optimising the analysis stage of FFmpeg using the "probesize" command line switch. It should be noted that setting this configuration lower than the default value can often cause transcoding issues, due to the stream being incorrectly identified; therefore, the "probesize" has to be either left at its default level or carefully chosen based on the content that it is set to be transcoded.

This prototype system provided an important insight into the resource requirements for even basic standard definition content, as well as invaluable knowledge of FFmpeg configuration parameters that would be required for future versions. It is clear that to produce a system that will provide real-time transcoding of even HD content will require substantial compute resources. To achieve this, the VMs need to either all be placed on a dedicated VM server configured with considerable resources, or the VMs need to be positioned on different machines to distribute the resource load. The next section implements this idea of increased resources, using VMs located on multiple machines to present a system that is capable of migrating transcoders while transcoding HD content in real-time.

A.3 Improved VM Migration with HD content

After successful migration using the system presented in Section A.2, it was required to increase the available resources for each of the VMs; this would enable improved quality content transmission, as well as improvement to the migration process. To do this the VMs were split between multiple host machines, allowing increased performance with additional RAM and compute resources. Ideally, locating each VM on individual dedicated servers with a hardware based OpenFlow switch would be the best scenario, however, at this stage, these resources were currently unavailable. As a compromise to this, two high performance servers were utilised for the majority of the VMs, while the client machine was hosted on a separate machine to simplify viewing the video content.

The main objective of this prototype was to successfully perform a transcoder migration while streaming a single high definition video to a single client. If successful, additional concurrent clients could then be introduced for further testing. 5 VMs were initialised in total, this included 4 VMs to serve as standard desktop/server machines, along with a single VM to run OVS and serve as an OpenFlow switch; this OVS VM was then used to link each of the other VMs together using tunnelling to pass traffic through the network connecting the host machines, in an overlay network type of design. Although the Floodlight controller could be run from a remote location using out-of-band communication, in this case, it was run locally on the OVS VM for simplicity.

Figure A.3 provides a reference to the physical locations of the VMs on the two servers and the additional host machine. It should be noted that the switch shown in the figure is not an OpenFlow enabled switch, but just a standard layer 2 switch interconnecting the host machines running the VMs.

During initial testing of HD content within the system, streaming issues started to appear in the form of artefacts in the video, as well as pauses and stuttering during viewing. Through testing components of the system, it was determined to be caused by the initial encoding of the video within VLC. Unfortunately, the lack of encoding configuration available in VLC prevented these issues from being resolved, which made VLC unsuitable for the initial streaming application. Therefore, the initial encoding and streaming tasks were now configured to be performed by FFmpeg, as it provided a significant improvement in configuration options. Additionally, using FFmpeg throughout the streaming process up until it is received at the client seemed to provide improved image quality and reduced bitrates; this can be attributed to being able to configure the quality and codec options in finer detail when using FFmpeg. The use of FFmpeg throughout the system also made setup and configuration of transcoders more efficient, as settings could be copied and used uniformly across VMs.

One important characteristic that needs to be taken into consideration when setting up the VMs for the prototype, is how they are to be connected together. This can be achieved very easily

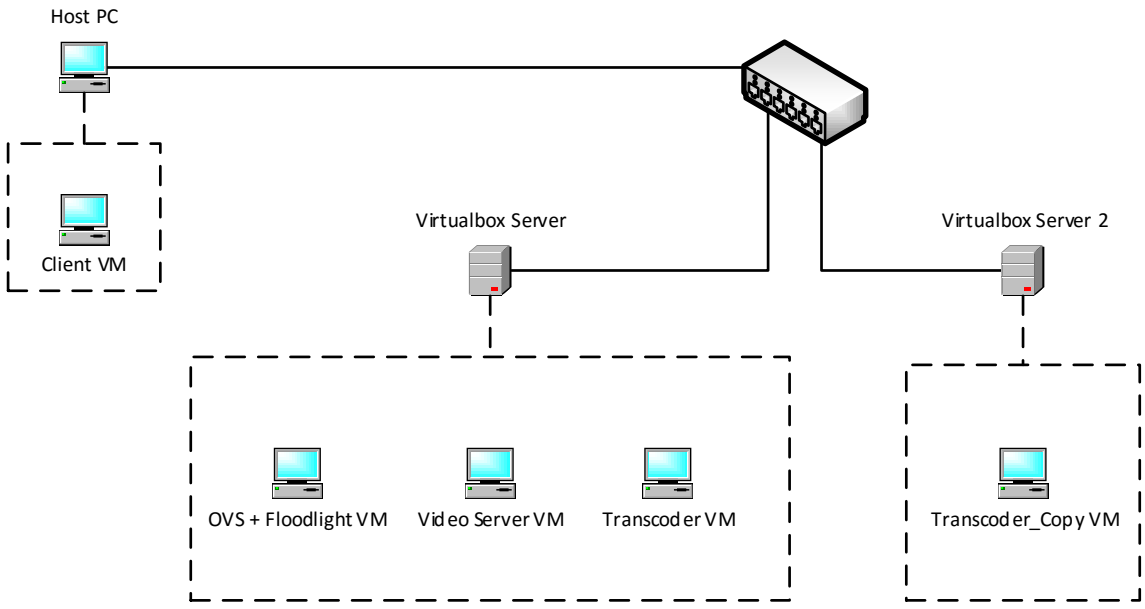


Figure A.3: VM locations on the physical network.

when using dedicated hardware with a physical OpenFlow switch, however, when using virtualised resources as they are in this case, it becomes a complex issue. It would be possible to connect all the machines together using physical cables if the host machines had a sufficient number of physical network interfaces. This would be achieved by mapping VMs to dedicated ports on the host, which could then connect to the OVS VM with multiple network ports assigned. Although this would give the best performance as it requires no overhead for the packets, it is not feasible for the host machines to be fitted with so many network interfaces. To overcome this issue, tunnels are required to provide an overlay network to connect each VM with the VM running OVS. The main thing to consider when connecting the VMs using a tunnelling protocol is that OVS is a link layer networking device; this requires the Ethernet headers to be transported along with the packets being sent from one VM to another. This makes selecting an appropriate tunnelling protocol an important step, as most tunnelling protocols work on the network layer, stripping off any Ethernet information before sending the packets through the tunnel; these network layer tunnelling protocols would then only use IP information for routing packets to their destination. Some tunnelling protocols which were investigated but found to be unsuitable were: standard GRE [240, 241], PPTP [242] (Uses a modified version of GRE), IPSec [243] and L2TP [244]. One suitable tunnelling protocol, however, is the GRETAP interface on Linux, which although similar to standard GRE, has a major difference in that it encapsulates and forwards the entire packet including the Ethernet frame. This allows packets from VMs to be tunnelled over the host network to the OVS VM, while ensuring OVS can correctly interpret and read the VM Ethernet header information of the packets when they are received. It should be noted that GRETAP tunnels are setup differently to standard GRE tunnels, requiring the use of the `ip link` command within Linux to be configured. OpenVPN [245, 246] is also able to achieve Ethernet tunnelling, but requires specific non-default configuration to achieve this; this would make configuration unnecessarily complex when compared with using GRETAP interfaces.

Although OVS has GRETAP support built into its link setup options, it was determined that it would be more suitable to set up the tunnels using standard Linux port configuration outside of OVS; these virtual Linux interfaces were then added to OVS using the standard add-port configuration command. The reason for configuring the ports in this way was due to restrictions that are placed on GRE ports created with OVS, as well as for simplicity of configuration. One of the main restrictions on OVS interfaces that would have caused significant inconvenience, is the fact they can not be monitored with network analysis software; this is due to the OVS interfaces being hidden from the Operating System (OS) and normal use. In order to monitor traffic on these OVS configured interfaces/tunnels, it would require monitoring the physical interface bonded to OVS and then filtering individual tunnel traffic. In contrast to this, using standard Linux virtualised interfaces allows packet inspection software such as Wireshark [152], to monitor packets from each individual tunnel interface separately.

It is important that migration events occur at specific times, so to automate the migration process, shell scripts were implemented to perform the required tasks. These tasks included starting and pausing the VMs, as well as inserting flow rules into specific OVS bridges. Optimising the timing of these events was crucial in achieving improved migration quality, which in effect meant reducing the switchover time between the old and new streams arriving at the client. To

achieve the automation in a shell script, password-less SSH was configured between the control machine, the VMs and also the host servers; this enabled remote system commands to be sent without user interaction. It is important to note that the timings between migration events mentioned in this section will vary based on a number of factors; these include the speed at which VMs can be paused and resumed, as well as the resources available to the VMs. These resources can include compute power, memory constraints and also the storage medium speed of the initial streaming server; all the stated resources can affect migration delays, due to them affecting the speed at which a VM can send/receive the content. The specific timings that were found to be optimal for the current implementation can be seen in Figure A.4; with the 10 second delay allowing the transcoder_copy VM to resume and bring up its network interfaces, and then the 5 second delay allowing initial analysis and transcoding to begin at the new transcoder.

The flow rules that are inserted during migration are improvements over the previous test environments. In this implementation, two rules were configured on each virtual switch to redirect packets to the new transcoder; one of these rules was inserted to redirect ARP requests and the other was configured to redirect IP packets. The flow rules inserted into the switches during migration can be seen in Appendix C.1. The flow rules configured use a very fine granularity when depicting the flows in this scenario, but could be expanded to encompass whole IP topologies rather than single transcoder machines. This could allow the migration of entire pools of transcoders in a single instance, rather than in several operations. Restrictions were also put in place using the flow rules to allow the initial transcoder to complete the transmission of any buffered content it was still processing once the migration had begun; this required a small delay before pausing the original transcoder VM after inserting the flow rules. A small delay after resuming the Transcoder_copy VM is also necessary to allow the VM to re-initialise its state and bring up the network interfaces.

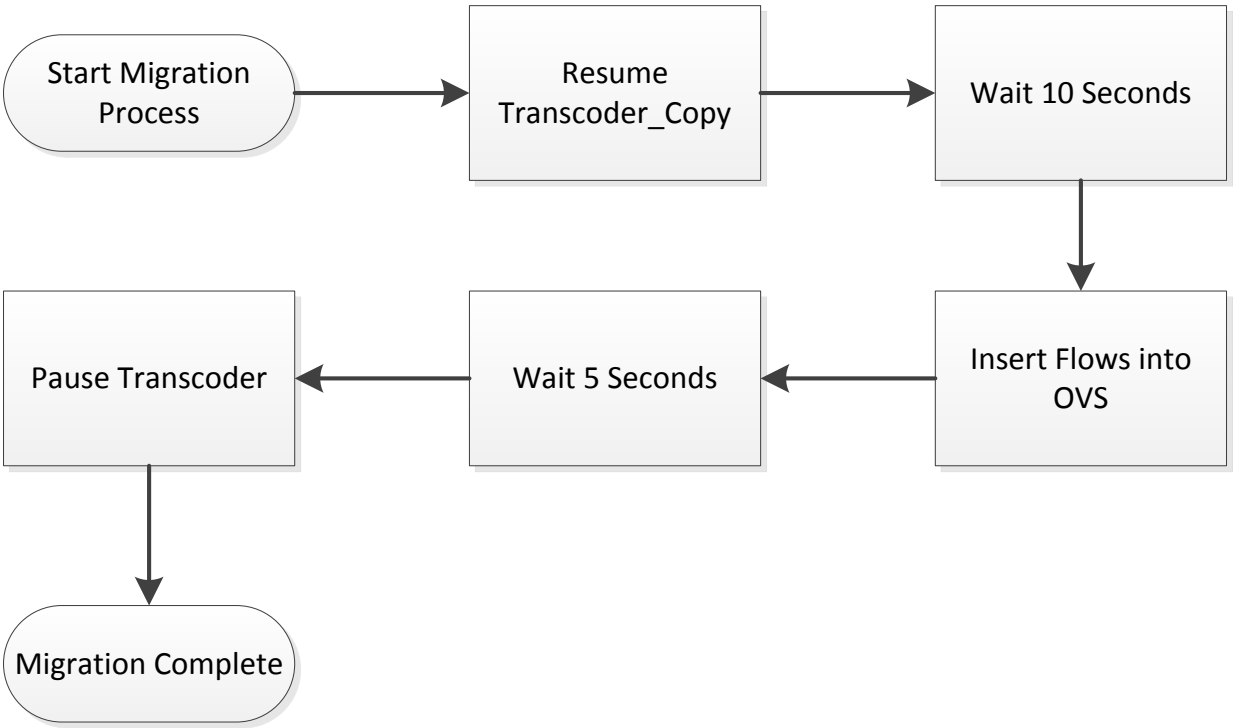


Figure A.4: Optimal timings for the migration process.

The switchover of transcoders was monitored using Wireshark, which is a packet capturing software described earlier in this section; it was configured to monitor the client VM, as this would be the best placement to detect the switchover point. This switchover point can be determined by searching for a change in source MAC address in the UDP video traffic packets, which would signify the change in transcoding machine. As seen in Figure A.5, the packet switchover occurs at 10.95 seconds into the capture and is complete by 11.05 seconds, this shows a relatively small switchover time of 0.1s; this is especially small in comparison to the 19.86 seconds it takes the standard network migration to occur, which can be seen at 251.78 seconds in Figure A.6. It should be noted that this was the best case result for the OpenFlow migration that was observed at this stage of development. Additional attempts observed switchover times between 0.1 and 3 seconds for the OpenFlow aided migrations. Migration times for the normal network migrations were consistently above 15 seconds, with some taking far longer to recover the stream. If it is possible to consistently maintain the fast OpenFlow aided switchover, it should significantly reduce interruption of service for the clients streaming real-time content. In its current form, it may be possible to further reduce this delay with optimisation of timings in the automation scripts, FFmpeg options, or the use of more powerful hardware to run the VMs; however, this is not described further, as development was moved on to the final 4k version of the migration system, where additional improvements were achieved.

Time	SRC MAC	Source	Destination	Protocol	Info
10.935679	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.935766	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.953289	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=8706) [Reassembled in #5146]
10.953300	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.953393	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=8707) [Reassembled in #5148]
10.953398	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.953404	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=8708) [Reassembled in #5150]
10.953408	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.954328	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=8709) [Reassembled in #5152]
10.954497	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.954705	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=870a) [Reassembled in #5154]
10.954711	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.954718	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=870b) [Reassembled in #5156]
10.954723	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
10.954728	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
11.058137	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7bdc) [Reassembled in #5159]
11.058320	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
11.058587	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7bdd) [Reassembled in #5161]
11.058594	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	SIGCOMP	Msg format 1
11.058601	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7bde) [Reassembled in #5163]
11.058606	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
11.058612	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7bdf) [Reassembled in #5165]
11.058616	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
11.058622	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7be0) [Reassembled in #5167]
11.058626	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
11.058631	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=7be1) [Reassembled in #5169]

Figure A.5: Wireshark packet capture during OpenFlow aided migration.

Time	SRC MAC	Source	Destination	Protocol	Info
251.785008	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785021	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=36a2) [Reassembled in #64585]
251.785136	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785143	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=36a3) [Reassembled in #64587]
251.785147	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785153	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=36a4) [Reassembled in #64589]
251.785157	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785256	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=36a5) [Reassembled in #64591]
251.785262	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785269	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=36a6) [Reassembled in #64593]
251.785273	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	UDP	Source port: 58140 Destination port: 6666
251.785728	f2:65:90:62:6c:fc	192.168.1.3	192.168.1.6	SIGCOMP	Msg format 1
255.198843	92:8f:b2:9e:71:da	92:8f:b2:9e:71:da	LLDP_Multicast	LLDP	Chassis Id = 8e:d0:1d:70:61:42 Port Id = TTL = 120
255.198850	92:8f:b2:9e:71:da	92:8f:b2:9e:71:da	Broadcast	0x8999	Ethernet II
270.114704	92:8f:b2:9e:71:da	92:8f:b2:9e:71:da	LLDP_Multicast	LLDP	Chassis Id = 8e:d0:1d:70:61:42 Port Id = TTL = 120
270.114710	92:8f:b2:9e:71:da	92:8f:b2:9e:71:da	Broadcast	0x8999	Ethernet II
271.647236	da:e4:54:57:b4:e1	da:e4:54:57:b4:e1	Broadcast	ARP	Who has 192.168.1.3? Tell 192.168.1.2
271.647248	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=03c4) [Reassembled in #64683]
271.648323	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=03cb) [Reassembled in #64602]
271.648461	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
271.648957	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
271.649347	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=03cc) [Reassembled in #64605]
271.650360	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
271.650823	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=03cd) [Reassembled in #64607]
271.650831	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666
271.650840	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	IPv4	Fragmented IP protocol (proto=UDP 17, off=0, ID=03ce) [Reassembled in #64609]
271.650845	06:35:a6:64:e2:eb	192.168.1.3	192.168.1.6	UDP	Source port: 44128 Destination port: 6666

Figure A.6: Wireshark packet capture during normal migration.

The focus on this prototype system was to investigate, in more detail, the switchover time when streaming HD content; this was performed successfully to a certain degree, providing significant results. However, it should be highlighted that the quality of video had to be severely reduced to achieve real-time transcoding. This was due to VM resource constraints, as well as network capacity related issues that were caused by routing all traffic in and out of the VM containing OVS. This placed a considerable load on not only CPU resources handling the network traffic, but also the single link that connected the host server with the other server and client hosting machine. The viewing experience for the client was also difficult to analyse, due to it being run on a VM; because of this, it was difficult to interpret whether any display issues that were observed were due to the streaming and migration system, or the VM display mechanism, which at times did cause artefacts and delayed refresh rates. Another scenario that was not possible, was the introduction of additional clients with varying resolution, which was due to the server resources being constrained by the initial system, with no room to scale demands. It was clear that the system could be further improved by transferring the system to physical machines, in order to increase their available resources; with improved resources and optimisation of the migration process and timings, it would be possible to provide a more reliable service and provide a clearer view on how migration affects the viewing experience of the client.

Appendix B

Migration Machine Specification

B.1 Local Testing with NetEm Machine Specifications

- Server: Intel(R) Core(TM) i7-4960X CPU @ 3.60GHz 66GB RAM
- Transcoder1: Intel(R) Core(TM)2 Duo CPU E8400 @ 3.00GHz 4GB RAM
- Transcoder2: Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz 6GB RAM
- Client: Intel(R) Core(TM) i7 CPU 965 @ 3.20GHz 6GB RAM

B.2 Transatlantic Testbed Machine Specifications

- Server: 2 x Intel(R) Xeon(R) CPU E5620 @ 2.40GHz 16GB RAM
- Transcoder1: 2 x Intel(R) Xeon(R) CPU E5620 @ 2.40GHz 16GB RAM
- Transcoder2: Intel(R) Xeon(R) CPU E3-1270 V2 @ 3.50GHz 32GB RAM
- Client: Intel(R) Xeon(R) CPU E5520 @ 2.27GHz 3GB RAM

Appendix C

Controlling Flows For Migration

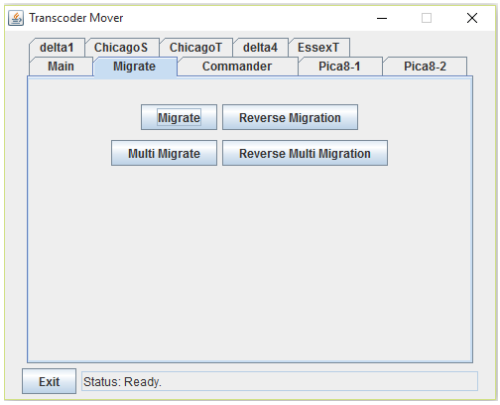
C.1 Basic Flow Rules

Switch	Flow Rule
SW1	in_port=serverPORT,dl_type=0x0800,dl_dst=transcoder1MAC,nw_src=serverIP,nw_dst=transcoderIP actions=output:sw1LinkPORT
SW1	in_port=serverPORT,dl_type=0x0806,nw_src=serverIP,nw_dst=transcoderIP actions=output:sw1LinkPORT
SW2	in_port=sw2LinkPORT,dl_type=0x0800,dl_dst=transcoder1MAC,nw_src=serverIP,nw_dst=transcoderIP actions=mod_dl_dst:transcoder2MAC,output:transcoder2PORT
SW2	in_port=sw2LinkPORT,dl_type=0x0806,nw_src=serverIP,nw_dst=transcoderIP actions=output:transcoder2PORT

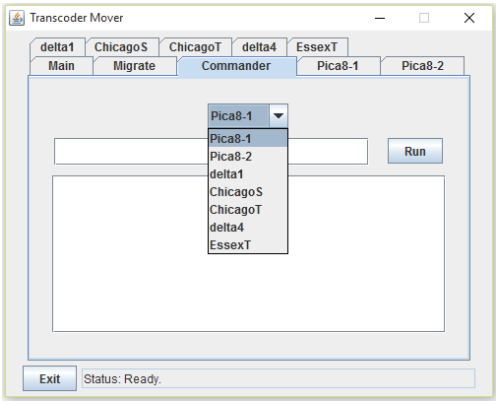
Table C.1: Flow Rule details for the basic migration process.

C.2 Migration Control Application

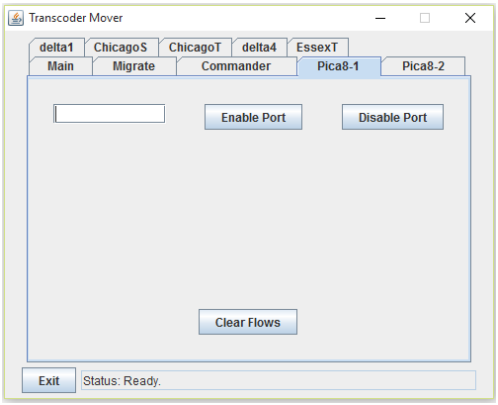
The custom controller application can be seen in Figure C.1, where tabbed panes allow access to a number of machines and functions. The main focus of the system is the migrate pane shown in Figure C.1, which presents the simple button interface to either initiate or reverse the migration process. The commander pane in Figure C.1 allows custom commands to be run on any of the devices being utilised within the system, which is useful for diagnostic purposes. Other panes control aspects of their respective devices, including: initiating transcoder software on the transcoder machines, running client viewing software on the client, initiating the video stream on the streaming server and also allowing current flow data/rules to be cleared from the OpenFlow switches.



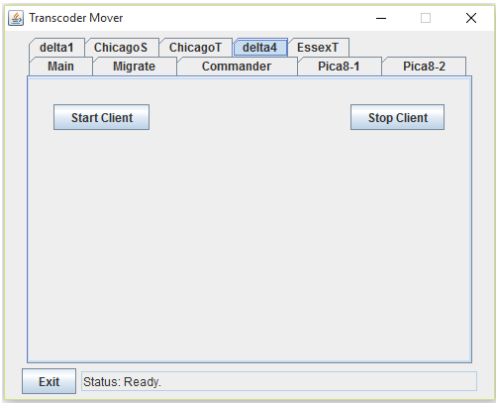
Migration Page



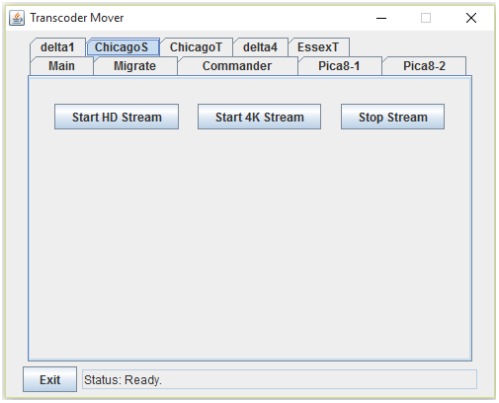
Commander Page



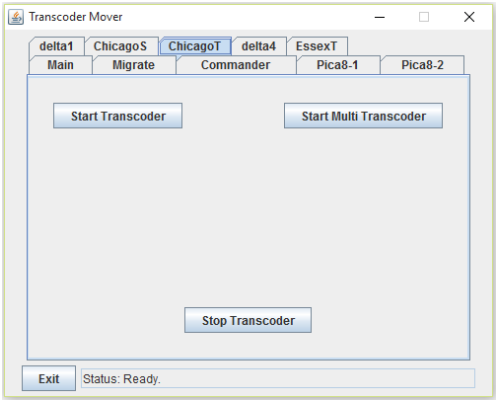
Pica8 Page



Client Page



Server Page



Transcoder Page

Figure C.1: Custom control software used for the OpenFlow migration system.

C.3 Advanced System Flow Rules

No.	Flow Rule
1	cookie=9998,in_port=serverPORT,dl_type=0x0800,nw_src=serverIP,nw_dst=transcoderIP actions=output:sw1LinkPORT,mod_dl_dst:transcoder1MAC,output:transcoder1PORT
2	cookie=9999,in_port=serverPORT,dl_type=0x0806,nw_src=serverIP,nw_dst=transcoderIP actions=output:sw1LinkPORT
3	cookie=9999,in_port=sw2LinkPORT,dl_type=0x0800,dl_dst=transcoder1MAC,nw_src=serverIP,nw_dst=transcoderIP actions=mod_dl_dst:transcoder2MAC,output:transcoder2PORT
4	cookie=9999,in_port=sw2LinkPORT,dl_type=0x0800,dl_dst=transcoder2MAC,nw_src=serverIP,nw_dst=transcoderIP actions=output:transcoder2PORT
5	cookie=9999,in_port=sw2LinkPORT,dl_type=0x0806,nw_src=serverIP,nw_dst=transcoderIP actions=output:transcoder2PORT
6	cookie=9997,in_port=transcoder2PORT,dl_type=0x0800,nw_src=transcoderIP,nw_dst=clientIP actions=
7	cookie=9999,in_port=transcoder2PORT,dl_type=0x0800,nw_src=transcoderIP,nw_dst=clientIP actions=output:clientPORT
8	del-flows sw2Name cookie=9997/-1
9	cookie=9999,in_port=serverPORT,dl_type=0x0800,nw_src=serverIP,nw_dst=transcoderIP actions=output:sw1LinkPORT
10	del-flows sw1Name cookie=9998/-1

Table C.2: Flow Rule details for the optimised final migration process.

Appendix D

OpenFlow CDN Additions

D.1 OpenFlow CDN Security Advancements

Further to previous discussions of improved security, the proposed OpenFlow CDN system would be able to tailor security aspects based on specific applications; this could be as simple as restricting access to servers, or masking whole networks from view depending on client authorisation. This type of control could be implemented in a number of ways. One less secure method would be to use the source IP address of requests as an identifier; however, it is well known that this can be easily spoofed [247, 248], so this type of authentication would not be used without additional security measures. An additional security measure that could compliment the IP address identifier is the use of an authentication server, which would communicate with the OpenFlow controller. A user would then be required to log in to the authentication server using a key based authenticator or a standard username and password method. Once the user has authenticated with the server, traffic could then be redirected using the controller to secure areas of the network; further security challenges could then be made, such as requests for a security token gained from the initial authentication process.

Using a security system such as the one described would provide improved security for both the network and connected devices. This improved security is achieved using the flexibility of OpenFlow to quickly modify rules within the network, allowing certain parts of the internal network to remain hidden, even from internal machines which may be compromised. This type of functionality would usually be achieved using a VLAN solution in a standard CDN scenario, but this would limit the flexibility of the system and would require extensive configuration both initially as well as during operation. With the OpenFlow controller communicating with the authentication server, full network access can be granted or removed almost instantly.

With the rapid and flexible traffic control available with OpenFlow, it would also be beneficial to investigate the possibilities of developing network threat detection into the controller. With a complete overview of the CDN topology, machine utilisation and network traffic, threat detection capabilities should be able to become more advanced; this includes providing improved detection of not only Distributed Denial of Service (DDoS) type attacks, but also internal malware infections and malicious activity. This would be an improvement over existing network monitoring solutions, which rely on taps into the network at several locations to perform these types of tasks; however, even with several taps into the network traffic flow, these types of applications would still only

have access to a small subset of the information available to the OpenFlow controller. With the added wealth of information that the controller has access to, in theory, it should allow better detection of malicious activity. This is not investigated further in this thesis, as it is discussed to demonstrate the possibilities for future work with the proposed system.

Although the security policy enforcement discussed in this section is not discussed further, it is important to note that the capabilities of OpenFlow required for this system to operate have been investigated in this Chapter 6; with further work into this area, it is feasible that a system such as the one proposed could be implemented with these types of security mechanisms.

Appendix E

Max-flow Optimisation

E.1 Multicommodity Flow with Minimum Congestion (MFMC) Description

The MFMC problem has been described in various existing literature, however, Reed describes the MFMC problem in combination with the concept of applying it to routing traffic through a network graph. Section E.1.1 goes on to describe the algorithm presented by Reed, using descriptions taken from Reed's presented work, as well as explanations presented by Garg and Konemann [249] and Fleischer [250].

E.1.1 Solving the MFMC

As stated in Chapter 7, the MFMC is a linear programming problem that has not been directly solved in previous literature, however, it is possible to transform the MFMC problem into a Maximum Concurrent Flow (MCF) problem.

The MCF objective is to at least route θd of each demand d, using a set of flows $y \in Y$. This maximisation problem can be stated as:

$$\max(\theta) \quad \text{s.t.}$$

$$\sum_{P_e : e \in p} y(p) \leq u(e) \quad \forall\, e \in E \tag{E.1}$$

$$\sum_{p \in P_d} y(p) \geq \theta d \quad \forall\, d \in D \tag{E.2}$$

where the variables are as defined within the MFMC problem in Section 7.1.1.

From observation of the MFMC and MCF problems, it can be seen that the solution of the MCF problem is equivalent to that of the MFMC problem, i.e., $\Omega_L = \theta$. Furthermore, dividing through the MCF problem by θ, shows that the optimum value θ' for all y in the MCF problem, is equivalent to the solution of the MFMC problem with optimum value Ω'_L for all x, with a relationship given by $1/\theta' = 1 - \Omega'_L$ and $x(p) = y(p)/\theta'$. The MCF (and therefore MFMC) problem can be defined as a linear problem, which has an optimum solution that can

be calculated in polynomial time. However, in all but trivial network scenarios, the number of paths $|P_a|$ can increase rapidly as the network size increases; this can significantly increase solution complexity and, therefore, also increases the calculation time when using generic linear optimisation algorithms. However, fully polynomial-time approximation schemes (FPTAS) for the MCF are available and are more efficient at providing a solution. Using the algorithm introduced by Garg and Konemann [249] combined with the improvements presented by Fleischer [250], it is possible to approximately solve the MCF problem. More precisely, the algorithm is able to calculate an ϵ-optimal solution, θ_ϵ, where $\theta' \geq \theta_\epsilon \geq \theta'(1-\epsilon)^{-3}$, and where $0 < \epsilon < 1$. Garg and Konemann [249] express the algorithm in very general terms. Here, the algorithm nomenclature is re-expressed using the specific case of a network optimisation scenario, using symbols that are compatible with the other described algorithms presented throughout this thesis.

The algorithm presented by Garg and Konemann [249] optimises the dual problem of the MCF, with lengths $l(e)$ being associated to each edge and the dual linear problem focusing on the objective of minimising the dual variable β. The algorithm uses phases and steps during its operation, such that the lengths at the nth step, written as $l_n(e)$, are continued to be updated until the optimum β value is achieved. A minimum-cost path $p_m(d)$ for demand d, according to the lengths l, is denoted by a shortest path function $p_m(d) = F(l)$ with the cost of this path being derived by $C(p_m(d))$. Dijkstra's algorithm is used in practice to calculate shortest path F. Garg and Konemann show that [249]: the dual variable can be expanded as $\beta = \Delta(l)/\alpha(l)$, where $\Delta(l) = \sum_{e \in E} u(e)l(e)$ and $\alpha(l) = \sum_{d \in D} dC(p_m(d))$; and, that the algorithm converges if the initial lengths are set to $l_1(e) = \delta/u(e)$, where $\delta = (|E|/(1-\epsilon))^{-1/\epsilon}$. In each phase of the algorithm, steps can be described as the consideration of each demand d. For each step within a phase, no more than d units of flow are allocated along each shortest path $p_m(d)$ for each demand d. The lengths l of each edge are updated after each step, in Line 20 of Algorithm E.1. This process continues until $\Delta(l) > 1$. Again Garg and Konemann show that with these initial starting conditions and length updates, the dual solution is ϵ-optimal; in addition to this, once a solution to the dual problem is calculated, the primal MCF solution can be determined by scaling the flows that were assigned along the paths in each step by $log_{1+\epsilon} 1/\delta$ [249]. Existing literature that discuss this algorithm do not directly state the complete MCF algorithm, neither do they provide any description of how the flow is updated. The MCF algorithm utilised as part of the IMFMC solution in Chapter 7 is shown clearly in Algorithm E.1, where it can be compared with the IMFMC algorithm shown in Algorithm 7.1, in order to observe the modifications implemented. Within the MCF algorithm, a flow $y \in Y$ is represented as a tuple $\langle flow, path \rangle$, with each of these values accessed through the functions $f(y)$ and $p(y)$ respectively.

The run time of the algorithm presented by Garg and Konemann is influenced by the optimal dual value β'. It is shown by Garg and Konemann [249] that the number of phases, t, can be derived from $t = \lceil \beta'/\epsilon \log_{1+\epsilon}(|E|/(1-e)) \rceil$, and that the algorithm is only successful when $\beta > 1$.

Fleischer describes how the algorithm parameters can be adjusted to give a reasonable runtime such that the reliance on β is mitigated. This is achieved by scaling the demands so that $\beta > 1$ is guaranteed, implying that each of the demands in the MCF can be fulfilled without exceeding capacity restrictions [250]. This can be done by calculating a suitable lower bound on a feasible flow, such as: a shortest path routing (SPR) scheme that first allocates all the demands over the

Algorithm E.1 The Garg and Konemann MCF algorithm [249], combined with the adaptations required in order to keep a record of the primal flow X [235]

Function MFMC$(G(E,V), D, \epsilon)$**:**
$n \leftarrow 1; \quad Y \leftarrow \emptyset; \quad \delta \leftarrow (|E|/(1-\epsilon))^{-1/\epsilon}$
$l_1(e) \leftarrow \delta / u(e) \quad \forall e \in E$
while $\Delta(l) \leq 1$ **do**
  for $d \in D$ **do**
    while $\Delta(l) < 1$ **and** $d > 0$ **do**
      $p_m(d) \leftarrow F(l_n)$
      $\mu \leftarrow \min(u(e) : e \in p_m(d))$
      if $d > \mu$ **then**
        $c \leftarrow \mu$
      else
        $c \leftarrow d$
      end if
      if $y \in Y$ **where** $p(y) == p_m(d)$ **then**
        $f(y) \leftarrow f(y) + c$
      else
        create new flow $y \in Y$ **where:**
        $p(y) \leftarrow p_m(d)$**;** $\quad f(y) \leftarrow c$
      end if
      $l_{n+1}(e) \leftarrow l_n(e) + \epsilon c / u(e) \quad \forall e \in p_m(d)$
      $d \leftarrow d - c; \quad n \leftarrow n + 1$
    end while
  end for
end while
$f(y) \leftarrow f(y) \log_{1+\epsilon} 1/\delta \quad \forall y \in Y$
$f_d \leftarrow \sum_{p \in P_d} y(p) \quad \forall d \in D$
$\theta \leftarrow \min(f_d / d) \quad \forall d \in D$
$p(x) \leftarrow p(y); \quad f(x) \leftarrow f(y)/\theta \quad \forall y \in Y$
return X

shortest path, assuming no capacity constraints, and then scales all the demands equally until the capacity constraint is met. This scaled demand will now satisfy $\beta > 1$, but has the possibility of giving a large β value, resulting in a large run time. Consequently, a *2-opt* MCF solution can also be calculated to provide an estimate of β as β_2 and the demands scaled by the solution, such that $1 < \beta \leq 2$. Here, *2-opt* means a solution that is within twice the optimum, which is fast to calculate, as it has a very loose tolerance. Fleischer shows that this produces a run time bounded by $t \leq \lceil 2/\epsilon \log_{1+\epsilon}(|E|/(1-e)) \rceil$ [250].

The described complex algorithm now provides a good solution to the *linear* MFMC problem; but the desired integer *IMFMC* solution is still not solved. However, extensions to the described MCF algorithm are presented in Chapter 7, which enable a solution for the IMFMC problem to be obtained; Chapter 7 then goes on to testing the presented IMFMC algorithm, with a detailed performance analysis through modelling techniques.

E.2 IMFMC Algorithm Modifications

The IMFMC algorithm presented in Algorithm 7.1 modifies the solution selection process of the original MFMC algorithm presented in Algorithm E.1, in order to store any integer solutions found. The addition of the described integer solution search functionality required a number of changes to be made to the MFMC optimisation process. In order to clearly highlight the required changes to the optimisation process, a description of the changes made to the algorithm will now be detailed by line number according to Algorithm 7.1. A few additional parameters are required to be instantiated, which can be seen at lines 3 and 5, these define the optimal IMFMC solution Ω' and the current integer solution flow data Γ respectively. Lines 14 and 15 are inserted to save flow and path data for the current demand to the current IMFMC solution Γ. In addition to this, Line 28 determines the minimum free capacity available on any given link Ω in the current solution Γ; this is then compared to the currently selected optimal solution on Line 29, where if it provides increased free capacity, it is set as the new optimal solution Ω', along with saving the corresponding flow data Γ as Γ'. It should be noted, however, that lines 28 and 29 are only processed if all demands in the current solution are met, as can be seen with the addition of Line 27. Finally, Line 36 returns both the MFMC solution X, as well as the best IMFMC flow data Γ' and its corresponding minimum free capacity Ω'. The remainder of the algorithm and its functions remain as described in Appendix E.1.1.

References

[1] J. Mambretti, M. Lemay, S. Campbell, H. Guy, T. Tam, E. Bernier, B. Ho, M. Savoie, C. de Laat, R. van der Pol, J. Chen, F. Yeh, S. Figuerola, P. Minoves, D. Simeonidou, E. Escalona, N. Amaya Gonzalez, A. Jukan, W. Bziuk, D. Kim, K. Cho, H.-L. Lee, and T.-L. Liu, "High performance digital media network (hpdmnet): An advanced international research initiative and global experimental testbed," *Future Generation Computer Systems*, vol. 27, no. 7, pp. 893–905, 2011.

[2] J. Mambretti, "Optiputer: Enabling advanced applications with novel optical control planes and backplanes," *Future Generation Computer Systems*, vol. 25, no. 2, pp. 137–141, 2009.

[3] D. Colle, G. Das, M. Pickavet, and P. Demeester, "Architectural trade-offs for video transport networks," in *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pp. 1–3.

[4] Sandvine. (2014) Sandvine - global internet phenomena report. [Online]. Available: https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf

[5] Cisco. (2015) Cisco visual networking index: Forecast and methodology, 2014–2019. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[6] ITU, "Bt.2020 : Parameter values for ultra-high definition television systems for production and international programme exchange," 2012.

[7] Apple Inc. (2015) Standard definition versus high definition video formats. [Online]. Available: https://documentation.apple.com/en/motion/usermanual/chapter_B_section_2.html

[8] ITU, "Bt.709 : Parameter values for the hdtv standards for production and international programme exchange," ITU, Recommendation, 2015.

[9] M. S. K. D. R. Sterling, M. Yoshimura, "Jvc d-ila 4096 x 2400 pixel projection display technology."

[10] J. Halak, M. Krsek, S. Ubik, P. Zejdl, and F. Nevrela, "Real-time long-distance transfer of uncompressed 4k video for remote collaboration," *Future Generation Computer Systems*, 2010.

[11] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video transcoding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 84–97, 2005.

[12] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang, "Video transcoding: an overview of various techniques and research issues," *Multimedia, IEEE Transactions on*, vol. 7, no. 5, pp. 793–804, 2005.

[13] S. Andriani, G. Calvagno, T. Erseghe, G. A. Mian, M. Durigon, R. Rinaldo, M. Knee, P. Walland, and M. Koppetz, "Comparison of lossy to lossless compression techniques for digital cinema," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 1, pp. 513–516 Vol. 1.

[14] B. Shi, L. Liu, and C. Xu, "Comparison between jpeg2000 and h.264 for digital cinema," in *Multimedia and Expo, 2008 IEEE International Conference on*, June 2008, pp. 725–728.

[15] F. Bellard, M. Niedermayer *et al.*, "Ffmpeg," *Available from: http://ffmpeg. org*, 2012.

[16] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[17] A. MacAulay, B. Felts, and Y. Fisher, "Whitepaper–ip streaming of mpeg-4: Native rtp vs mpeg-2 transport stream," *Envivo Inc. Retrieved October*, vol. 25, p. 2011, 2005.

[18] T. Schierl, K. Gruneberg, and T. Wiegand, "Scalable video coding over rtp and mpeg-2 transport stream in broadcast and iptv channels," *Wireless Communications, IEEE*, vol. 16, no. 5, pp. 64–71, October 2009.

[19] VideoLan, VLC. (2015) VLC Media player. [Online]. Available: http://www.videolan.org/

[20] G. Pearson and M. Gill, "An evaluation of motion jpeg 2000 for video archiving," *Proc. Archiving*, vol. 29, no. 2005, pp. 237–243, 2005.

[21] G. C. K. Abhayaratne, "Lossless and nearly lossless digital video coding," Ph.D. dissertation, 2002.

[22] J.-B. L. Meessen and Jerome, "Reliable video quality from production to archiving."

[23] J. R. N. Bancroft and Dave, "Jpeg 2000 in the hd broadcast production workflow," 2010.

[24] H. Kalva, "The h.264 video coding standard," *MultiMedia, IEEE*, vol. 13, no. 4, pp. 86–90, 2006.

[25] M. Mastriani, "Supercompression for full-hd and 4k-3d (8k) digital tv systems," *International Journal of Information and Mathematical Sciences*, vol. 6, no. 3, pp. 186–199, 2010.

[26] G. J. Sullivan, J. Ohm, H. Woo-Jin, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012.

[27] H.-M. Hang, W.-H. Peng, C.-H. Chan, and C.-C. Chen, "Towards the next video standard: high efficiency video coding," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conf*, pp. 609–618.

[28] I. E. Richardson, "Video codec design, developing image and video compression system," *John Wiley & Sons, Ltd. ISBN*, vol. 471485535, p. 9780471485537, 2002.

[29] G. Conklin, G. Greenbaum, K. Lillevold, A. Lippman, and Y. Reznik, "Video coding for streaming media delivery on the internet," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 11, no. 3, pp. 269–281, Mar 2001.

[30] R. Pantos and W. May, "Http live streaming draft-pantos-http-live-streaming-16," *Published by the Internet Engineering Task Force (IETF)*, 2015. [Online]. Available: https://tools.ietf.org/html/draft-pantos-http-live-streaming-16

[31] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, no. 4, pp. 62–67, 2011.

[32] A. Zambelli, "Iis smooth streaming technical overview," *Microsoft Corporation*, vol. 3, p. 40, 2009.

[33] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced engineering informatics*, vol. 19, no. 1, pp. 43–53, 2005.

[34] IBM. (2015) Cplex optimizer. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[35] B. E. Bixby N., *Using the CPLEX Callable Library.* CPLEX Optimization Inc, 1996.

[36] M.-B. Hu, R. Jiang, R. Wang, W.-B. Du, and Q.-S. Wu, "Optimization of network traffic," in *Control and Automation, 2009. ICCA 2009. IEEE International Conference on*, Dec 2009, pp. 2278–2281.

[37] J. Sun, S. Gao, W. Yang, and F. Huang, "Replica placement algorithms in content distribution networks," in *Information Engineering and Electronic Commerce (IEEC), 2010 2nd International Symposium on*, July 2010, pp. 1–4.

[38] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2001, pp. 1587–1596 vol.3.

[39] M. Yang and Z. Fei, "A model for replica placement in content distribution networks for multimedia applications," in *Communications, 2003. ICC '03. IEEE International Conference on*, vol. 1, May 2003, pp. 557–561 vol.1.

[40] T. Wauters, J. Coppens, B. Dhoedt, and P. Demeester, "Load balancing through efficient distributed content placement," in *Next Generation Internet Networks, 2005.* IEEE, 2005, pp. 99–105.

[41] I. Cidon, S. Kutten, and R. Soffer, "Optimal allocation of electronic content," *Computer Networks*, vol. 40, no. 2, pp. 205 – 218, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128602002517

[42] T. Leighton, "Improving performance on the internet," *Commun. ACM*, vol. 52, no. 2, pp. 44–51, Feb. 2009. [Online]. Available: http://doi.acm.org/10.1145/1461928.1461944

[43] T. V. Nguyen, C. T. Chou, and P. Boustead, "Provisioning content distribution networks over shared infrastructure," in *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, Sept 2003, pp. 119–124.

[44] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X08001957

[45] Amazon. (2015) Amazon Web Services. [Online]. Available: http://aws.amazon.com

[46] H. Li, *Introducing Windows Azure*. Berkely, CA, USA: Apress, 2009.

[47] Google, Inc. (2015, July) Google cloud platform. [Online]. Available: https://cloud.google.com/

[48] Amazon, "Amazon elastic compute cloud (Amazon EC2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[49] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: Understanding personal cloud storage services," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 481–494. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398827

[50] R. Uhlig, G. Neiger, D. Rodgers, A. Santoni, F. Martins, A. Anderson, S. Bennett, A. Kagi, F. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, May 2005.

[51] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/1165389.945462

[52] M. Pretorius, M. Ghassemian, and C. Ierotheou, "An investigation into energy efficiency of data centre virtualisation," in *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on*, Nov 2010, pp. 157–163.

[53] J. Nie, "A study on the application cost of server virtualisation," in *Computational Intelligence and Security (CIS), 2013 9th International Conference on*, Dec 2013, pp. 807–811.

[54] C. A. Waldspurger, "Memory resource management in vmware esx server," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844146

[55] C.-H. Li, "Evaluating the effectiveness of memory overcommit techniques on kvm-based hosting platform," in *Proceedings of World Academy of Science, Engineering and Technology*, no. 70. World Academy of Science, Engineering and Technology, 2012.

[56] KVM. (2015, July) Kvm - kernel-based virtual machine. [Online]. Available: http://www.linux-kvm.org/page/Main_Page

[57] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[58] Oracle. (2015, July) Virtualbox - x86 and amd64/intel64 virtualization. [Online]. Available: https://www.virtualbox.org/

[59] J. Watson, "Virtualbox: bits and bytes masquerading as machines," *Linux Journal*, vol. 2008, no. 166, p. 1, 2008.

[60] OpenVZ. (2015, July) Openvz - a container-based virtualization for linux. [Online]. Available: http://www.openvz.org/Main_Page

[61] A. Kovári and P. Dukan, "Kvm & openvz virtualization based iaas open source cloud virtualization platforms: Opennode, proxmox ve," in *Intelligent Systems and Informatics (SISY), 2012 IEEE 10th Jubilee International Symposium on*. IEEE, 2012, pp. 335–339.

[62] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *Communications Magazine, IEEE*, vol. 50, no. 9, pp. 34–40, September 2012.

[63] J. T. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," in *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, Nov 2010, pp. 87–92.

[64] S. Sharma and M. Chawla, "A technical review for efficient virtual machine migration," in *Cloud Ubiquitous Computing Emerging Technologies (CUBE), 2013 International Conference on*, Nov 2013, pp. 20–25.

[65] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[66] T. Hirofuchi, H. Nakada, S. Itoh, and S. Sekiguchi, "Reactive consolidation of virtual machines enabled by postcopy live migration," in *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*. ACM, 2011, pp. 11–18.

[67] E. Silvera, G. Sharaby, D. Lorenz, and I. Shapira, "Ip mobility to support live migration of virtual machines across subnets," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 13:1–13:10. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534548

[68] C. Perkins, "IP Encapsulation within IP," RFC 2003 (Proposed Standard), Internet Engineering Task Force, Oct. 1996, updated by RFCs 3168, 6864. [Online]. Available: http://www.ietf.org/rfc/rfc2003.txt

[69] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd International Conference on Virtual Execution Environments*, ser. VEE '07. New York, NY, USA: ACM, 2007, pp. 169–179. [Online]. Available: http://doi.acm.org/10.1145/1254810.1254834

[70] F. Travostino, P. Daspit, L. Gommans, C. Jog, C. De Laat, J. Mambretti, I. Monga, B. Van Oudenaarde, S. Raghunath, and P. Y. Wang, "Seamless live migration of virtual machines over the man/wan," *Future Generation Computer Systems*, vol. 22, no. 8, pp. 901–907, 2006.

[71] A. Bestavros, M. Crovella, J. Liu, and D. Martin, "Distributed packet rewriting and its application to scalable server architectures," in *Network Protocols, 1998. Proceedings. Sixth International Conference on*, Oct 1998, pp. 290–297.

[72] H. Maziku and S. Shetty, "Network aware vm migration in cloud data centers," in *Research and Educational Experiment Workshop (GREE), 2014 Third GENI*, March 2014, pp. 25–28.

[73] R. Cziva, D. Stapleton, F. P. Tso, and D. Pezaros, "Sdn-based virtual machine management for cloud data centers," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 388–394.

[74] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, and R. Martinez, "Experimental seamless virtual machine migration using an integrated sdn it and network orchestrator," in *Optical Fiber Communications Conference and Exhibition (OFC), 2015*, March 2015, pp. 1–3.

[75] J. Liu and D. Jin, "Software defined live virtual machine migration," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, Oct 2013, pp. 1–3.

[76] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "Openflow supporting inter-domain virtual machine migration," in *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*, May 2011, pp. 1–7.

[77] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "Crossroads: Seamless vm mobility across data centers through software defined networking," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 88–96.

[78] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, pp. 109–114.

[79] Y. Ye, Y. He, and X. Xiu, "Manipulating ultra high definition video traffic," *MultiMedia, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[80] A. Passarella, "A survey on content-centric technologies for the current internet: Cdn and p2p solutions," *Computer Communications*, vol. 35, no. 1, pp. 1–32, 2012.

[81] S. Lukasik, "Why the arpanet was built," *Annals of the History of Computing, IEEE*, vol. 33, no. 3, pp. 4–21, March 2011.

[82] BT Group. (2015) BT Vision and TV. [Online]. Available: https://www.productsandservices.bt.com/products/tv-packages/

[83] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1842733.1842736

[84] Alcatel-Lucent, *Leverages Alcatel-Lucent Velocix solution for new Content Delivery Network*, 2010. [Online]. Available: https://www.alcatel-lucent.com/press/2010/002229

[85] A. Berson, *Client-server architecture*. McGraw-Hill, 1992, no. IEEE-802.

[86] H. S. Bassali, K. M. Kamath, R. B. Hosamani, and L. Gao, "Hierarchy-aware algorithms for cdn proxy placement in the internet," *Computer Communications*, vol. 26, no. 3, pp. 251–263, 2003.

[87] A. Vakali and G. Pallis, "Content delivery networks: status and trends," *Internet Computing, IEEE*, vol. 7, no. 6, pp. 68–74, Nov 2003.

[88] D. Dan, C. Lin, Z. Xuemei, and Z. Yiwei, "Cache mechanism in p2p streaming media system," in *Networking and Digital Society (ICNDS), 2010 2nd International Conference on*, vol. 2, pp. 376–378.

[89] J. Hai, L. Jun, L. Zhongcheng, and B. Xiangyu, "Performance evaluation of content distribution in hybrid cdn-p2p network," in *Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on*, vol. 1, pp. 188–193.

[90] H. Yin, C. Lin, Q. Zhang, Z. Chen, and D. Wu, "Truststream: A secure and scalable architecture for large-scale internet media streaming," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, no. 12, pp. 1692–1702, Dec 2008.

[91] E. T. Lin, A. M. Eskicioglu, R. L. Lagendijk, and E. J. Delp, "Advances in digital video content protection," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 171–183, 2005.

[92] P. Rodriguez, S.-M. Tan, and C. Gkantsidis, "On the feasibility of commercial, legal p2p content distribution," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 75–78, 2006.

[93] B. S. Davie and Y. Rekhter, *MPLS: technology and applications*. San Francisco, 2000.

[94] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta, "MPLS Label Stack Encoding," RFC 3032 (Proposed Standard), Internet Engineering Task Force, Jan. 2001, updated by RFCs 3443, 4182, 5332, 3270, 5129, 5462, 5586, 7274. [Online]. Available: http://www.ietf.org/rfc/rfc3032.txt

[95] F. J. Hens and J. M. Caballero, "Carrier-class ethernet," *Triple Play: Building the Converged Network for IP, VoIP and IPTV*, pp. 305–350.

[96] M. Laubach, "Classical IP and ARP over ATM," RFC 1577 (Proposed Standard), Internet Engineering Task Force, Jan. 1994, obsoleted by RFC 2225. [Online]. Available: http://www.ietf.org/rfc/rfc1577.txt

[97] C. Brown and A. Malis, "Multiprotocol Interconnect over Frame Relay," RFC 2427 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2427.txt

[98] N. Briscoe, "Understanding the osi 7-layer model," *PC Network Advisor*, vol. 120, no. 2, 2000.

[99] S. Skiena, "Dijkstra's algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.

[100] J. Moy, "OSPF Version 2," RFC 1247 (Draft Standard), Internet Engineering Task Force, Jul. 1991, obsoleted by RFC 1583, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc1247.txt

[101] G. Malkin, "RIP Version 2 Protocol Analysis," RFC 1721 (Informational), Internet Engineering Task Force, Nov. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1721.txt

[102] R. Albrightson, J. Garcia-Luna-Aceves, and J. Boyle, "Eigrp–a fast routing protocol based on distance vectors," *Interop 94*, 1994.

[103] K. Lougheed and Y. Rekhter, "Border Gateway Protocol 3 (BGP-3)," RFC 1267 (Historic), Internet Engineering Task Force, Oct. 1991. [Online]. Available: http://www.ietf.org/rfc/rfc1267.txt

[104] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC 1654 (Proposed Standard), Internet Engineering Task Force, Jul. 1994, obsoleted by RFC 1771. [Online]. Available: http://www.ietf.org/rfc/rfc1654.txt

[105] Open Networking Foundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, 2012.

[106] K. Kirkpatrick, "Software-defined networking," *Commun. ACM*, vol. 56, no. 9, pp. 16–19, Sep. 2013. [Online]. Available: http://doi.acm.org/10.1145/2500468.2500473

[107] H. Kim and N. Feamster, "Improving network management with software defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, February 2013.

[108] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[109] C. Alaettinoglu, "Software defined networking," 2013. [Online]. Available: http://www.packetdesign.com/resources/white-papers/sdn-white-paper.pdf

[110] ONF. (2015, July) Open Networking Foundation (ONF). [Online]. Available: https://www.opennetworking.org/about/onf-overview

[111] "OpenFlow v1.5.1 Specification," 2015. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[112] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 2211–2219.

[113] Google Inc. (2015, July) Google. [Online]. Available: https://www.google.com

[114] U. Hoelzle, "Opening address: 2012 open network summit," *http://www. opennetsummit. org/archives/apr12/hoelzle-tue-openflow. pdf, Date Retrieved*, vol. 8, no. 08, p. 2014, 2012.

[115] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter." in *HotNets*, 2009.

[116] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[117] S. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 136–141, February 2013.

[118] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[119] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, ser. ITC '11. International Teletraffic Congress, 2011, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043468.2043470

[120] A. Bianco, R. Birke, L. Giraudo, and M. Palacin, "Openflow switching: Data plane performance," in *Communications (ICC), 2010 IEEE International Conference on*, May 2010, pp. 1–5.

[121] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043164.2018466

[122] M. Fernandez, "Comparing openflow controller paradigms scalability: Reactive and proactive," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, March 2013, pp. 1009–1016.

[123] K. Phemius and M. Bouet, "Openflow: Why latency does matter," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, May 2013, pp. 680–683.

[124] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "Oflops: An open framework for openflow switch evaluation," in *Passive and Active Measurement*. Springer, 2012, pp. 85–95.

[125] C. Rotsos, G. Antichi, M. Bruyere, P. Owezarski, and A. Moore, "An open testing framework for next-generation openflow switches," in *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, Sept 2014, pp. 127–128.

[126] Z. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight sdn controller," in *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, Dec 2014, pp. 671–676.

[127] "OpenFlow v1.2 Specification," 2011. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf

[128] "OpenFlow v1.1 Specification," 2011. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf

[129] NOX. (2015) Nox openflow controller. [Online]. Available: http://www.noxrepo.org/

[130] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[131] A. Lara, A. Kolasani, and B. Ramamurthy, "Network innovation using openflow: A survey," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 493–512, First 2014.

[132] X. Zhang, W. G. Hou, P. C. Han, and L. Guo, "Design and implementation of the routing function in the nox controller for software-defined networks," in *Applied Mechanics and Materials*, vol. 635. Trans Tech Publ, 2014, pp. 1540–1543.

[133] Project Floodlight. (2015) Floodlight openflow controller. [Online]. Available: http://www.projectfloodlight.org/

[134] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[135] OpenDaylight. (2015) Opendaylight controller infrastructure. [Online]. Available: https://www.opendaylight.org/

[136] J. Medved, A. Tkacik, R. Varga, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, June 2014, pp. 1–6.

[137] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, vol. 54, 2012.

[138] S. Shah, J. Faiz, M. Farooq, A. Shafi, and S. Mehdi, "An architectural evaluation of sdn controllers," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 3504–3508.

[139] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[140] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "Balanceflow: Controller load balancing for openflow networks," in *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, vol. 02, Oct 2012, pp. 780–785.

[141] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2491193

[142] J. Nagano and N. Shinomiya, "Efficient information sharing among distributed controllers of openflow network with bi-connectivity," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*, Feb 2015, pp. 320–324.

[143] GENI.net. (2015) Global environment for network. [Online]. Available: http://www.geni.net

[144] M. Berman, C. Elliott, and L. Landweber, "Geni: Large-scale distributed infrastructure for networking and distributed systems research," in *Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on*, July 2014, pp. 156–161.

[145] P. Yi and Z. Fei, "Characterizing the geni networks," in *Research and Educational Experiment Workshop (GREE), 2014 Third GENI*, March 2014, pp. 53–56.

[146] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[147] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, "Carving research slices out of your production networks with openflow," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1672308.1672333

[148] S. Min, S. Kim, J. Lee, B. Kim, W. Hong, and J. Kong, "Implementation of an openflow network virtualization for multi-controller environment," in *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, Feb 2012, pp. 589–592.

[149] OvS. (2015) Open vSwitch. [Online]. Available: http://openvswitch.org/

[150] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," *Proc. HotNets (October 2009)*, 2009.

[151] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the production network be the testbed," *Proc. OSDI (October 2010)*, 2010.

[152] G. Combs, "Wireshark," 2015. [Online]. Available: https://www.wireshark.org/

[153] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," RFC 3176 (Informational), Internet Engineering Task Force, Sep. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3176.txt

[154] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[155] V. Mann, A. Vishnoi, and S. Bidkar, "Living on the edge: Monitoring network flows at the edge in cloud data centers," in *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, Jan 2013, pp. 1–9.

[156] sFlow. (2015) Traffic monitoring using sflow. [Online]. Available: http://www.sFlow.org

[157] S. Rehman, W.-C. Song, and M. Kang, "Network-wide traffic visibility in of@tein sdn testbed using sflow," in *Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific*, Sept 2014, pp. 1–6.

[158] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 245–256, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1030194.1015495

[159] L. Deri and N. SpA, "nprobe: an open source netflow probe for gigabit networks," in *TERENA Networking Conference (TNC 2003), Zagreb, Croatia*, 2003.

[160] M. Wichtlhuber, R. Reinecke, and D. Hausheer, "An sdn-based cdn/isp collaboration architecture for managing high-volume flows," *Network and Service Management, IEEE Transactions on*, vol. 12, no. 1, pp. 48–60, 2015.

[161] B. Quinn and K. Almeroth, "IP Multicast Applications: Challenges and Solutions," RFC 3170 (Informational), Internet Engineering Task Force, Sep. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3170.txt

[162] R. Finlayson, "IP Multicast and Firewalls," RFC 2588 (Informational), Internet Engineering Task Force, May 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2588.txt

[163] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *Communications Surveys & Tutorials, IEEE*, vol. 9, no. 3, pp. 58–74, 2007.

[164] S. Banerjee and B. Bhattacharjee, "A comparative study of application layer multicast protocols," *Network*, vol. 4, p. 3, 2002.

[165] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *Network, IEEE*, vol. 14, no. 1, pp. 78–88, 2000.

[166] J. Guochao, T. Tsuchiya, H. Sawano, T. Kugimoto, K. Sebayashi, O. Akashi, M. Maruyama, and K. Koyanagi, "Study on multi-path transmission model based on p2p overlay networks for streaming data," in *Applications and the Internet, 2009. SAINT '09. Ninth Annual International Symposium on*, pp. 271–274.

[167] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1456–1471, Oct 2002.

[168] Twitch. (2015) Twitch.tv streaming app. [Online]. Available: http://www.twitch.tv

[169] ——. (2014) Twitch.tv streaming app report. [Online]. Available: http://www.twitch.tv/year/2014

[170] Google. (2015) Youtube. [Online]. Available: http://www.youtube.com

[171] Periscope. (2015) Periscope website. [Online]. Available: https://www.periscope.tv

[172] Meerkat. (2015) Meerkat website. [Online]. Available: https://meerkatapp.co/

[173] Kamcord. (2015) Kamcord website. [Online]. Available: https://www.kamcord.com

[174] DeNA Co, Ltd. (2015) Mirrativ: Live stream any app. [Online]. Available: https://play.google.com/store/apps/details?id=com.dena.mirrativ

[175] Medium. (2015) Periscope, by the numbers. [Online]. Available: https://medium.com/@periscope/periscope-by-the-numbers-6b23dc6a1704

[176] T. Hoff. (2015) What do we know about how meerkat works? [Online]. Available: http://highscalability.com/blog/2015/4/6/what-do-we-know-about-how-meerkat-works.html

[177] T. Hatmaker. (2015) Meerkat's founder explains his vision for the internet's favorite livestreaming app. [Online]. Available: http://www.dailydot.com/technology/meerkat-sxsw-2015-ben-rubin/

[178] M. Alam. (2015) Meerkat – livestreaming app. [Online]. Available: http://www.alamtechstuffs.com/meerkat-livestreaming-app/

[179] T. Levent-Levi. (2015) Can webrtc improve meerkat and periscope? [Online]. Available: https://bloggeek.me/webrtc-improve-meerkat-periscope/

[180] F. Hao, T. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009, pp. 37–44.

[181] P. Yan, D. Yilong, and A. Nakao, "Cloud rack: Enhanced virtual topology migration approach with open vswitch," in *Information Networking (ICOIN), 2011 International Conference on*, pp. 160–164.

[182] Y. Sambe, S. Watanabe, Y. Dong, T. Nakamura, and N. Wakamiya, "High-speed distributed video transcoding for multiple rates and formats," *IEICE transactions on information and systems*, vol. 88, no. 8, pp. 1923–1931, 2005.

[183] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, pp. 277–288, 1984.

[184] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *Communications Magazine, IEEE*, vol. 50, no. 9, pp. 34–40, 2012.

[185] E.-D. Zhao, Y.-Q. Qi, X.-X. Xiang, and Y. Chen, "A data placement strategy based on genetic algorithm for scientific workflows," in *Computational Intelligence and Security (CIS), 2012 Eighth International Conference on*, pp. 146–149.

[186] J. Wu and K. Ravindran, "Optimization algorithms for proxy server placement in content distribution networks," in *Integrated Network Management-Workshops, 2009. IM '09. IFIP/IEEE International Symposium on*, June 2009, pp. 193–198.

[187] R. Shevchuk, "The location of transcoders in multiservice networks," 2010.

[188] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid cdn-p2p: why limelight needs its own red swoosh," in *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2008, pp. 75–80.

[189] J. Cervino, P. Rodriguez, I. Trajkovska, A. Mozo, and J. Salvachua, "Testing a cloud provider network for hybrid p2p and cloud streaming architectures." IEEE, pp. 356–363.

[190] S. Wuyao, H. Lei, Z. Xingming, and Z. Chong, "A new streaming media network architecture based on the fusion of p2p and cdn," in *Multimedia Technology (ICMT), 2010 International Conference on*, pp. 1–6.

[191] L.-y. Zhou and X. Zhang, "The research of vod system performance based on cdn and p2p technologies," in *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 4, pp. 385–388.

[192] T. Junjie, X. Ke, and P. Renjie, "A new web service structure of combining p2p and cdn technologies," in *Web Society (SWS), 2010 IEEE 2nd Symposium on*, pp. 475–479.

[193] K. Thinh Nguyen, J. Seil, and K. Younghan, "A cdn-p2p hybrid architecture with content/location awareness for live streaming service networks," in *Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on*, pp. 438–441.

[194] L. Yu, Y. Hao, Z. Guangxi, and L. Xuening, "Peer-assisted content delivery network for live streaming: Architecture and practice," in *Networking, Architecture, and Storage, 2008. NAS '08. International Conference on*, pp. 149–150.

[195] L. Xuening, Y. Hao, and L. Chuang, "A novel and high-quality measurement study of commercial cdn-p2p live streaming," in *Communications and Mobile Computing, 2009. CMC '09. WRI International Conference on*, vol. 3, pp. 325–329.

[196] K. Junemann, P. Andelfinger, J. Dinger, and H. Hartenstein, "Bitmon: A tool for automated monitoring of the bittorrent dht," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, Aug 2010, pp. 1–2.

[197] M. Kelaskar, V. Matossian, P. Mehra, D. Paul, and M. Parashar, "A study of discovery mechanisms for peer-to-peer applications," in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, May 2002, pp. 444–444.

[198] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys Tutorials, IEEE*, vol. 7, no. 2, pp. 72–93, Second 2005.

[199] S. M. Y. Seyyedi and B. Akbari, "Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes," in *Computer Networks and Distributed Systems (CNDS), 2011 International Symposium on*, pp. 175–180.

[200] A. Mansy and M. Ammar, "Analysis of adaptive streaming for hybrid cdn/p2p live video systems," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pp. 276–285.

[201] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[202] G. Mann, "4k technology – is the tipping point in sight?" *Broadband, Journal of the SCTE*, vol. 36, no. 3, pp. 66–70, 2014.

[203] P. Farrow and M. Reed, "Optimising the geographical location of transcoding resources," in *Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on*. IEEE, 2013, pp. 58–62.

[204] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976. [Online]. Available: http://dx.doi.org/10.1137/0205048

[205] G. Vignaux and Z. Michalewicz, "A genetic algorithm for the linear transportation problem," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 21, no. 2, pp. 445–452, Mar 1991.

[206] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs paralleles, reseaux et systems repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[207] Odeon Cinemas. (2015) Event cinema. [Online]. Available: http://www.odeon.co.uk/eventcinema/

[208] Vue Cinemas. (2015) Event cinema. [Online]. Available: http://www.myvue.com/event-cinema/coming-soon

[209] Showcase Cinemas. (2015) Event cinema. [Online]. Available: http://www.showcasecinemas.co.uk/events

[210] J. Pettit and E. Lopez, "Openstack: Ovs deep dive," 2013. [Online]. Available: http://noxrepo.net/slides/OpenStack-131107.pdf

[211] Pica8. (2015) Pica8: White box sdn. [Online]. Available: http://www.pica8.com

[212] ——. (2015) Pica8 datasheet for p-3290 and p-3295. [Online]. Available: http://www.pica8.com/documents/pica8-datasheet-48x1gbe-p3290-p3295.pdf

[213] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (INTERNET STANDARD), Internet Engineering Task Force, Jul. 2003, updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[214] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*. Citeseer, 2005, pp. 18–23.

[215] PSNC, "Poznan Supercomputing and Networking Center." [Online]. Available: http://www.man.poznan.pl/online/en/

[216] J. Mambretti, "iCAIR - International Center for Advanced Internet Research." [Online]. Available: http://www.icair.org/

[217] G. Combs, *tshark - The Wireshark Network Analyzer 1.12.2*, 2014. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[218] C.-C. Yeh and C.-W. Chiu, "Mint: A cost-effective network-address translation architecture with multiple inexpensive nat servers," in *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*, July 2015, pp. 879–881.

[219] S. Yan, Q. Zhao, X. Huang, and Y. Ma, "A migrating optimization method for cdn based on distributed mobility management," in *Broadband Network Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on*, Nov 2013, pp. 155–159.

[220] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787 (Best Current Practice), Internet Engineering Task Force, Jan. 2007, updated by RFC 6888. [Online]. Available: http://www.ietf.org/rfc/rfc4787.txt

[221] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663 (Informational), Internet Engineering Task Force, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2663.txt

[222] D. Badami, N. Thanthry, T. Best, R. Bhagavathula, and R. Pendse, "Port address translation based route optimization for mobile ip," in *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, vol. 5, Sept 2004, pp. 3110–3114 Vol. 5.

[223] Netflix, Inc. (2015) Netflix. [Online]. Available: https://www.netflix.com/

[224] ——. (2015) Netflix open connect. [Online]. Available: https://openconnect.netflix.com/

[225] Google, Inc. (2015) Peering and content delivery. [Online]. Available: https://peering.google.com/about/faq.html

[226] M. Anvari, "Telesurgery: remote knowledge translation in clinical surgery," *World journal of surgery*, vol. 31, no. 8, pp. 1545–1550, 2007.

[227] M. Anvari, C. McKinley, and H. Stein, "Establishment of the world's first telerobotic remote surgical service: for provision of advanced laparoscopic surgery in a rural community," *Annals of surgery*, vol. 241, no. 3, p. 460, 2005.

[228] R. Khosla, S. Fahmy, and Y. Hu, "Content retrieval using cloud-based dns," in *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, March 2012, pp. 1–6.

[229] P. Vixie, "What dns is not." *Commun. ACM*, vol. 52, no. 12, pp. 43–47, 2009.

[230] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Comparing dns resolvers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 15–21. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879144

[231] M. J. Freedman, "automating server selection with oasis," *; login:: the magazine of USENIX & SAGE*, vol. 31, no. 5, pp. 46–52, 2006.

[232] K. Kumar. (2014) Andromeda: internal sdn and nfv at google. [Online]. Available: http://opennetsummit.org/blog/2014/03/andromeda-internal-sdn-and-nfv-at-google/

[233] C. Ellrod. (2014) Is akamai aura virtualized. [Online]. Available: https://community.akamai.com/blogs/akaCraig/2014/10/31/is-akamai-aura-virtualized

[234] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz, "An overview of constraint-based path selection algorithms for qos routing," *IEEE Communications Magazine, 40 (12)*, 2002.

[235] M. J. Reed, "Traffic engineering for information-centric networks," in *Communications (ICC), 2012 IEEE International Conference on.* IEEE, 2012, pp. 2660–2665.

[236] Y. Lee, I. Park, and Y. Choi, "Improving TCP performance in multipath packet forwarding networks," *Journal of Communications and Networks*, vol. 4, no. 2, pp. 148–157, 2002.

[237] H. Bryhni, E. Klovning, and O. Kure, "A comparison of load balancing techniques for scalable web servers," *Network, IEEE*, vol. 14, no. 4, pp. 58–64, Jul 2000.

[238] O.-D. Ntofon, D. Hunter, and D. Simeonidou, "Towards a resource-aware scheduling framework for cost optimization in future media infrastructures deployed over optical networks," in *Transparent Optical Networks (ICTON), 2012 14th International Conference on*, July 2012, pp. 1–6.

[239] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European control conference*, vol. 1, 2001, pp. 2603–2608.

[240] S. Hanks, T. Li, D. Farinacci, and P. Traina, "Generic Routing Encapsulation over IPv4 networks," RFC 1702 (Informational), Internet Engineering Task Force, Oct. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1702.txt

[241] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation (GRE)," RFC 2784 (Proposed Standard), Internet Engineering Task Force, Mar. 2000, updated by RFC 2890. [Online]. Available: http://www.ietf.org/rfc/rfc2784.txt

[242] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)," RFC 2637 (Informational), Internet Engineering Task Force, Jul. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2637.txt

[243] S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)," RFC 4310 (Proposed Standard), Internet Engineering Task Force, Dec. 2005, obsoleted by RFC 5910. [Online]. Available: http://www.ietf.org/rfc/rfc4310.txt

[244] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[245] OpenVPN Technologies, Inc. (2015) OpenVPN - Open Source VPN. [Online]. Available: URLhttp://openvpn.net

[246] M. Feilner, *OpenVPN: Building and integrating virtual private networks*. Packt Publishing Ltd, 2006.

[247] T. Ehrenkranz and J. Li, "On the state of ip spoofing defense," *ACM Trans. Internet Technol.*, vol. 9, no. 2, pp. 6:1–6:29, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1516539.1516541

[248] R. Beverly, A. Berger, Y. Hyun, and k. claffy, "Understanding the efficacy of deployed internet source address validation filtering," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 356–369. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644936

[249] N. Garg and J. Konemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *39th Symp. On Foundations Of Computer Science*, 1998, pp. 300–309.

[250] L. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM JOURNAL ON DISCRETE MATHEMATICS*, vol. 13, no. 4, pp. 505–520, 2000.