HYBRID MEDIA STREAMING ARCHITECTURE, FOCUSSING ON QUANTIFICATION OF SCALABILITY OF MEDIA STREAMING ARCHITECTURES, INCLUDING STREAMING PROTOCOL EVALUATION

Wayne Frederick

Submitted in fulfilment of the Master of Science in Engineering to the Faculty of Engineering, University of KwaZulu-Natal

9 September 2013

Supervisor: Dr. A.L.L. Jarvis

Declarations

As the candidate's Supervisor I agree/do not agree to the submission of this thesis.

Signed: _____ Name: _____ Date: _____

DECLARATION

I, Wayne Frederick, declare that

(i) The research reported in this dissertation, except where otherwise indicated, is my original work.

(ii) This dissertation has not been submitted for any degree or examination at any other university.

(iii) This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

a) their words have been re-written but the general information attributed to them has been referenced;

b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) Where I have reproduced a publication of which I am an author, co-author or editor, I have indicated in detail which part of the publication was actually written by myself alone and have fully referenced such publications.

(vi) This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Signed: _____

Acknowledgements

My gratitude is deserved by many for their support during the course of this research. Unfortunately, I cannot mention everyone here, but hope that they are aware of my appreciation.

My first thanks goes to Allied Technologies Limited (Altech) for their financial support and allowing me to pursue a Master of Science in Engineering Degree. To the material science lab crowd, Jonathan, Roland, Darryl and Sergio, thank you for your support through the good and bad, and for those real interesting conversations at tea. To all my family and friends, thank you for understanding my missing out on so many occasions and always offering a helping hand whenever I needed it.

It is with great pleasure that I thank my Supervisor, Dr Leigh Jarvis, perhaps the busiest person I have ever met, for always making that extra bit of time for me. For those early mornings and late afternoons, your guidance has helped me the most in performing this research. I must also mention your spark of genius that got the ball rolling for this research, in a time that did seem very dire to me. Being an undergrad and postgrad under your supervision is something I will remember with honour and gratitude and I am forever indebted to you.

To my parents, who have had to put up with me for the longest time during this research, Mum and Dad, I especially thank you. Not just for the last two years, but for every year before that, that helped me get to this stage of my life. Also, a special thank you to Kimmy, for always being there for me, and never letting me give up no matter how difficult it got.

Abstract

An analysis of media streaming architectures is performed in this dissertation drawing focus to the advantages and disadvantages of the various architectures. Detailed examination is given to scalability of decentralized Peer-to-Peer (P2P) systems in comparison to centralized client-server systems in an aim to quantify the difference in scalability between the architectures. Research has indicated that decentralized architectures are found to have better scalability.

Taking into consideration the various factors and protocols encountered in many researched multimedia streaming architectures, a decentralized P2P Video on Demand (VoD) media streaming system using Set-Top Boxes (STBs) is proposed.

Both centralized and decentralized architectures are simulated in Network Simulator 2 (NS2), with the inclusion of an enhanced Evalvid (Video Evaluation) toolset which allows utilization of Peak Signal to Noise Ratio (PSNR) to determine received video quality. An evaluation methodology, referred to as the Quality and Scalability Quantification System (QSQS), is developed and successfully tested using PSNR to determine differences in quality and scalability between the architectures. Results indicate that the decentralized architecture has on average 15 % higher delivered video quality and 34 % higher scalability than the centralized architecture under similar operating conditions. Under very high network loads, drawbacks to decentralized architectures are observed and examined, with a centralized nature emerging amongst peer nodes causing a resultant loss in network robustness. In response to this, a protocol that introduces serving peers is implemented in the decentralized architecture. Evaluation of the serving peer protocol found an increase of almost 30 % in delivered video quality to all peers when compared to the standard architecture.

Performance of the evaluation methodology in evaluating the serving peer protocol enhances possibilities of using this method as a Quality of Experience (QoE) indicator in addition to existing Quality of Service (QoS) evaluation methods.

Acronyms

API	-	Application Protocol Interface
AVC	-	Advanced Video Coding
CBR	-	Constant Bit Rate
CDN	-	Content Delivery Network
dB	-	Decibel
DHT	-	Distributed Hash Table
DLL	-	Dynamic Link Library
ET	-	Evaluate Trace
FIFO	-	First-In First-Out
FQ	-	Fair Queuing
ISP	-	Internet Service Provider
LAN	-	Local Area Network
MB	-	Megabyte
Mbps	-	Megabits per second
MDC	-	Multiple Description Coding
MDCC	-	Mixed-Driven Congestion Control
MOS	-	Mean Opinion Square
MPEG	-	Moving Picture Experts Group
ms	-	millisecond
NAM	-	Network Animator
NS2	-	Network Simulator 2
OTcl	-	Object Oriented Tool Command Language

P2P	-	Peer-to-Peer
PSNR	-	Peak Signal to Noise Ratio
QoE	-	Quality of Experience
QoS	-	Quality of Service
QSQS	-	Quality and Scalability Quantification System
RTCP	-	RTP Control Protocol
RTP	-	Real Time Transport Protocol
SFQ	-	Stochastic Fair Queuing
SIP	-	Session Initiation Protocol
STB	-	Set-Top-Box
SVC	-	Scalable Video Coding
TCP	-	Transmission Control Protocol
TFRC	-	TCP Friendly Rate Control
UDP	-	User Datagram Protocol
VLC	-	VideoLAN
VoD	-	Video-on-Demand
VS	-	Video Sender

Contents

List of Figures

List of tables

Chapter 1 Introduction

1.1 Media Streaming

Over the past decade, media streaming has attracted much attention and forms the basis of many intensive research areas. Online sharing of videos is responsible for the highest traffic on the Internet [1]. To address the vast increase in public demand for streaming media, many systems were developed which address the issue of serving clients. Central Client-Server and decentral P2P designs have emerged as the two primary architectures for effective media streaming.

Initially, a central server that serves client requests directly was developed. Although simple in nature, as demand for media increased, flaws with this architecture emerged. Video data requires higher bandwidth as quality of the video increases. An increasing demand for a video translates to a central server having to serve large amounts of data to multiple clients simultaneously. For example, a typical Blu-Ray disc using the MPEG4 AVC video codec requires 8-12 Mbps for streaming a 1080p Full High Definition (FHD) video [2]. Maintaining a persistent 18 Mbps download rate for a video, with duration of about 2 hours, for a number of clients, is hardware intensive due to high bandwidth requirements and increased Central Processing Unit (CPU) server load. Increasing hardware allocation requires more servers, thus rendering centralized architecture economically inefficient for a modest client base. With an increase in clients, a central server is under provisioned to handle the higher load and cannot meet the demands posed by increased client requests. A drop in streamed video quality across the network is a further drawback of increased demand. Alternatives to centralized architecture were developed as a result of the scalability issue with central servers [3].

P2P overlay networks, running at the application layer of a central backbone network, were the first improvement to scalability issues of the client-server network [4]. Under higher network loads, the caching and streaming ability of clients are utilized by the server. As a client is streaming from a server, it caches the part of the video it has already watched so that the next client that requests

the same video can stream from the first client. In this way, the server can deal with an increased load due to a higher number of requests from an increasing client base. However, this approach was limited by the lack of scalability in handling diverse client requests and network quality still degraded.

Server duplication to handle increasing user bases is an effective approach to dealing with the scalability problem of client-server architectures. These networks are generally referred to as Content Delivery Networks (CDNs) for streaming media applications. Servers are duplicated across different areas of the user population, where most of the client bases are situated, and are referred to as 'edge' servers. A main content server feeds the duplicate servers, allowing a higher number of requests to be served to an increasing client base. As the user base grows, more duplication is needed and costs start to significantly increase. CDNs for streaming media are an active field of research.

The P2P architecture is the least expensive but most unreliable approach to media streaming. In a pure P2P architecture, no server exists and the network uses the storage and streaming capabilities of network peers to allow streaming. This approach was borne out of decentralized file sharing systems such as BitTorrent [5] or Gnutella [6]. Since the mid 2000's, these approaches have been steadily replacing client-server architectures and seem to be an improved solution for scalability, while less prone to single point of failure problems. P2P architectures are intrinsically capable of large scaling unlike client-server architectures, as new users bring with them new hardware capable of storage and streaming, thus introducing higher storage capacity and streaming bandwidth into the network. High scalability in this architecture is feasible, but it does have drawbacks. Most significantly, decentralized architecture algorithms have to contend with a highly dynamic network where peer status constantly changes and there are minimal service guarantees to users.

Research on various systems for streaming media services has been performed and it is evident that a scalable, reliable, cost effective and high quality media streaming network is achievable dependent on the requirements of the user base it is developed for. In this dissertation, a P2P Video-on-Demand (VoD) Media Streaming System using STBs is conceptually designed, taking into account factors and protocols addressed in much of the researched material. A set of constraints are adhered to in the design of P2P VoD STB Media Streaming System, with the requirement that the system is developed for a large scale media streaming business. A private network of users is assumed where users are subscribers to a service provided by the business in question. This type of network could exist in a hotel, private community, hospital or any other residential area where people share a network.

The proposal of STB deployment in the private network stems from current research into solving reliability issues with peers in a decentralized architecture [7]. Recent systems have emerged with STBs allowing for more reliable decentralization of networks [8]. An ability to restrict the behaviour of users in a P2P network by means of a central authority is useful [9], as network

algorithms could be simplified and the unpredictability of network dynamics reduced. Thus, the proposed media streaming network in this dissertation includes a central authority, classing it as a 'hybrid' architecture. In theory, a 'hybrid' architecture is capable of alleviating scalability and economic issues experienced with centralized and decentralized architectures as suggested in Laoutaris *et al* [7].

Included in the proposal of the P2P VoD STB media streaming system are streaming protocol choices in terms of replication, authoritative network figures, video encoding, bandwidth restrictions, network monitoring, router buffering protocols, video segmentation, initial buffering, storage, popularity, churning, congestion handling and serving peers. Inclusion of these protocols aims to meet the constraints placed on the network design and improve media streaming quality of the network.

Currently, most methods of network evaluation are Quality of Service (QoS) based. Media streaming services have to provide QoS guarantees to users in the same fashion as Internet Service Providers. In multiple works researched [9, 10, 11, 12, 13], system performance is gauged in terms of client request rejection probability, a QoS based technique. QoS techniques for evaluation are effective in determining network statistics, but do not always give the best indication of an end user's perception of received media content [14]. End-user perception, which can be classified as a Quality of Experience (QoE) technique, provides a good representation of overall network performance. A new methodology of media streaming system evaluation based on QoE techniques is presented in this dissertation.

Using the Network Simulator 2 (NS2), the EvalVid framework for network video transmission evaluation and by use of Peak Signal to Noise Ratio (PSNR) measurements, this dissertation discusses development of a Quality and Scalability Quantification System (QSQS) for QoE evaluation of media streaming networks. Integration of NS2 into the EvalVid framework is adopted from Chih-Heng *et al* [15]. Progression with the use of NS2 is discussed with subsequent understanding of network simulation applied to the research. To qualitatively validate the initial simulation results, setup of a physical central network is first performed. This allows for a gauging of expected results from simulation.

To achieve the task of quantifying scalability of centralized and decentralized network architectures, the QSQS was employed in the evaluation of two basic comparable media streaming architectures. Results achieved indicate that the decentralized architecture scales better than a centralized architecture, but degradation in performance is observed as network load increases further. Evidence of a 'central nature' emerging in the decentralized system is discussed.

Furthermore, the proposed quality evaluation system is employed to evaluate a serving peer protocol, developed to alleviate the problem with the decentralized architecture under very high loads.

Results obtained exhibit the ability of the P2P decentralized architecture to scale up with basic software protocol deployments. In the context of this dissertation, the ability of the proposed quality evaluation system is demonstrated with the successful testing of this protocol.

A discussion of all results is performed thereafter. From the work performed and many of the researched areas in this field, the potential of the quality evaluation system can be gauged. Future work in this area is discussed, with an insight to the capability of the proposed quality evaluation system to form a new paradigm for media streaming system evaluation. QoE based methods are effective in assessing realistic network performance and ensuring user experience with their deployment in the field of media streaming.

1.2 Contributions

Research presented in this dissertation commenced on 14th September 2012, under the supervision of Dr Leigh Jarvis and was completed within a year. Development of the Quality and Scalability Quantification System (QSQS) for media streaming architectures was an original contribution to the field of streaming evaluation methods. The system provided a substantial quantification of scalability variations in centralized and decentralized media streaming architectures. This research resulted in the production of a journal paper titled "Quantifying Scalability of Decentral Media Streaming Networks over Central Networks, including Empirical Evaluation of a P2P Protocol, by use of Peak Signal to Noise Ratio". This paper was submitted to IEEE Transactions on Circuits and Systems for Video Technology and is currently under review. It is attached in Appendix B.

1.3 Dissertation Outline

Chapter 2 presents a literature survey which briefly discusses the body of research gathered for the work performed in this dissertation. Various media streaming architecture designs including associated streaming methods and protocols for P2P architectures are discussed. An assessment of evaluation methods of media streaming architectures is presented.

Chapter 3 details a conceptual design of the P2P VoD STB Media Streaming Architecture. Comparisons between streaming architectures are highlighted, with an explanation of all choices made in development of the hybrid architecture.

Chapter 4 explains the choice of NS2 and its associated characteristics and capabilities. The manner in which simulations are performed is addressed including demonstrations of Tcl scripting of basic networks. This chapter concludes with observations of the physical streaming media implementation.

Chapter 5 describes the development of the QSQS using EvalVid and NS2. Discoveries made by experimentation with the quality evaluation system are discussed, followed by a methodology for evaluation of media streaming architecture scalability and quality. Discussion of the serving peer

protocol follows including its associated evaluation methodology.

Chapter 6 presents an examination of results obtained from proposed evaluation methods, including insights into centralized and decentralized architecture performances. Analysis of the serving peer protocol is also discussed.

Chapter 7 concludes the work performed during the course of this research, with a vision of future research in the field of media streaming described.

Chapter 2 Literature Survey

2.1. Media Streaming Solutions

From the beginning of the 21st century, streamed media has grown in popularity worldwide and is currently responsible for the highest traffic on the internet [1]. Primarily, this is due to high bandwidth and storage requirements of video data. Due to the hardware intensive nature of media streaming, a wide range of architectures and technologies emerged in the last two decades attempting to serve users efficiently delivered, high quality videos. As demand for video increased, media streaming systems had to constantly adapt and grow to supply users' demand. An increase in video quality, with the introduction of High Definition for example, meant significant increase in video data size, leading to higher bandwidth requirements for streaming. Popular videos also placed stress on the streaming system as more users would need to be served. A central client-server streaming architecture was the first to emerge as a streaming system. Videos were delivered directly with minimal delay. However, this system was flawed and other architectures such as P2P overlay networks, CDNs and Pure P2P networks developed as replacements or alternatives to the traditional client-server architecture [3].

2.1.1 Client-Server Architectures

The simplest solution to media streaming is employment of a central server which downloads videos directly to clients upon request. Figure 2-1 depicts the basic structure of client-server architecture.

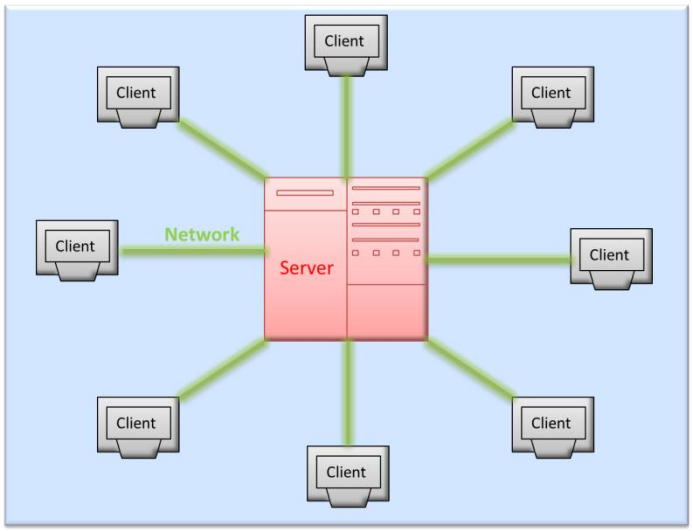

Figure 2-1 Basic Client-Server Network

The download bandwidth of the central server is a system bottleneck as more users join the network. Considering the large size of video data, an increase in demand by a modest number of requests can cause the server to fall over and result in general degradation of quality across the network. Limits of client-server systems are well documented in many of the consulted works [3, 4]. The issue of an overwhelmed server due to a high volume of user requests is discussed in Chou *et al* [4], citing the "flash crowd" that occurred on September 11, 2001 which caused a news server to crash. Many popular live events can cause severe issues with central servers which become overloaded when their hardware is not provisioned for very high loads. A solution to this problem was firstly over-provisioning, where servers were designed to function at a fraction of their peak for most of the day [3]. However, the high cost and inefficient use of hardware during off-peak times made this approach undesirable for media streaming. A number of papers focus on the lack of scalability with client-server architectures for media streaming [3, 4, 16, 17, 18]. To increase the scalability of these systems, a new type of application layer protocol, a central backbone P2P overlay network, was designed to help the server cope with higher loads [4].

2.1.2 Central Backbone P2P Overlay Networks

The tree-like formation of nodes establishing an overlay network atop the central backbone network is illustrated in Figure 2-2. The central server links to all nodes either directly or indirectly, dependent on network demand. [4]

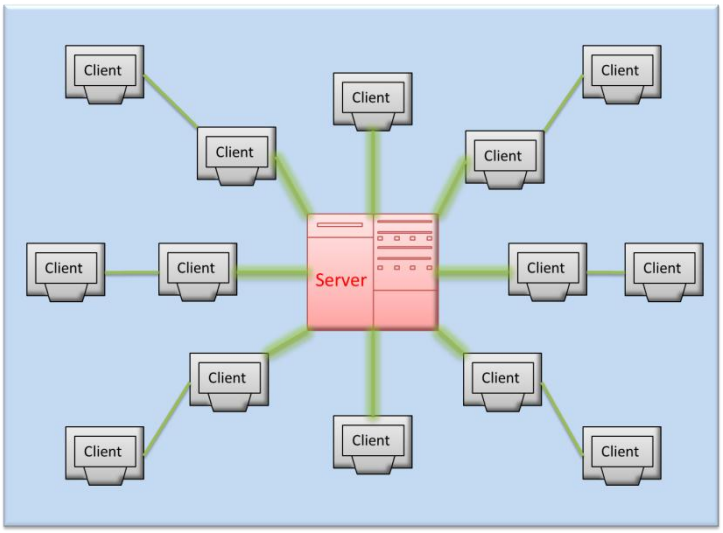

Figure 2-2 P2P Overlay Network on a Central Backbone Network

When a client requests a video, that video being streamed to the client can be cached by the client. When another client comes onto the network requesting the same video watched by the previous client, the new client can then access the cached copy of the video from the first client through redirection from the server. The new client is best served by the client with the cached copy as opposed to the server as the clients are more distant to the server in the tree-like structure than they are to each other. In this way, the server has to serve one request instead of two. Cooperative Networking [4] and DirectStream [10] suggest this method known as a P2P overlay network. It reduces server load and operates in a more scalable way than traditional client-server architecture. In Yadav *et al* [19], a comparison between centralized and decentralized overlay networks is given. Findings suggest that overlay networks suffer from issues such as single point of failure problems, high cost factors and increasingly dynamic client accessibility problems, amongst others. Other overlay networks include SpreadIt [20], which uses client resources for streaming high bandwidth video. Scalable Resilient Media Streaming [17], which employs application layer multicast, is another form of overlay network. With high network loads, quality of streaming degrades in overlay networks, albeit at a reduced rate compared to traditional centralized architecture [17]. Overlay network topologies have been in constant development but still employ the use of a central server, which remains a drawback to high scalability. An alternative approach to overlay networks was the use of CDNs for streaming media, alternatively referred to as CDNs.

2.1.3 CDNs

Figure 2-3 shows how server duplication is implemented to serve wider user bases. A central server feeds many smaller servers placed at the edge of the network, 'edge' referring to the location of

users in the network. The 'edge' servers are basically smaller duplicates of the main server designed to handle smaller user bases.

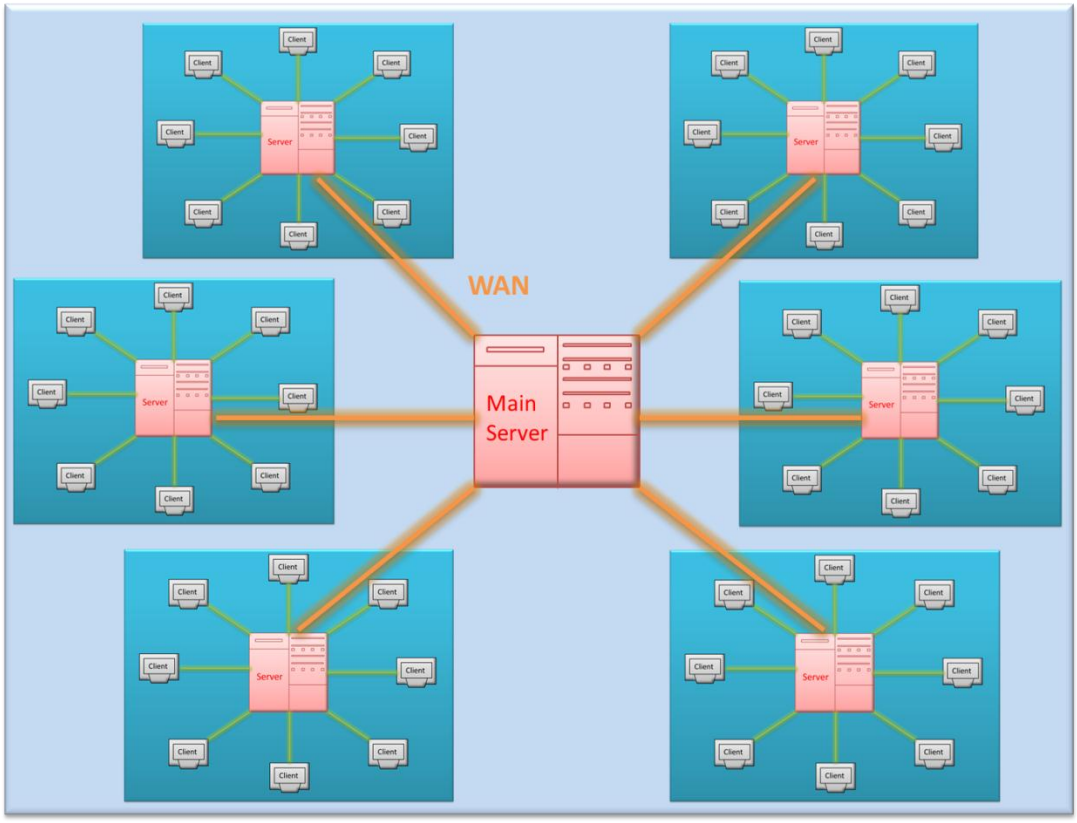

Figure 2-3 CDN consisting of multiple client-server architectures

An increase in client population demands more servers be erected to supply increased demand. Placement of 'edge' servers and determination of appropriate sizes are complex tasks, as it is difficult to optimally place servers for efficient and good quality distribution. A genetic algorithm for Server Location and Storage Allocation is presented in Sun-Jin *et al* [21]. CDNs are usually very expensive, and much research is dedicated to optimization of network resources based on cost analysis [21]. However, CDNs provide high scalability and high quality. A multiple server network, proposed in Peng *et al* [18], presents a hybrid P2P and Central network. This gives rise to much of the work presented later in this dissertation. Importantly, the recent introduction of Cloud Computing is based on the use of CDNs as suggested in Xiong *et al* [22] and Khazaei *et al* [23]. These CDNs are usually spread across multiple servers in various parts of a wide network to serve very high demands. Although this approach effectively solves the issue with scalability in media streaming networks, it still requires use of high cost and high power usage servers. Damaging to the environment and high maintenance, an alternative to CDNs is needed as they are not optimal.

2.1.4 P2P Networks

P2P file sharing networks like Gnutella, Freenet and BitTorrent have been in existence for many years. These networks formed the basis for P2P (P2P) media streaming systems as suggested in Stais *et al* [5] and Berkes [6]. A decentral system's intrinsic ability to grow with an increasing user base, as content is shared amongst peers, gives it a distinct advantage over the centralized architectures discussed above. A basic P2P architecture is illustrated in Figure 2-4.

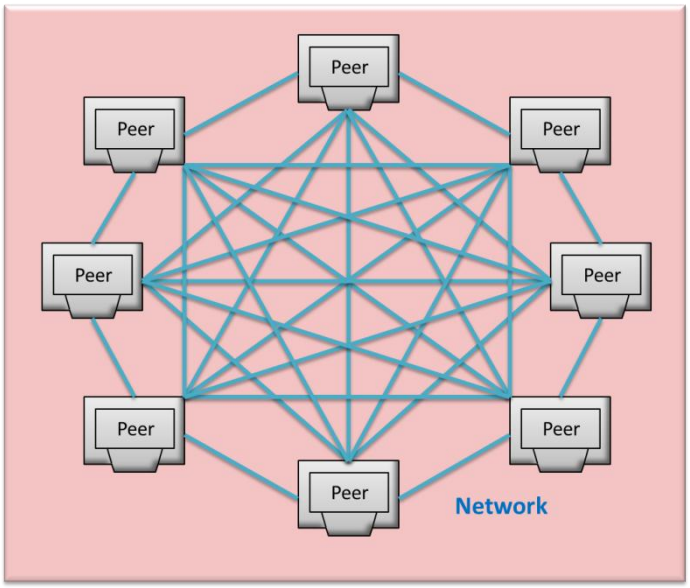

Figure 2-4 P2P Architecture with all nodes linked directly

Currently, many forms of P2P media streaming networks are in existence and this area is of great interest to researchers. A P2P media streaming system relies only on the storage and upstream capabilities of all peers in the network. When a peer requests a video, various portions of the video stored in different peers are streamed to the requesting peer. One of the earlier P2P systems was CollectCast [11]. This system operates at the application level and infers the underlying network topology and performance. Peer status and connections are monitored and the system reacts dynamically to connection failures or reductions in uplink bandwidth by changing to other active peers on the fly [11]. Coordination of network connections is performed through hashing techniques or central administration, discussed in section 2.1.5.3. It can be seen that network dynamics of P2P systems are a source of much overhead and this is one of the main issues with implementation of reliable decentralized P2P architectures [7, 24].

P2P systems are usually divided into two categories: P2P Live and P2P VoD. Network algorithms for live systems are different to those for VoD systems. CollectCast is an example of a P2P Live system. In contrast to this, P2Cast [12] is a P2P VoD system which uses patching

techniques amongst peers to cooperatively stream video. Decentralized Media Streaming Infrastructure, DeMSI [25], effectively portrays benefits of the division of responsibility principle applied in a decentralized media streaming network. Application of this principle to network bandwidth and content storage results in a growth of network resources with the introduction of an increased number of users [25].

In more recent times, P2P TV systems such as SopCast and Internet based systems such as Internet Protocol Television (IPTV) [26] have been implemented. These systems serve very large user bases and continue to grow maintaining good QoS. In light of the difficulty and high overhead in handling changing peer status and other network dynamics in P2P networks, methods for controlling peer behaviour have been implemented. Perhaps the most effective method for performing this action is the introduction of network controlled STBs placed at the user's premises. Peer assisted VoD for STB based IP network is highlighted in [27]. IP enabled STBs are becoming key devices in home media streaming. The use and performance of P2P systems employing STBs is discussed in Nafaa *et al* [8], Suh *et al* [9] and Laoutaris *et al* [7], which discusses ECHOS, edge capacity hosting overlays of Nano Data Centers. ECHOS is one of the earlier works to propose the innovative use of STBs for media storage and streaming.

Many P2P networks for media streaming exist but improvements to these systems can be made in various ways. Streaming protocols serve different purposes in these networks, with performance, overhead, and scalability forming the key motivators for development of these protocols.

2.1.5 P2P Protocols

The complexities of network dynamics in decentralized systems have led to the development of a number of different protocols. Understanding the development of these solutions entails understanding drawbacks to decentralized architectures adopted for media streaming. Servers in centralized architectures are characterized by large server storage capacity and high download bandwidth capabilities. Decentralized systems, however, do not have high resource capabilities and gain their hardware resources from a combined system of multiple nodes. Therefore, each node on its own has limited storage and streaming capability and must be well managed to ensure proficient use of all available bandwidth and storage. The most efficient use of limited resources is the foundation of some research in this field [28]. As a result of hardware limitations, many problems arise with P2P streaming architectures. Unpredictable peer behaviour ('churning'), node failure, over-demanding of popular contents, lack of content availability, network congestion due to high overhead, long video buffering times, etc. are some of the concerns that are prevalent in these systems.

Video Segmentation and Initial Buffering

Due to limited storage capability in nodes of a P2P system, complete videos with considerable size cannot be realistically stored for upload in each node. Instead, videos are segmented and stored in different nodes, thereby sharing the high storage requirements. Segmentation is adapted from file sharing systems like BitTorrent [5]. Video fragmentation into complementary sub-streams ensures that the video is spread across the network [25]. The level of segmentation is highly dependent on various network factors and hardware capability of the nodes themselves. The extent to which videos are segmented is the focus of much research. Video segmentation optimization is discussed by Sato *et al* [29]. Setting the video segment length to the inverse of the arrival rate of requests in a P2P system maximizes the effectiveness of segment accumulation amongst receiving peers [29]. Large segment size compared with the average request interval lessens the effectiveness of traffic reduction. Large segment sizes also run the risk of taking too long to download to the requesting peer's cache. Alternatively, too small segment sizes introduce unnecessary protocol overhead of P2P operations in the network. Video segment size could also be determined by the average video length and number of users in the network [29].

Research on placement of video segments in the network has proven to be largely an optimization problem, with a number of different techniques, each with their own merits. One proposal, when dealing with an STB network, is the suggestion that splitting the video equally amongst the number of STBs in the network is most effective. However, an issue that exists with this is STB failure, which will cause pieces of the video to become unrecoverable [30]. This is an important consideration in the P2P VoD STB network proposed in Chapter 3.

Video buffering techniques form another important part of P2P systems. Storing the initial segment of each video in a library in each node is a useful technique employed in some systems [12], as it ensures that there is no initial buffering delay whenever a user requests a video. VuDu [8] is a streaming system that uses initial content storage. This technique does however have its shortcomings, as it will require greater hard drive storage capacity on each node, especially for a large video library. An alternate approach often used is the initial delay buffering technique [31]. This allows video segments to be downloaded while the user waits for a calculated delay period after the request to ensure that no delays occur later during playback.

Network Coding and Video Compression

As stated earlier, maintaining the best utilization of limited storage and uplink bandwidth in a P2P network is the source of many protocols. Network coding to ensure efficient transport of video segments has been in steady development throughout the existence of P2P media streaming systems.

A network coding solution on P2P streaming media on demand system [32] discusses the negative effect on QoS that limited network resources have. The solution presented in Huang *et al* [32] reduces the download time of peers. It is stated in Huang *et al* [32] that P2P networks are characterized by "fast change network status, uneven and limited network bandwidth and service capabilities of peers, often high system failure rates of downloading, slow and uneven downloading speeds from different sources and limited upload capacity of the peers."

Use of compression technologies for video data compression like MPEG or H.264 can reduce network traffic [2]. Scalable video coding (SVC) [33], an extension to the H.264/AVC format, "enables the transmission and decoding of partial bit streams to provide video services with lower temporal or spatial resolutions or reduced fidelity while retaining a reconstruction quality that is high relative to the rate of the partial bit streams". SVC provides functionalities such as graceful degradation in lossy environments as well as bit rate, format and power adaption.

Network coding [32] is a paradigm shift to allow coding at intermediate nodes between the source and the receivers. In random linear network coding, the output encoded data piece is a linear combination of the input pieces. The pre-processing of streaming media data mainly includes two steps. Firstly, the original streaming media data is divided into data blocks with fixed size. Secondly, each block is divided into smaller pieces with equal size. Blocks are segments of the original streaming media which can be downloaded and played independently by client peers. The pieces in a block are the smallest unit of network coding.

Multiple Description Coding (MDC) [19] is a technique used where many streams of the same content are served to a peer. A higher number of streams received by a requesting peer translates to higher quality video decoding. The diversity of this method ensures that there is almost always some content received. This technique is very useful in lossy networks like the Internet. However, transmitting an increased number of streams can become a problem with very limited bandwidth resources.

Hashing Techniques and Central Administration

Successful operation of a decentralized P2P architecture is dependent on administration that keeps track of all video segments and peers in the network. Each peer needs to know where to request videos from, and sending peers need to know where to send requested video segments. Information on storage allocation, bandwidth utilization of the nodes and various other network metrics need to be dynamically tracked.

MediaDART [16] adopts a Distributed Hash Table (DHT) which provides resource storage and parallel resource processing for operations of feature extraction, adaptation and composition. Many DHT designs exist such as Pastry [34] and Kademlia [35] or DHT with Session Initiation Protocol [36]. Pastry comes with application level services for building distribution trees and

managing the persistence and copies of resource records. Kademlia adopts a routing table structure that makes it easier to compute global network properties from statistics extracted locally. Its basic operations use an iterative, asynchronous algorithm that can be optimized to have very short response times. Systems such as CollectCast [11] and DeMSI [25] utilize DHTs to collect network statistics and make inferred decisions based on network conditions. Methods of tracking peer information are discussed further in Chapter 3, including a decision on adoption of a certain protocol for the P2P VoD STB media streaming system. An important aspect of DHTs is that their implementation must be lightweight so as not to increase network overhead too significantly by their deployment.

An alternative approach to DHTs which must be implemented in each node, is the use of central administrative points in a decentralized network. In terms of an STB network, it is useful to allow the content provider some central control over the network. A Control Server or system administrator would reduce the load on the peer nodes of a decentralized system allowing greater capacity and processing for video encoding and transmission. Push-to-Peer VoD System [9] suggests the presence of a Control Server at the user premises. This server provides a directory service to boxes in addition to management and control functionalities. A content server discussed in Kin Wah Yim *et al* [25], also referred to as a cache or backup server, can be introduced into a P2P network to ensure that all content in the video library is always accessible, compensating for an event such as multiple node failure where many video segments will be unavailable.

These methods provide stability to a P2P network ensuring that it can always maintain a high QoS.

Prefetching Content, Replication and Popularity Based Techniques

Placement of content on network peers is a fairly complex task. Decisions on where and when to upload video segments to peer nodes are very often made heuristically. Depending on the choice of network, videos can be pre-fetched through streaming from a CDN to the content server in a private network. It can also be a case of uploading videos to STBs in a private network through a server located within close proximity to the STBs. When pre-fetching is performed, it is usually done during times of minimal network load. The number of copies fetched from the server can be formulated as an optimization problem [37].

Deciding on the number of copies to fetch implies the use of content replication techniques in P2P systems. Streaming capacity in a P2P VoD system is defined as the maximum streaming rate that can be received by every user. He *et al* [28] suggests that streaming capacity is limited by over-demanded video segments. The streaming capacity problem is formulated into an optimization problem which maximises the streaming rate subject to peer bandwidth constraints, and solves it with a distributed algorithm [28]. In He *et al* [28], Sethi *et al* [38] and Munoz-Gea *et al* [13], amongst others, it is suggested that the most effective way to prevent any single peer node from becoming

overloaded is through content replication. Bounced [38] is a replication strategy that intelligently replicates files on-demand to improve availability of scarce files in P2P networks. Serving peers, or 'helpers' [28], are peers who are willing to contribute their remaining upload bandwidths to help other peers. Helper assignment and rate allocation are optimized in He *et al* [28] to improve streaming capacity. It is found that maximum streaming capacity can be obtained in a distributed manner by optimizing resource allocation in the P2P VoD system [28], that is, through the use of a Helper protocol. Munoz-Gea *et al* [13] also discusses an optimization strategy for replication in P2P networks and the use of serving peers. This protocol is the basis of much work presented later in this paper.

As mentioned above, over demanded content causes reductions in maximum achievable streaming capacity. It must be remembered that peers in this state are forced to work as small servers which is more than their hardware is provisioned for. Research on video content popularity is widespread, as popular videos are most often the reason for over demanded content. Video popularity is addressed in Bermudez et al [26], Fujimoto *et al* [39], Wu *et al* [40] and Nafaa *et al* [8].

In Bermudez *et al* [26], a good indication of some of the current successful businesses borne out of P2P media streaming is given, such as SopCast. A novel approach to analysis of graph properties and the traffic generated by P2P-TV applications run by customers in operative networks is illustrated in Bermudez *et al* [26]. Importantly, collection of real network usage data over one year for a P2P-TV system is performed and analysed. Research has revealed that P2P-TV usage is discontinuous and associated to events that are popular but expensive to retrieve through normal TV broadcasting systems as monthly subscription fees are generally high [26]. Understanding user behaviour in P2P systems is important as it allows for an assessment of demand for popular content and planning for replication methods to supply that demand.

Considering that over-demanded contents become P2P system bottlenecks, it is often suggested that contents should be replicated proportional to the popularity of the video. Several recent papers have analytically demonstrated that the optimal number of replicas of each video must be proportional to its relative popularity within the content library [13]. Other works have reported that the proportional replication may have poor performance for unpopular movies, and for this reason the replication strategy should be greedier for this kind of content. It is suggested in Munoz-Gea *et al* [13] that due to diverse user preferences, it is necessary to pro-actively adapt the number of replicas to video popularity, which entails monitoring and storing user requests in a network. A more recent exploration on the optimal replication strategy in P2P VoD systems [40] asks: "What is the 'optimal replication ratio' in a P2P VoD system such that peers will receive service from each other and at the same time, reduce traffic to the content server?" It is formally shown how video popularities can affect server workload, and video replication is formulated as an optimization problem. Interestingly,

it is shown in Wu *et al* [40] that the conventional way of replication proportional to popularity is not optimal. Instead, it is shown that replication proportional to the deficit bandwidth is optimal [40]. Thus, replication strategies should be greedier for unpopular movies. The design space is expanded in Wu *et al* [40] by considering both passive and active replication strategies. A passive replacement algorithm to decide which video to purge when a local storage is full is presented. Also proposed is an active replication algorithm to aggressively push data to peers so as to achieve the desired replication ratios [40].

As can be gauged, video content popularity and replication in P2P streaming media systems is a current active research topic and is used partly in the design of the proposed P2P VoD STB media streaming system proposed in Chapter 3 of this dissertation.

Transport Protocols and Congestion Control Strategies

User Datagram Protocol (UDP) transport mechanisms are used in most media streaming systems which exist in all Internet Protocol based networks. Transmission Control Protocol (TCP) is used as the facilitating mechanism between nodes of the P2P systems. The introduction of Real-Time Transport Protocol (RTP) has grabbed much attention and is used in most mainstream media streaming systems over the internet [41]. RTP provides end-to-end network transport functions suitable for applications transmitting real-time data such as media streaming applications. The data transport is augmented with the RTCP control protocol to allow monitoring of the data delivery in a manner scalable to large multicast networks [42]. These transport mechanisms for media streaming are the subject of much deliberation and are constantly being improved upon. P2P media streaming systems are often hampered by congestion at certain points in the network which can significantly reduce QoS in the network.

Chaomei *et al* [43] suggests a route queue management scheme based on end-to-end congestion. TCP Friendly Rate Control (TFRC) and Mixed-Driven Congestion Control (MDCC) [43] are route queue management schemes. TFRC works as follows: UDP transmits streaming media because of its low latency and smooth transmission rate. However, UDP, which lacks congestion control, cannot compete with TCP flow for bandwidth. Once congestion happens, TCP will use its congestion control to reduce the flow rate, but UDP flows will occupy the TCP's flow bandwidth resources, which make congestion difficult to resolve effectively, thereby seriously affecting the quality of streaming media. TFRC adjusts the sending rate to control congestion based on the TCP throughput model equations [43]. Thus congestion is relieved and QoS in the network improves.

DeMSI [25] uses congestion points in the network to employ new active peer sets. Congestion remains an important form of real time network monitoring in many systems. Detecting Shared Congestion of flows via end-to-end measurement [44] is another congestion control strategy

researched. This is employed in Internet based networks where cooperative congestion control strategies can improve performance for many applications. Its benefits in media streaming are alluded to in Buyya *et al* [25].

2.1.6 Media Streaming Network Analysis and Evaluation Methods

Many of the protocols discussed above improve performance of the architectures they are implemented within. However, it is difficult to gauge their impact without proper analysis and evaluation. System modelling is the first important step towards accurate evaluation of networks. This view is supported in Xiong *et al* [22] and Khazaei *et al* [23]. Improving cloud computing service performance can only be achieved through modelling, evaluation and benchmarking of current generic systems. In cloud computing, it is important to provision the right amount of resources required. This can only be achieved through accurate performance evaluation as Khazaei *et al* [23] necessitates. Similarly, the need for accurate performance evaluation and system modelling is critical for furthering the development of media streaming systems.

Network QoS based performance evaluation of media streaming networks is a very popular method used in the assessment of most of the systems researched. An important and very commonly used performance metric is rejection probability. This is the probability that a client's request will be blocked in a network based on various factors. Such factors include system workload, number of peers in the network, peer cache size, server load, and many others. DirectStream [10] performance is measured using overall rejection probability versus workload, and server stress against workload. CollectCast [11] employs frame level performance analysis and packet loss rates in the network. Complete mathematical models of active peer sets and rejection models are developed in Hefeeda *et al* [11]. Similarly, the performance of P2Cast [12] is measured using client rejection probability, average network load and server stress. Push-to-Peer VoD system [9] is evaluated using rejection probability set against request arrival rates.

In analysis and evaluation methodology as well, rejection probability is used as the main measure of performance. Munoz-Gea *et al* [13] exhibits the use of mathematical models to describe the behaviour and performance of decentralized media streaming architectures, again using rejection probability as the main indicator. This is used to measure the accuracy of the strategy itself in determining network performance.

The characterization of demands and optimization of supplies [45] uses blocking probability (rejection probability) system wide versus peer cache size and system workload to determine optimal points of performance. An analysis of large scale P2P VoD architecture [8] is performed mathematically with the intention of minimizing rejection probability. A parameter 'alpha',

representing popularity distribution, is used to maximize network resource utilization by reducing the overall number of VoD request rejections.

Media streaming systems focus on user experience and are driven this way in terms of performance as opposed to a network performance driven approach. Packet loss rates and rejection probabilities give an accurate view of network conditions. However, these do not give a strong indication of what is viewed by the end-user at the edge of the network. Currently, most performance evaluation methods are network QoS based and not QoE based. A media streaming system, in addition to network metrics, should be evaluated using video quality metrics. Use of PSNR for measuring quality of video passed through a network is proposed in Chih-Heng *et al* [15]. A toolset based on Evalvid for evaluating video quality is developed using NS2 as a replacement to the simple error simulation model used in Evalvid. This toolset enables the investigation of the relationship between two popular objective metrics for QoS assessment of video quality: PSNR and the fraction of decodable frames [15]. A complete discussion of this toolset is presented in Chapter 5 of this dissertation, with modifications made by the author to introduce a new method of evaluation for P2P VoD media streaming systems.

NS2 is used in the simulation of most of the media streaming systems discussed thus far. DeMSI [25] is a P2P media streaming system which served much of the stimulus for the research performed leading to this dissertation. NS2 was used extensively to model and evaluate the performance of DeMSI and the use of NS2 to perform simulations in this dissertation is fully discussed in Chapters 4 and 5.

Chapter 3 P2P VoD System Proposal

In the development of any model or system, constraints are placed depending on the purpose of the investigation. In this chapter, a high level design of a media streaming system is proposed. A decentralized video delivery service system operating in a private network is conceptually designed with the ability to be highly scalable while maintaining high quality media delivery. The system must maintain good reliability and ensure that all users in the network enjoy a high QoS and experience.

3.1 System Constraints and Assumptions

3.1.1 Running under the control of a Media Streaming Organization

It is assumed that the system is developed for an organization that provides streaming services to users for a service fee. This entails maintaining economic efficiency during development and growth and that content is published onto the network by the organization.

3.1.2 Network Size

Extent of the organization's business model is significant in determination of the number of users to serve. Under the assumption that the network is private, it is thought that this system would be operable in a small community of users such as in a hotel, hospital, holiday resort or gated community. Addition of users to the network should be simple and not affect the quality of existing users in the network.

3.1.3 User Control of Services

Maintaining organizational control of the network to ensure highest efficiency of network resource utilization is paramount. Thus, users should only be afforded minimal control of network

resources. User capability should be limited to media selection from a library and basic control over said media.

3.2 System Topology

3.2.1 Comparison of Media Streaming Network Architectures

The benefits and drawbacks of centralized and decentralized architectures have been alluded to earlier in the dissertation. Researching the various works on media streaming architectures in existence, it is apparent that the development of a robust, reliable and scalable architecture involves exploitation of advantages of the various architectures. Table 3-1, adapted from Laoutaris *et al* [7], depicts the difference between traditional Central, Central Overlay, P2P and CDN architectures.

Table 3-1 Comparison of Basic Architectures in Media Streaming [7]

Architecture	Central	Central Overlay	P2P	CDN
Data Plane	Centralized	Centralized with some distribution	Distributed	Centralized
Control Plane	Centralized	Centralized	Distributed Uncoordinated	Centralized with Partial Distribution
QoS	Guaranteed	Guaranteed	Best Effort	Guaranteed
Distance from End-User	Large	Large but dependent on Load and Content	Small	Large but shorter than Central
ISP Friendly	Yes	Yes	No	Yes
Security	Strong	Strong but with weaknesses	Weak	Strong
Capital Requirements	High	High	Low	Very High
Scalability	Very Low	Low	High	Low-to-High (Capital Dependent)

Analysis of Table 3-1 indicates that none of the architectures can individually satisfy all the constraints highlighted in Section 3.1. It is apparent that a hybrid version of the various architectures presented in Table 1 is required. Considering the constraints in Section 3.1, system requirements are proposed in Section 3.2.2, Table 3-2.

3.2.2 Hybrid Architecture Specification

Table 3-2 Hybrid Architecture Requirements

Architecture	Hybrid Network
Data Plane	Distributed
Control Plane	Distributed Coordinated
QoS	Guaranteed
Distance from End-User	Small
ISP Friendly	Yes
Security	Strong
Capital Requirements	Low to Medium
Scalability	High

To achieve requirements set out in Table 3-2, modification of a CDN is required to employ a P2P architecture at the edge of the network, i.e. at the user premises. This is an idea shared in Laoutaris *et al* [7] and Chellouche *et al* [24]. It is suggested in Chellouche *et al* [24] that intense demand, in terms of server and network hardware resources, is currently placed on the Internet due to growing popularity of VoD services. Service scalability is thus a critical issue for which the solution is most often server replication. This technique is referred to as a CDN and, as mentioned in the literature survey, faces complex tasks in terms of server placement and selection in addition to high cost factors. CDNs have to be dimensioned for peak capacity, and this causes scalability and economic efficiency problems, especially when flash crowds or popular events are shown.

Thoughts in Chellouche *et al* [24] are shared by the author of this dissertation, in terms of the introduction of a P2P service layer at the user premises to alleviate some of the issues with CDN's. A CDN is operated by the media streaming organization. Traditionally, a host central server, maintained by the organization, feeds data to servers located at the edges of the network. These servers are then required to serve growing user populations. There can also be multiple layers of servers but this is arbitrary as it depends on the scale of network infrastructure and organization which provides these services.

Modification of the CDN in this dissertation occurs at the edge of the network, i.e. at the user premises. Instead of users feeding off a central server, which is normally fed content by servers higher up in the network hierarchy, users form a P2P network that operates within a certain range of the population or closed community as mentioned earlier. Figures 3-1 and 3-2 are an illustration of the concept of the hybrid architecture.

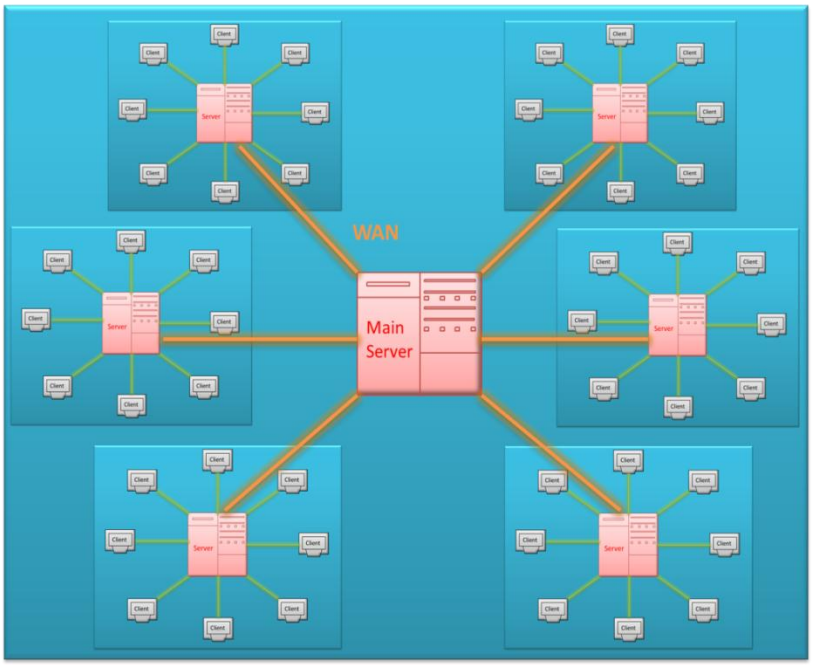

Figure 3-1 Standard CDN Architecture with Multiple Server Hierarchy

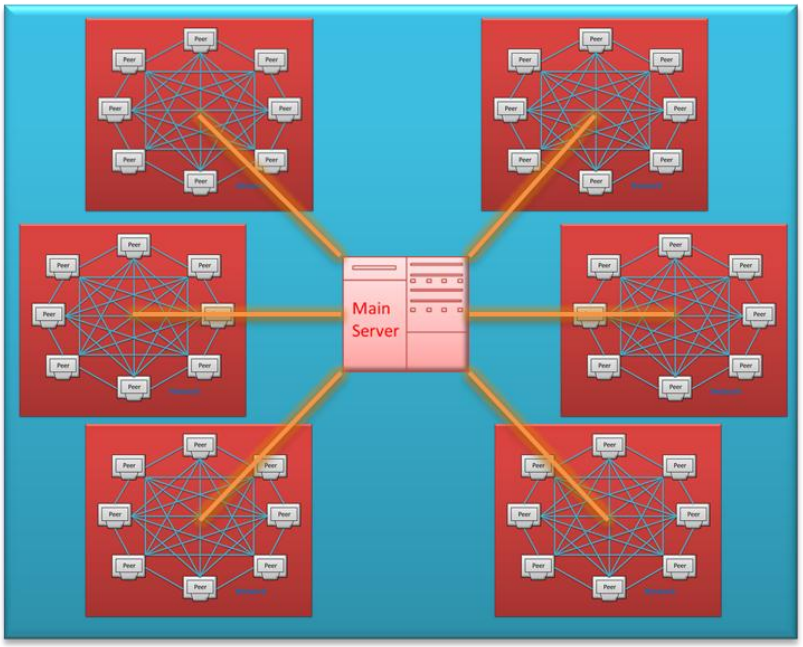

Figure 3-2 Hybrid CDN Architecture with P2P Networks replacing Multiple Servers

The CDNs illustrated in Figures 3-1 and 3-2 are similar in nature with one important distinction: the need for multiple layers of servers, as with a traditional CDN depicted in Figure 3-1, is due to load distribution and allowing for greater scalability across a wider range of the user population. A P2P network that draws content from the central server, naturally distributes the load of its user base owing to the network being self-serving. Unlike traditional CDNs which serve individual user requests via servers, the hybrid architecture depicted in Figure 3-2 requires a single request for data from the server. Thereafter, fetched content is distributed amongst the peer nodes at

the network edge by methods described later in this chapter. This in turn, places little load on the CDN itself, thereby reducing high cost factors and eliminating the need for complex server placement and selection algorithms.

3.3 STBs

The solution proposed in Section 3.2.2, termed 'hybrid' architecture, cannot achieve all of the objectives laid out in Table 3-2. As a result of the P2P network implementation at the lowest network level, i.e. at the user premises, unreliability and unpredictability of peer dynamics is introduced into the hybrid architecture. This is due to peers possessing the ability control uplink and downlink bandwidth and storage, as with users in a standard file-sharing P2P network, resulting in poor network management.

Managed boxes exist in many places on the Internet where broadband access is available [7]. It is suggested in Laoutaris *et al* [7] that these boxes, generally referred to as STBs in the media streaming group, can be used at user premises to implement a widely distributed service platform. It was found that Laoutaris *et al* [7] and Chellouche *et al* [24] further support STB introduction to allow management by higher authorities in the media streaming system.

Traditional P2P systems allow users a high level of control in the network as peers are capable of turning off their systems, leaving the network, downloading high volumes of data while uploading minimal data and having highly variable bandwidth allocation. Controlling user behaviour in P2P networks is the basis of substantial research, a subject mentioned in Ge *et al* [46] which discusses mitigation of peers downloading large volumes of data while minimizing upload volumes in torrent networks. Lack of network control through these peer actions reduces the reliability, quality and scalability of media streaming P2P networks.

Unlike traditional P2P networks, a system of STBs can be managed by the media streaming organization. Access bandwidth to the STBs and storage of contents can be facilitated and controlled by the organization. As a result, unpredictability and unreliability associated with P2P networks can be lessened. Introduction of STB P2P networks simplifies network dynamics and complexity of network algorithms.

Churning, which is the term that describes peers dropping off the network unexpectedly, is reduced with STBs, as peers are only required to keep the STB powered on for correct network functionality. The media streaming organization can offer free incentives to the consumer to keep the STB on, such as free access for limited periods or a free video.

Storage and bandwidth capabilities of the STBs themselves are dependent on the budget and size of the network in which they are implemented. In the simulation network setup section later in the dissertation, hardware dimensioning of STBs are discussed as network metrics are known.

3.4 Network Fundamentals

The size or extent of the user base served by the media streaming organization is dependent on economic issues and popularity of the organization brand. It is assumed that the organization has a public internet broadband connection between its central server, at its base of operations, and its multiple private P2P STB networks, located at the various paying communities. The private networks each have routers which link to the organization's main central server. Broadband charges are normally data dependent, which requires maintaining the least amount of data transfer between the central server and P2P networks.

The private network, assumed to be located in a closed community, maintains a broadband connection to the central server. It is then the task of the private network to filter received content from the central server to on-board storage of the STBs. The type of private network interconnecting STBs through the router is of interest. Currently, Gigabit Ethernet Local Area Networks are commonplace. Bandwidth requirements of the P2P network are important in considering the type of Local Area Network implemented. Importantly, to maximize economic efficiency and quality, the LAN should be optimized, with a trade-off between scalability and cost. Implementation of a Gigabit LAN requires more expensive networking equipment such as installation of Gigabit Ethernet Network Interface Cards in STBs, including high-end routers for gigabit network connections. However, if High Definition content, which requires relatively high bandwidth, is to be streamed in the network, the implementation of a Gigabit LAN would be sufficient as it would allow high network scalability while maintaining high quality content delivery.

Traditional P2P networks formed over the Internet are interconnected in an array with every node being able to connect to every other node either directly or indirectly. Figure 3-3 depicts this topology in comparison to the central-like topology, depicted in Figure 3-4, implemented in the private network proposed in this dissertation.

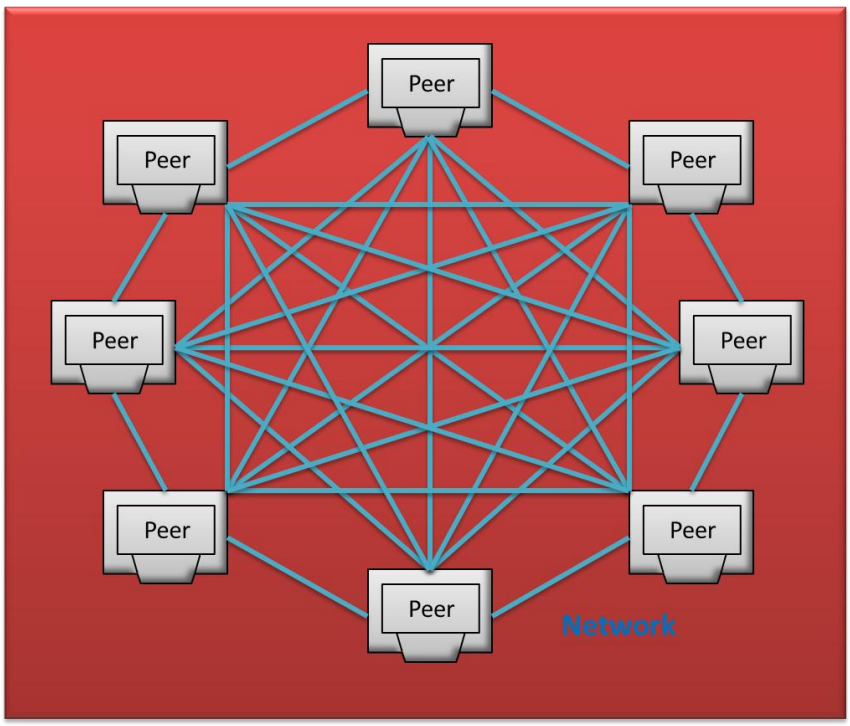

Figure 3-3 Standard Interconnection of Nodes in a P2P Architecture

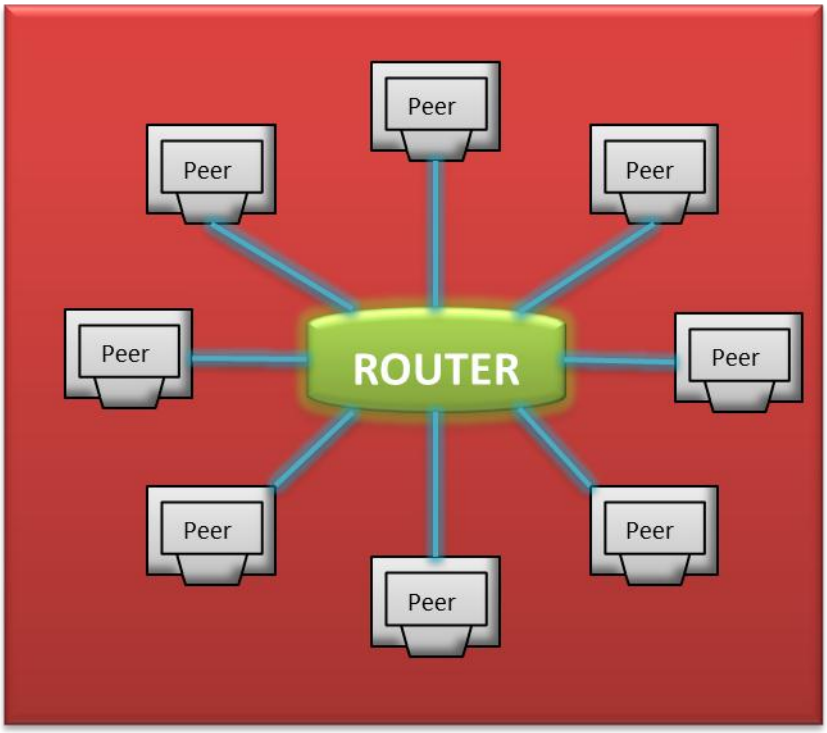

Figure 3-4 Central-like Linkage between Peers of Private Network

Within the closed community of users, all STBs connect to a router which serves as the central point in the network. Routers, including Gigabit LAN routers, are generally low cost hardware, with the task of equipment supply and maintenance left to the media streaming organization. Choice of router is network size dependent and there are limits on number of switches in a router. To solve this problem, in the case of an increasing user base, routers can be duplicated and linked which allows for unimpeded expansion in the network. Each STB is directly connected to every other STB, dissimilar to the way traditional P2P networks connect, as shown by Figures 3-3 and 3-4. Capabilities of the router in terms of queuing and buffering are discussed in the network setup section of this dissertation.

3.5 Central Authority and Distributed Hash Tables in the Private Network

Maintaining a dynamic database of video contents and peer bandwidth availability in the network is vital. A decentralized approach to control in a P2P network entails implementation of a tracker or Distributed Hash Table (DHT) in the network. This technique is one that is commonly used, as supported by Agelli *et al* [16], Yang *et al* [36], and Buyya *et al* [25] amongst others. A Distributed Hash Table or tracker maintains a table, or database, in each peer node of the network. A record of the peer status, peer bandwidth availability and available content is regularly updated in the network by each peer. It also tracks network behaviour and using congestion techniques, can detect whether a peer's upload capacity is saturated or is able to serve more content. Requests made by users are also kept track of by these methods. Maintaining this database in every peer introduces significant overhead in the network. As network node status constantly changes, every network node needs to be updated for each network change or periodically. This puts strain on the STB which uses up some bandwidth and processing power for these TCP connections.

An alternative way of controlling the network would be the introduction of a small network node dedicated to maintaining a dynamic database of peer status, peer bandwidth availability, user requests, content availability and other important network metrics. This could be referred to as a Control Server or central administration server [10] in the network, coordinating communication between all network STBs. When a user makes a request, it is sent to the Control Server, which references its database and instructs relevant peers with the required contents to serve the requesting peer. Implementation of a central Control Server in the network must not be confused with a typical server implemented in a client-server network. The Control Server requires minimal computing power and minimal storage space. It can take the form of an STB in the network dedicated to coordination. The problem with this technique is that it introduces a single point of failure issue, similar to that suffered in centralized networks. However, as cost of this hardware is minimal,

possibly the same as an STB in the network, it can be duplicated. Thus, the single point of failure problem can be eliminated with a backup Control Server.

Implementing a hybrid of the two control methods discussed is another possibility. If each STB maintains a 'lightweight' DHT, and a Control Server is employed to ensure that the peers of the network are working in correct order, this would ensure network functionality, including the case of a failed Control Server.

3.6 P2P VoD, Content Publishing and Backup Content Serving

It must be explicitly stated that the P2P network proposed in this dissertation is VoD. This means that users are offered a library of media content by the media streaming organization. P2P VoD systems have provided many interesting research avenues and the proposed network system can be converted to P2P Live if required, whereas conversion of a P2P Live system to P2P VoD requires many protocol changes and is a more complicated process. A distinct difference between Live and VoD content is that P2P Live tends to have many viewers watching the same content around the same time bracket. In contrast to this, P2P VoD viewers have varied viewing times for different content and this poses many interesting design and optimization problems.

In Section 3.4, the receipt of content in the private network from the central server through a broadband connection was discussed. This section focusses on the dissemination of that content through the private network. It is assumed that a large library of content is made available to users in the private network.

When content is sent to the private network, it must first be stored in a 'Backup Storage Server', which must not be confused with a typical centralized server. The Backup Storage Server is implemented in the P2P network to provide stability in the case where certain content may not be available from any of the STBs [25], in the case of a power failure for example. The purpose of the Backup Storage Server is two-fold. It has the additional feature of publishing new content onto network STBs. Fetching and publishing of content from the organization's main central server will need to be done at off-peak times, i.e. when network load is minimal, such as when people are asleep.

The Backup Storage Server is supported by the Control Server which coordinates its activities in terms of content publishing and serving of missing pieces in the network. It must be noted that the addition of a Backup Storage Server in the P2P network is for achieving network robustness and reliability. Implementation of this equipment is not dissimilar in nature to the implementation of the Control Server. Maintained by the media streaming organization itself, but installed at the location of the private network, the Backup Storage Server is similar to standard STBs, with the exception of

having larger storage capacity than an individual STB. It must be capable of storing every video in the library as a backup to network STBs.

The function of the Backup Storage Server and the Control Server differ in nature from that of a typical CDN central server such as the one present in the organization's base of operations. Whereas a central server is required to stream content to the complete client base for all requests on a full-time basis, the Backup Storage Server and Control Server distribute content to clients during times of low network load and do not serve any requests unless multiple STB failure occurs. Hardware requirements for the Backup Storage Server and Control Server are similar to that of an STB, and unlike the high bandwidth and storage requirements of typical CDN servers.

3.7 Router Buffering

Many buffering protocols exist, although Gigabit Ethernet LANs implement a First in First out (FIFO) Droptail queuing protocol. It is apt for the media streaming system as it simplifies design. Other protocols such as Stochastic Fair Queuing are not implemented in this system as FIFO offers the simplest solution around which other system protocols can be designed.

3.8 Transport Protocols

Content in the private network is transported between nodes using the RTP media streaming protocol which uses UDP as the transport protocol. RTP gives real time feedback on channel conditions and the state of transmission, making it highly suitable for media transmission. RTCP is the component of RTP which sends control and timing signals between the sending and receiving parties. Control messages from the Control Server, which include source and destination IP packet headers, timestamps and fragmentation information, are sent to the STBs using RTCP. The content itself is transmitted via RTP, which sets up dedicated transmission channels or data paths before communication. The proposed private network operates in Island mode, ensuring that the RTP protocol runs at optimal efficiency.

3.9 Video Segmentation, Initial Buffering, Storage and Caching

As mentioned in the literature survey in Chapter 2, video segmentation is an active area of research. Segmentation of videos is an important part of operation in the proposed private network. Adapted from file sharing systems such as BitTorrent, it allows the video to be available across the network and strengthens the P2P nature of the system. Strengthening the 'nature of the P2P system' refers to the principle of 'division of responsibility'. Dividing the video amongst STBs in the network

allows for even distribution in the data plane, thereby enhancing content availability in the network and making optimal use of available storage space on the STBs. Deciding on how segmentation occurs is an important part of the process.

The first decision is on placement of the initial part of every video sequence in the library, in each STB. The result of this storage is that whenever a user makes a request, playback begins immediately, as the remaining required video pieces are streamed from other STBs in the network. This results in no delays during playback and gives a time buffer against network faults. A drawback to initial segment storage is the amount of storage space wasted due to redundant segments being stored in every STB.

Employment of an initial time buffer after a user request is an alternative to initial segment storage. A short waiting period is imposed upon the user while the STB gathers required pieces to begin playback. This method contributes negatively toward network QoS, as delays in media networks are not ideal. Whereas initial segment storage has a higher QoS, initial buffering makes more efficient use of network resources in terms of storage.

A trade-off between QoS and efficient use of resources must be decided upon. Maintaining a balance between quality and economic efficiency (low cost) can be achieved by storing the initial segment of popular contents available in the library. Methods of popularity determination are discussed in Section 3.11. By storage in this manner, it is likely that most users, on average, will not have to wait a buffer period for content to be streamed to their STB. Only videos less requested will require an initial buffer period, which will be minimized by placing a high priority on fetching of the initial piece of that video.

Discussion of video segmentation and its attributes was performed in Chapter 2. Very high segmentation is plagued by high network overhead, whereas low segmentation results in parts of the video being too long, and users having to wait long periods to view content. A constant segment length can be determined, based on the number of videos available in the library, storage space of each STB and number of users in the private network.

The Control Server and Backup Storage Server are responsible for receiving content from the central server and segmentation of videos. This equipment is then responsible for disseminating the segmented pieces amongst the STBs in the private network. Each substreamed video is meant to be streamed by a different STB to the requesting STB. The objective of having each STB contribute is to maintain the division of responsibility principle, and in doing so, reduce the contribution, in terms of bit rate, of each STB to overcome the limited uplink bandwidth.

When video segments are received by the requesting STB, they are cached and kept for the duration of the viewer's request. Once a viewer makes a new request, the old video is then removed

from cache storage. The cache is cleared because the Control Server infers decisions about network metrics and optimally stores all library videos in the networked STBs' storage drives. Maintaining the cache storage of the requesting STB would thus be redundant and unnecessary as STBs making new requests would never need to access this temporary storage.

3.10 Multiple Description Coding and Video Compression

The purpose of Multiple Description Coding (MDC) is to make the most efficient use of limited resources available in a P2P network. Many streams of the same content are served to a peer, and as more streams are received, higher quality decoding can be achieved. This technique is not particularly suitable to the private network, as overhead is increased and the private network has sufficient resources to perform high quality transmissions.

SVC is an option for IP based networks as it maintains graceful degradation in lossy environments and has the features of bit rate, format and power adaption. However, the private network environment is expected to be efficient and have low losses, as it operates in Island mode, thus not requiring the need for multiple partial bit streams to provide high quality video.

Video compression methods are important as they reduce the size of the transmitted stream with a very minimal reduction in quality. The H.264 format is effective for video encoding and efficient streaming of encoded video.

3.11 Replication and Popularity

Storage of all videos available in the library is achieved through segmentation and placing of segmented contents into all STBs in the network. As a result of this technique, only one segmented copy of each video exists in the network. As discussed in Chapter 2, a widespread problem with P2P networks is the emergence of a 'central' nature in the decentralized architecture. This occurs when some video contents are over-demanded, or as a result of high network loads due to popular contents. 'Central' nature refers to some of the STBs having to perform like small servers, a task which they are incapable of, due to a lack of resources.

Replication is a strategy used to overcome limited streaming capacity of individual STBs. Content replication is achieved in many ways and is responsible for a vast amount of research as discussed earlier. Deciding which content to replicate can either be a random process or dependent on content popularity.

In the private network, video content will be randomly replicated. The Control Server will keep statistics of user requests across the network and decide heuristically on how to replicate content. Replication proportional to popularity has proven to be detrimental to unpopular videos, as requests in

this domain result in higher network stress [13]. The system proposed in this dissertation adopts a pro-active content replication policy, as discussed in Munoz-Gea *et al* [13].

A number of copies of the same video can exist in the network dependent on number of active network users. A higher number of network STBs implies greater network storage capacity, allowing more replication per video.

With an even distribution of replicated contents, a serving peer protocol is adopted in the private network. This protocol works as follows: when the streaming capacity of an STB is saturated, other STBs with the same content being served by the saturated STB, but with more available streaming bandwidth, are employed by the Control Server to assist the saturated STB. The STB with the least load is chosen to serve the requesting STB required content. This method is tested later in this dissertation with results showing an improvement in received video quality across the network under high loads.

The private network achieves maximum streaming capacity in the network in a distributed manner by optimizing the resource allocation in the system, a theory presented in Munoz-Gea *et al* [13].

3.12 Network Congestion Handling, Blocking and Waiting Models

Under higher network loads during peak system times, it is important to compensate for a worst case scenario. If too many requests are made, the network can become overloaded resulting in high network congestion and a significant packet loss rate. Quality across the network will degrade and this is not ideal when serving a paying customer base. This situation may be unavoidable at certain times. The adoption of a blocking or waiting model under these circumstances is the most effective way of relieving network congestion.

In Suh *et al* [9], adoption of a blocking or waiting model is discussed. A blocking model ignores user requests during times of high network congestion. This allows the system to recover from congestion once overloaded. The user can then make requests once the network is free of congestion.

Conversely, a waiting model employs a strategy of buffering user requests. Coupled with the initial buffering model proposed in Section 3.9, during high network congestion periods, this model does not reject user requests, but waits for a period where the network is capable of high quality video delivery. Although a delay policy reduces the QoS in the network, it does eventually satisfy customer need. Thus, the trade-off in allowing system delays ensures improved performance in the private network.

3.13 Churning

As highlighted in Section 3.3, the media streaming organization has to minimize churning amongst users. However, due to unforeseen events, power failures for example, the private network needs to have measures in place to deal with loss of an actively serving STB. This can be done by utilizing the Control Server. It can be responsible for reallocation of active serving peers if a peer unexpectedly falls off the network. By referencing its dynamic database of all active peers and requests, the Control Server can make informed decisions about reallocation of requested content. Ability to reallocate to other less constrained peers is achieved through content replication, described in Section 3.11. If no peers are available for serving content, either because of churning or low availability of content, the Control Server can always employ the Backup Storage Server to send required segments to a requesting STB. However, the Backup Storage Server is used as a last resort.

3.14 Summary of Complete System Elements

In Sections 3.1 to 3.13, all elements of the P2P VoD STB media streaming system were discussed. In this section, those elements are put together to highlight system functionality holistically.

This system is designed for a media streaming organization that serves a large or growing customer base. For the organization to maintain a well-organized, high quality, low cost network, control of services is performed by the organization itself.

Users are afforded minimal control in the network, allowing the network to perform optimally in most situations. Control of user behaviour is provided by the use of STBs for streaming media content to viewers.

Users can make requests from a library of media content supplied by the media organization. A main central server maintained by the organization at their premises feeds content to private networks of users via broadband Internet connections.

A private network of users exists in a closed community. Customers in the private network are interconnected via a router operating under the Gigabit Ethernet LAN standard.

The private network of STBs forms a P2P network, with every STB able to connect directly to every other STB. A Control Server facilitates connections between peers in the network while the

backup server receives content from the main central server over the broadband connection.

The Control Server gathers information dynamically on networked peers and maintains a table of all important network metrics. This information is then used to decide how much to segment the videos and perform replication amongst the STBs in the network. Segmentation is generally proportional to the number of STBs in the network. Instructions are fed to the Backup Storage Server which feeds segmented contents to network STBs during times of very low network activity. Replication is performed at this stage, with replicated contents distributed evenly amongst network STBs.

The initial segments of popular videos, with popularity determined by the media streaming organization, are stored in every STB. With less popular videos, an initial buffer delay policy is adopted with users having to wait a short period for the first few segments of the required video to be streamed to their STBs.

Videos that are streamed to the requesting STB are cached in that STB and deleted once a new request is made. Cached storage of requested content is redundant and uses up storage space in the network unnecessarily as fragments are assumed to be optimally stored in the STBs. Communication between the Control Server and STBs, and amongst the STBs themselves, is performed using RTCP connections. Media contents sent from the Backup Storage Server to STBs, and from STBs to each other, are transported using the RTP protocol.

During times of high network congestion, a waiting model is employed to ensure high quality content delivery across the network. Users that make requests when the network is saturated are required to wait a pre-determined period for congestion to lighten.

In the case of a saturated uplink STB, a serving peer protocol is engaged by the Control Server. In this situation, another STB with the same content as the saturated STB serves the requesting STB, thereby alleviating strain on the saturated STB. Content replication in the private network is thus performed proportional to deficit bandwidth in the network, and not based on content popularity.

The media streaming organization offers incentives to customers to keep STBs on to ensure optimal functionality in the network. In the case of churning, where STBs drop off the network unexpectedly due to a variety of problems, the Control Server employs another active peer to mimic the STB that has been shut down.

HD quality video delivery in the network is achieved through the use of the MPEG4/AVC video codec. High QoS delivery is the main constraint on which the P2P VoD STB Media streaming network is designed.

Figure 3-5 shows the complete P2P VoD STB system including network structure, connections between the various network nodes, and broadband connection to the main server.

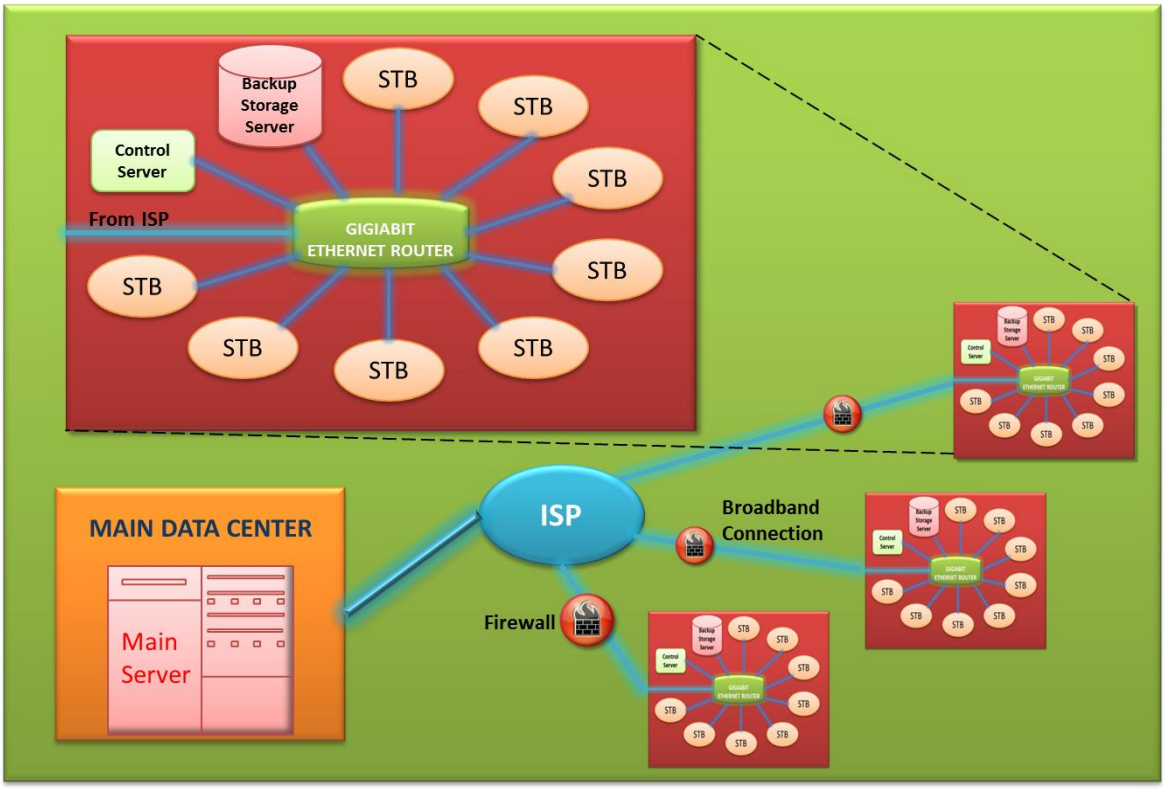

Figure 3-5 Complete P2P VoD STB Media Streaming Network Structure including link to Media Organization's Server

3.15 High Level Structure of Private Network Communications and Agents

Basic operation of the private network is illustrated with the use of diagrams in this section. Initially, a high level view of interconnecting nodes is given. Following this, an exploration of the agents facilitating functionality of the Control Server, Backup Storage Server and STBs is divulged. Relationships between the Control Server and Backup Storage Server, STBs and Control Server, STBs and Backup Storage Server, and between the STBs themselves is discussed within these sections.

3.15.1 Private Network Communication Overview

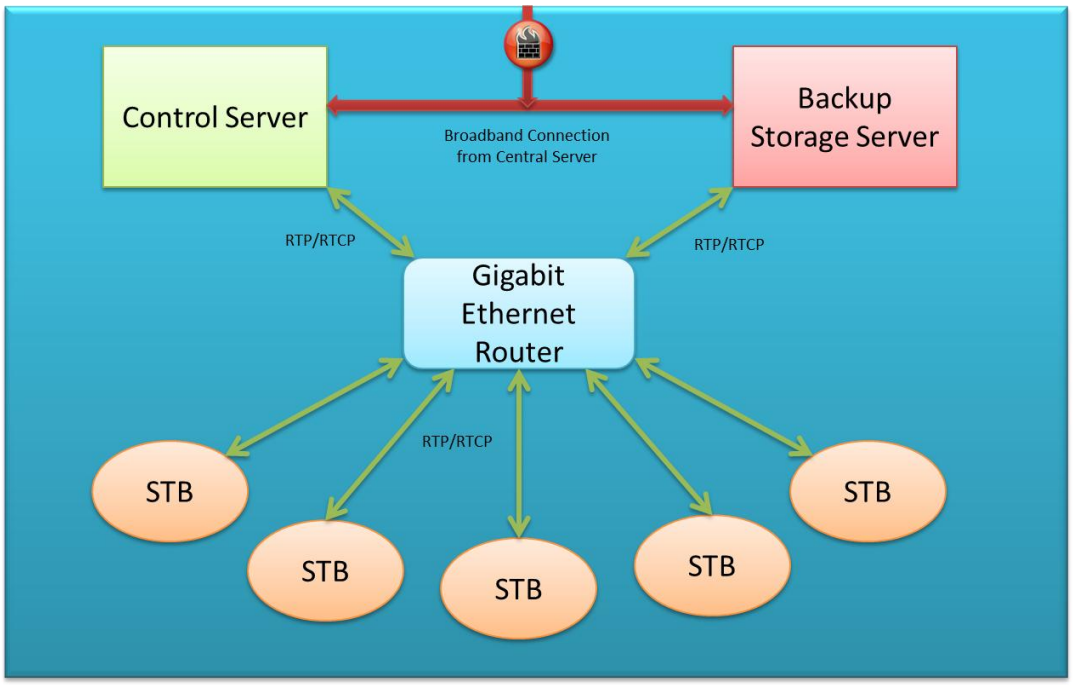

Figure 3-6 Private Network Communication Overview

Figure 3-6 depicts the basic layout of the private network, highlighting how the control and storage servers communicate with network STBs. It is seen that a firewall exists between the ISP (public) network and the private network of STBs. This ensures that unwanted network intrusions do not occur and only the organization has access to its private networks. The router's role in this depiction is noteworthy as all network communications flow through this single point. Maintenance of the router is an important consideration for the media streaming organization.

The RTP/RTCP transport protocol is implemented between the STBs and the Control and Backup Storage Servers. Instructions flow from the Control Server to the Backup Storage Server using the RTCP protocol. The Backup Storage Server publishes content to storage drives on network STBs via RTP transmissions. The STBs stream content to each other via RTP transmissions and maintain full duplex communication with the Control Server via RTCP.

In Figure 3-6, both Control Server and Backup Storage Server are linked to the main offsite central server of the media streaming organization via an Internet broadband connection. Content is sent to the private network by the organization via this downlink.

3.15.2 Control Server Structure

The Control Server is responsible for tracking behaviour of network peers, maintaining a dynamic database of all requests and segments being served, and most importantly, facilitating

communication between network STBs. Agents such as the Scheduler and Active Peer Monitor are adapted from Buyya *et al* [25].

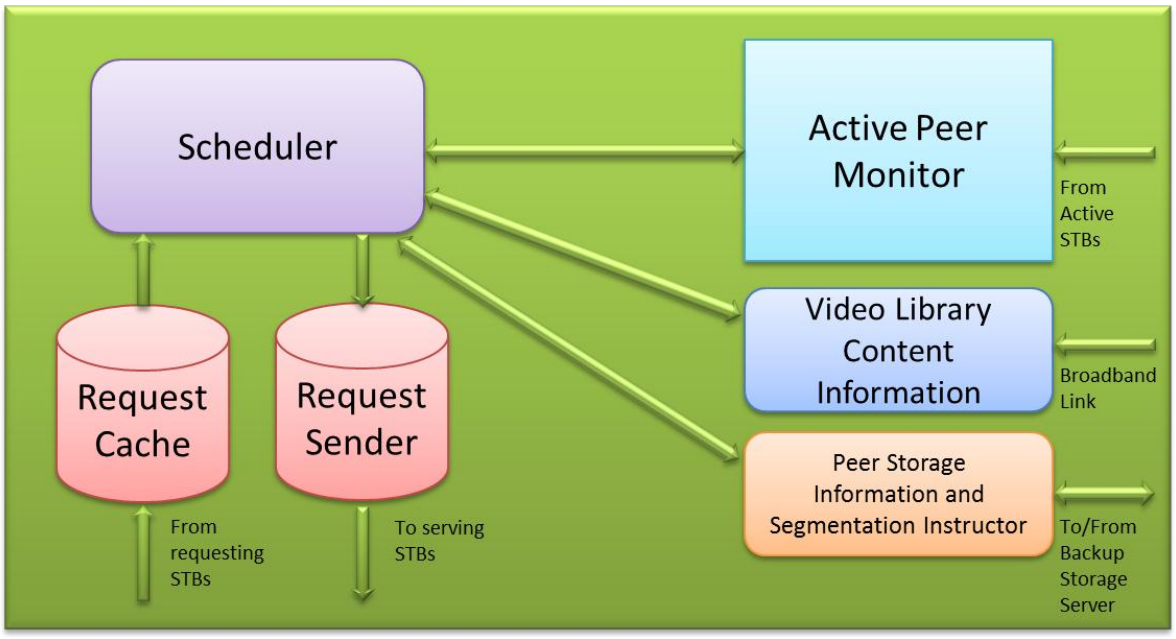

Figure 3-7 Control Server Structure Overview

A high level structure of the Control Server is shown in Figure 3-7. The server consists of five main agents.

All peers in the private network are monitored by the Control Server. This is achieved by the Active Peer Monitor which gathers information sent by active peers after a set amount of time. In this way, the Control Server is able to gather information on the current uplink and downlink speeds of peers. Also, if a peer does not send update information at any time, it is assumed that the peer is inactive. This information is passed to the Scheduler agent.

The Scheduler, which can be perceived as the Control Server brain, is responsible for communication facilitation in the system. All user requests are first sent to the Control Server request cache. Each request is then sent to the scheduler, which, based on information gathered on uplink and downlink speeds of each STB from the Active Peer Monitor, finds STBs that are likely candidates for serving the required content segments to the requesting STB. It chooses STB candidates with the required pieces, but also makes sure that the selected serving STB candidates have sufficient uplink bandwidth capacity to serve the respective pieces. It then forwards request messages to each serving STB to forward the required segments to the requesting STB, via the request sender agent.

Video library content information, sent from the central server of the media organization, is stored in the Control Server and sent to the Scheduler by the Video Library Content Information agent. Information such as video length, popularity and resolution are sent to this agent. This

information is used by the Scheduler to decide how to segment videos and how to distribute the segmented contents to network STBs. The Scheduler sends this information to the Peer Storage Information and Segmentation Instructor which then forwards it to the Backup Storage Server. For example, the Scheduler will inform the Backup Storage Server to place the initial segment of a video with high popularity rating on the storage drive of every STB in the system.

3.15.3 Backup Storage Server Structure

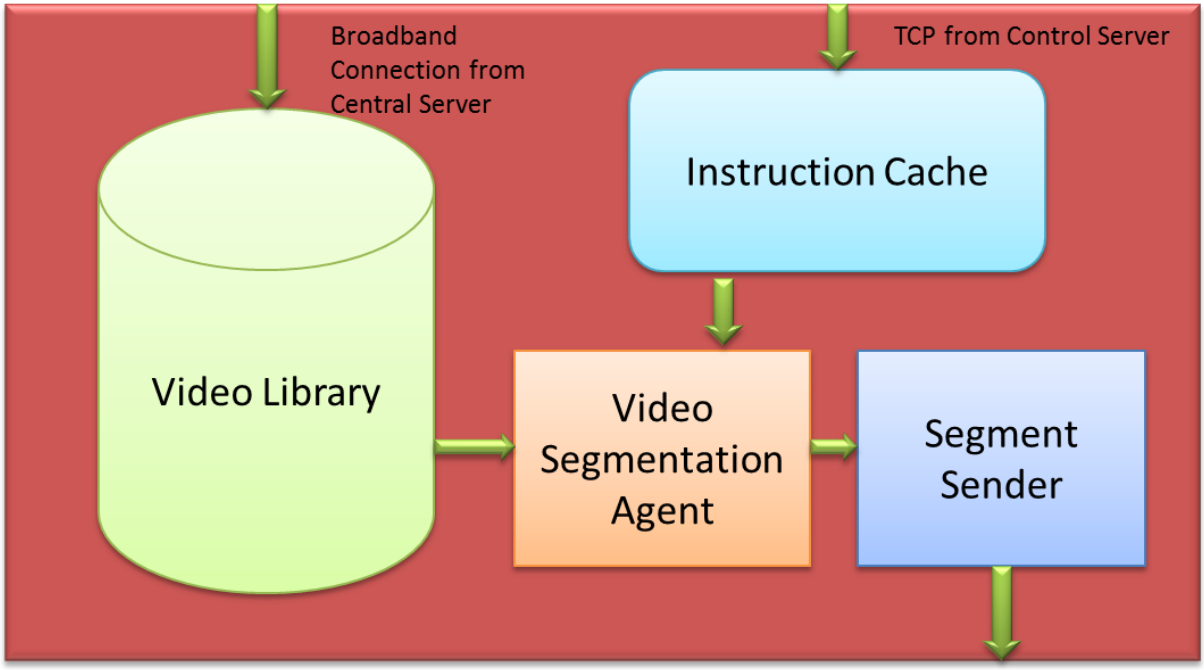

Figure 3-8 Backup Storage Server Structure Overview

A broadband connection from the Backup Storage Server to the media organization's central server allows the organization to publish content to the private network. A larger hard drive is used in the Backup Storage Server to store all videos made available by the organization. This storage is depicted in Figure 3-8 by the Video Library agent.

An Instruction Cache agent receives segmentation information for all videos in the library from the Control Server and forwards this information to the Video Segmentation Agent. This agent subsequently segments all videos in the library and publishes segmented content to network STBs accordingly, via the Segment Sender Agent.

The Backup Storage Server is utilized by the Control Server to serve video pieces when no STBs have the required content. This can only occur when there are blackouts in the network and a number of STBs are turned off.

3.15.4 STB Structure

Overhead in traditional P2P networks is a problem which is addressed by the design of STBs in the private network [24]. Figure 3-9 shows the high level structure of private network STBs, with Request Sender and Receiver, Segment Sender and Receiver, Segment Cache and Storage Manager agents adapted from Buyya *et al* [25].

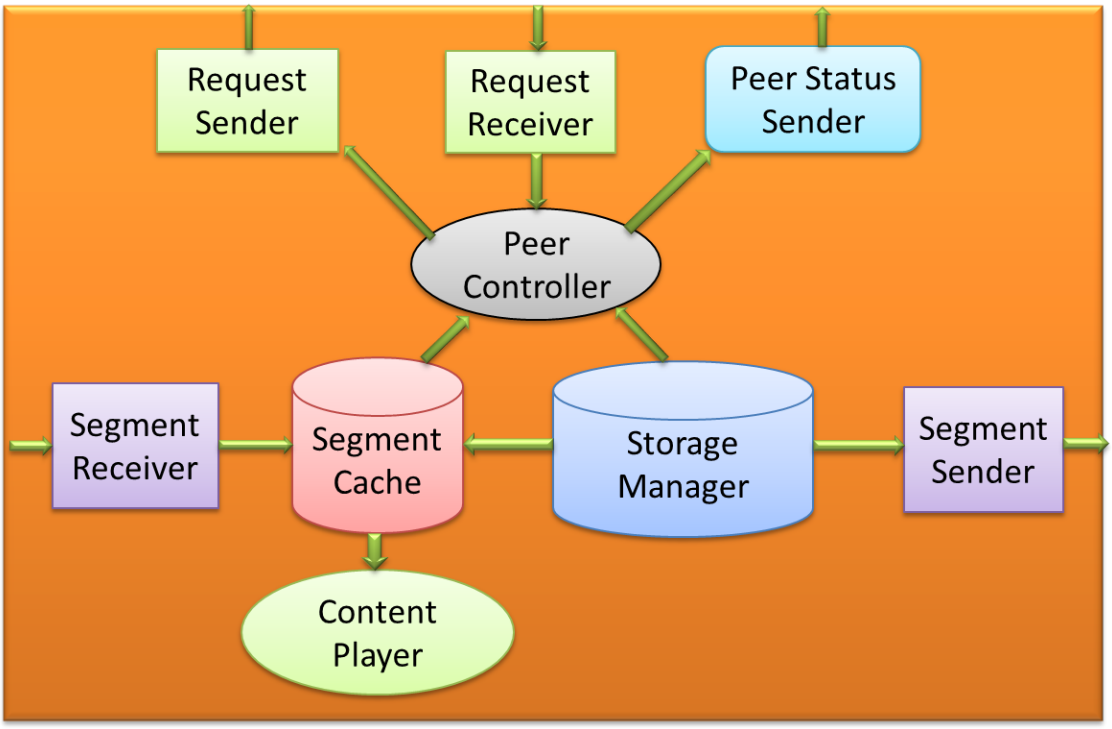

Figure 3-9 Private Network STB Structure

The Peer Controller Agent is an important fixture in the STB structure. The STB user interface is a gateway to the Peer Controller Agent. When a user makes a request, this information is sent to the Request Sender Agent by the Peer Controller. Via a RTCP connection, this request is then sent to Control Server's Request Cache.

STB uplink and downlink speeds are calculated by the Peer Controller, and this information is passed to the Peer Status Sender every set period of time. The Peer Status Sender subsequently sends this information packet to the Control Server periodically, alerting the Control Server's Active Peer Monitor agent of the STB's status.

Requested segments served by other STBs in the network are gathered by the Segment Receiver Agent of the requesting STB and pushed to the Segment Cache storage agent. The Segment Cache stores received segments in the STB until a new request is made. If segments are received out of order, the segment cache orders the received segments accordingly and passes these segments to

the Content Player. The user is allowed to watch and control video segments stored in the Segment Cache by utilising the Content Player agent.

Requests from the Control Server to serve other STBs are cached by the Request Receiver Agent. The Peer Controller obtains information stored by the Request Receiver and contacts the STB's Storage Manager. All video segments published on the STB by the Backup Storage Server are accessed by the Storage Manager. The Storage Manager then sends requested segments to the Segment Sender agent along with packet information about the requesting STB's network location. The Segment Sender then sends, via RTP, the relevant video segments in order to the Segment Receiver Agent of a requesting STB.

3.16 Summary

The P2P VoD STB media streaming system is presented as a hybrid network combining the strengths of centralized and decentralized media streaming architectures. The high level design highlighted in Section 3.15 gives light to the fact that this system scales well with a growing user base. Its strength is gained with numbers as more network STBs allow for better replication and the ability to serve more users. The next important step is the evaluation of this high level system. Chapter 4 discusses the work performed in evaluation of media streaming systems.

Chapter 4 NS2

This chapter discusses NS2 for performing evaluation of Client Server and P2P architectures. Reasons for utilisation of NS2 over other simulation environments are given based on requirements of the research performed for this dissertation.

Background information on NS2 is given, with a description of its characteristics and toolset, including the manner in which simulations are performed on it.

The system platform on which NS2 is implemented is discussed, along with a detailed description of the setup procedure. Following this, examples of simulations performed to familiarise with the software are detailed.

Physical setup of a media streaming client server architecture using VLC [47] media streaming software is performed to distinguish, qualitatively, basic simulation parameters and network behaviour. VLC is an effective streaming software for quickly setting up media streams to multiple clients via most streaming protocols. Comparison of physical results to those obtained from simulations is a method of verifying simulation validity. Furthermore, an understanding of discrepancies in results that may occur during simulation can be explained using this method.

4.1 Simulation Systems

Real world experiments can be extensive, complex and very expensive. This holds true for physical implementation of large scale media streaming networks. Thus, a simulator is needed for obtaining behavioural characteristics of networks proposed in this dissertation, thereby ascertaining real world performance.

From research of media streaming systems, it was found that a few network simulation software packages were used by the media streaming research community. NS2, OmNet ++ and PlanetLab were used as tools for simulation. NS2 was used or made mention of by Yadav *et al* [19],

Rubenstein *et al* [44], Buyya *et al* [25], Yang *et al* [14], Boronat *et al* [41], Fitzek *et al* [48], de Souza-Daw *et al* [49], Chaomei *et al* [43] and Chellouche *et al* [24]. OmNet ++ is used in Stais *et al* [5] and discussed in Fitzek *et al* [48] whereas PlanetLab is used to evaluate CollectCast [11].

PlanetLab is a group of computers available as a test bed for computer networking or distributed systems research [50]. This service is closer to emulation than simulation based and requires individuals to be affiliated with organizations or universities that host PlanetLab nodes.

OmNet ++ and NS2 are more suitable to the research involved in this dissertation. Table 4-1 highlights attributes of OmNet ++ and NS2.

Table 4-1 Comparison of OmNet ++ and NS2 [51]

	OmNet ++	NS2
Effective Simulation Runtime	Runtime Increases linearly with number of nodes	Runtime increases exponentially with increasing nodes
Memory Usage	Usage increases linearly with number of nodes	Usage increases linearly with number of nodes
Learning Curve	Fast	Steep
Availability of Models	Good variety for simulating computer systems	Vast set of communication protocols
Documentation	Well documented	Vast documentation but fragmented
Online Community	Small	Large online help available

From Table 4-1, it can be inferred that both systems have their merits, with NS2 more suitable to this research. For the purposes of this dissertation, NS2 is chosen primarily for the large online community support available. NS2 is well established in the research community [52]. The large body of work and protocols surrounding the NS2 platform make it useful in developing new simulation models for novel research purposes. NS2 also suited hardware available for research. Research has shown NS2 to produce results that generally correlate well with developed system models [52].

4.2 NS2

The work performed in this section is based on Teerawat *et al* [52]. NS2 was conceived in 1989 as an open-source event-driven simulator for the purpose of simulating computer communication networks. Having grown in industry, government and academia, many advancements and additions to the platform have been made [52]. NS2 contains modules for a wide range of network components that have become commonplace in current day networking. Researchers employ a scripting language to configure a network, and observe results generated by NS2. NS2 is an open source network simulator and one of the most widely used [52].

Simulation models which are not already incorporated into the NS2 platform are sometimes required. A good foundational understanding of the NS2 architecture is required to include these modules into NS2. The ability to include new modules into NS2 gives the researcher the capacity to simulate almost any network scenario [52]. Modules developed for the simulation scenarios in this dissertation are explained in Chapter 5.

Network Animator (NAM) is an animation display tool used with NS2 that allows researchers to observe a detailed visual representation of the network being simulated. All nodes and transmissions between nodes in the network are visually generated in NAM from the input trace file generated by NS2 during the network simulation.

4.2.1 NS2 Basic Architecture

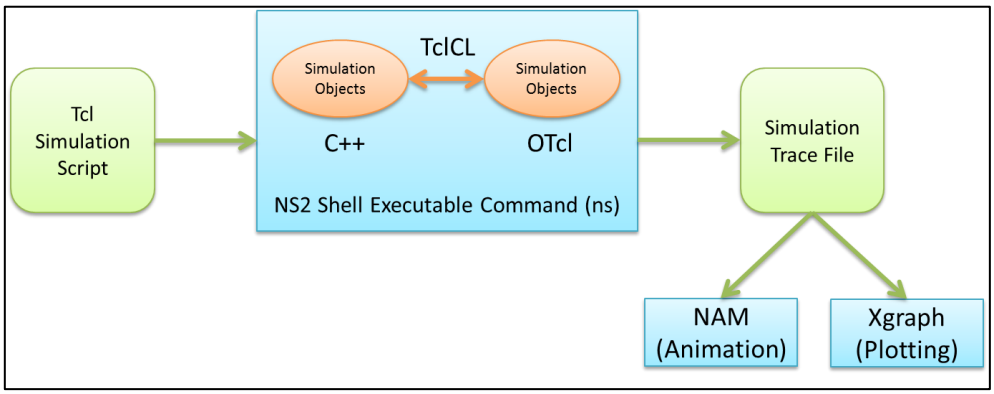

Figure 4-1 Basic Architecture of NS2 [52]

Figure 4-1, adapted from Teerawat *et al* [52] is a representation of NS2 architecture. Users input a Tcl simulation script as an input argument to the executable command 'ns' for simulation setup. Once run, NS2 outputs a simulation trace file, which can be plotted in Xgraph or visually depicted in NAM. Information from the trace file allows users to write out results needed for their research. This technique is used in this dissertation.

As seen from the large blue box of Figure 4-1, NS2 is written in two languages, C++ and Object Oriented Tool Command Language (OTcl). C++ defines the internal mechanism of the simulation objects whereas OTcl configures simulation through assembly and configuration of objects including scheduling discrete events. Thus, C++ makes up the backend whereas OTcl is responsible for the frontend. Linkage between the languages is achieved with the use of TclCL. [52]

The main reason for using two languages in NS2 is seen when experimenting with and changing network parameters and protocols. Detailed simulations of protocols require a systems programming language which can manipulate bytes, packet headers and implement algorithms that run over large data sets. Efficiency for these tasks is important and run time speed must be

minimized. Time between simulations, i.e. debugging, recompilation, etc., is less important. This is where C++ is effectively used in NS2. Alternatively, a large part of network research involves small network parameter variations or multiple runs of the same simulation model with different parameters to quickly explore different scenarios. Iteration time, where the model parameters are varied and re-run, is more important for these types of simulations. Configuration is run once so run-time is less important. Tcl is useful for these scenarios as it allows for quick turn-around times.

Variables in the OTcl domain are referred to as handles, where a handle is a string in the OTcl domain without any functionality. Its mapping to the C++ object class is where its functionality lies. The handle acts as a frontend which interacts with users and other OTcl objects. Procedures and variables in the OTcl domain are called instance procedures and variables. The power of NS2 lies with the large database of built-in C++ objects. These objects can be used by most regular users through manipulation of a Tcl simulation script. Although, users requiring unique functionality are required to develop modules pertaining to their research in C++, and use a OTcl configuration interface to put together these objects.

4.2.2 NS2 Installation and Directory Hierarchy

NS2 runs on Linux, Windows and Mac Systems. Cygwin, the Linux emulation platform for Windows, is used for research in this dissertation. NS2 source code is distributed as an all-in-one suite or component wise. The package is provided with an install script that configures the NS2 environment. This is followed by running the make utility to create the NS2 executable command seen in Figure 4-1. NS2 has four main components: NS, Tcl/Tk, OTcl and TclCL. The directory structure of NS2, when installed on the Cygwin system, is given in Figure 4-2, adapted from Têerawat *et al* [52].

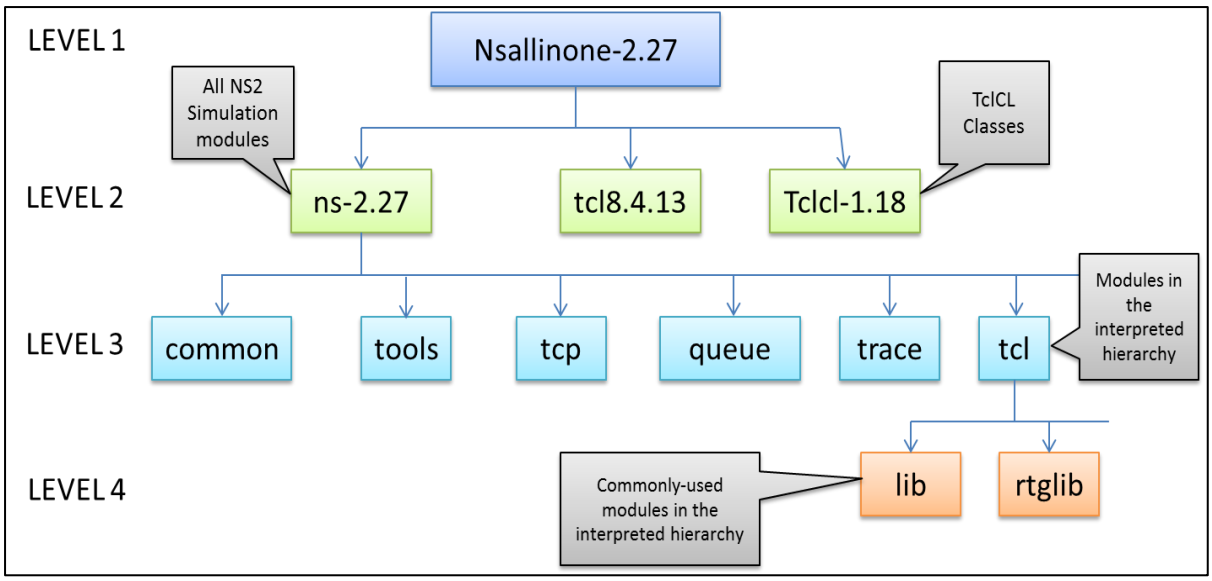

Figure 4-2 NS2 Directory Hierarchy [52]

Figure 4-2 illustrates C++ and OTcl class hierarchies. C++ is referred to as the compiled hierarchy and OTcl is the interpreted hierarchy.

4.2.3 NS2 Simulations

4.2.3.1 NS2 Program Call

When NS2 is installed, the ns executable command created in the NS2 home directory can be invoked using the following shell command line:

>> ns [<file>] [<args>]

Optional input argument 'file' sends a Tcl simulation script to NS2 for interpretation. Input argument 'args' allows the user to send input to the Tcl file.

4.2.3.2 NS2 Simulation stages

Designing a simulation involves determination of simulation purposes, configuration of the network, performance metrics and results to be expected.

The network configuration phase is important, as this is where components that constitute the network such as nodes, TCP or UDP connections are created. Simulation design dictates configuration of network components. This phase also entails timing simulation events. All configurations are performed through the scripting of the Tcl file.

Once the simulation has run, post processing of the results involves verification of program integrity (debugging) and performance of the simulated network. Performance evaluation is done by collecting and compiling simulation results. [52]

4.2.3.3 NS2 Simulation Example

To understand how simulations are designed and run, a basic example is given in this section, adapted from Marc Greis [53]. This example consists of four nodes connected by duplex links. The network is depicted in Figure 4-3, similar to NAM representation.

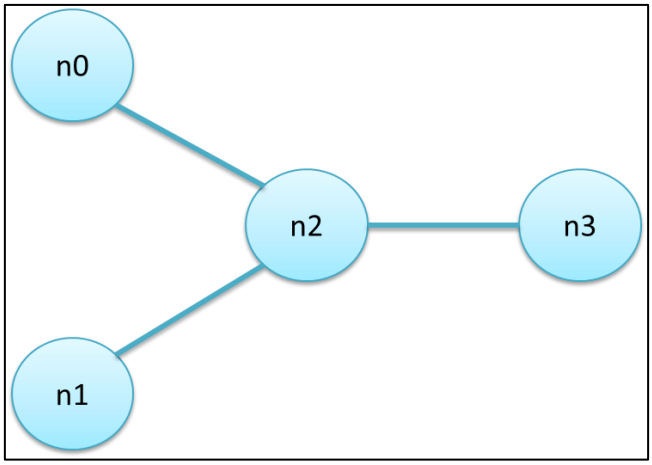

Figure 4-3 Illustration of Network Architecture from NS example as shown by NAM

Tcl is used to write the simulation script. The first step is creation of a simulator object using the following command:

```
set ns [new Simulator]
```

The simulator handle is returned and stored in the variable 'ns'.

A new file is open for writing of NAM trace data (the output of the simulation). The next command writes all trace data to the NAM handle.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Now that NS2 is able to collect all trace information for a NAM trace, a finish procedure is defined. The procedure finish {} is invoked immediately before the simulation terminates.

```
proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam out.nam &
        exit 0
}
```

The buffer of the packet tracing variables is flushed with command flush-trace. NAM is then executed to display data obtained from the 'nf' trace handle.

Topology creation is an important step in network configuration. All nodes and links are defined in this phase. A node is created using the command '[$ns node]'.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

Four nodes, no, n1, n2 and n3 are now created. Links between the nodes are defined using the following commands:

```
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms DropTail
```

Duplex communication between the nodes is specified, including link speeds (1 Mb), link delay (10 ms) and the queuing protocol (DropTail). Many queuing protocols are built into NS2 modules, including Stochastic Fair Queuing, Fair Queuing, etc. For a clear display of the results in the NAM trace animation, it is good practice to orient the nodes. The following lines illustrate node orientation.

```
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```

Now that a network has been defined, data must be transferred between nodes. UDP agents are created with CBR (constant bit rate) traffic sources. The UDP agents are attached to n0 and n1, and the CBR traffic sources are attached to the UDP agents.

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
```

CBR and UDP are agents built into NS2. The identifiers, ‘packetSize’ and ‘interval_’, are used to configure the traffic source. CBR is attached to UDP by the attach-agent command.

```
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0
```

Agent null is created and attached to n3 as a sink for the traffic source. In this example, the final task is to call events at required times. This is performed using the following commands:

```
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

The numbers in the above commands define the time in seconds at which commands must be run. 'cbr0'refers to the traffic source which is started at 0.5 s and stopped at 4.0 s. Finally, to run the simulation, the following command is executed:

```
$ns run
```

Data flow in the NAM representation looks similar to Figure 4-4.

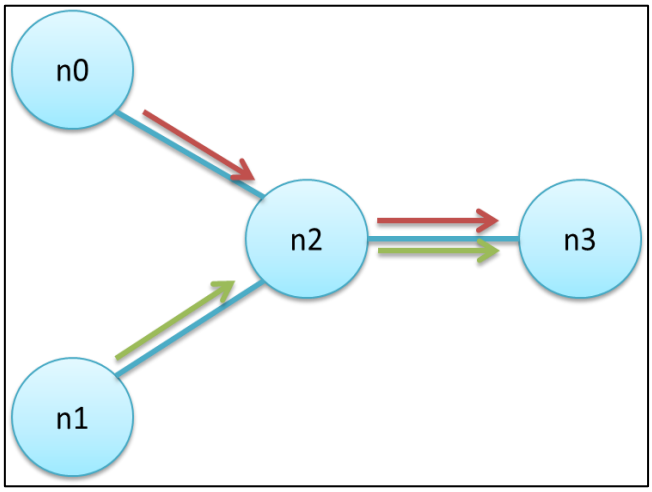

Figure 4-4 NAM representation of Data Flow

This example is intended to give a basic understanding of the way networks can be configured by means of a Tcl script, and simulated in NS2, with the results displayed in NAM.

4.2.4 Adding New Agents to NS2

As discussed earlier, usage of only the built-in features in NS2 may not be enough to carry out more flexible research. For this reason, NS2 allows users to add functionality they require to the simulation system by development of new modules for the purposes required. An example, adapted from Pradeepkumar [54], illustrating the method used to add agents to NS2 is given in this section.

In this agent the user writes two values to the Tcl file, radius and height of a cylinder. Following this, an agent written in C++, named 'Agentfile.cc', accepts these two values and computes the surface area of the cylinder. 'Agentfile.cc' is attached in Appendix A. An explanation of important code segments is discussed below.

Class TclClass defines the linkage between OTcl and C++. Class TclClass provides methods to access the interpreted hierarchy 'Agent/TSPAgentOtcl' from the compiled hierarchy 'TSPAgentClass ()'.

TclObject is the base class for all C++ simulation objects in the compiled hierarchy, referring in the case of this example to 'argc' and 'argv', which are passed from Tcl.

```
TSPAgent::TSPAgent() : Agent(PT_UDP) {
bind("tsp_var1_otcl", &tsp_var1);
bind("tsp_var2_otcl", &tsp_var2);}
```

The above code snippet is a constructor agent for the ping class that contains the bind function. It binds variables which have to be accessed from both C++ and Tcl. All changes made to the variable in Tcl, are made to the compiled hierarchy in C++ as well. 'tsp_var1_otcl' is initialised by the user in Tcl, and that value is used in C++.

```
int TSPAgent::command(int argc, const char*const* argv) {
if(argc == 2) {
if(strcmp(argv[1], "call-tsp-priv-func") == 0) {
TSPPrivFunc();
return(TCL_OK); }}
```

The function 'command()' is called when a Tcl command for the 'TSPAgent' class is scripted by the user. In this example, if the Tcl script contains 'call-tsp-priv-func', then member function 'TSPPrivFunc' of class 'TSPAgent', which calculates the surface area of the cylinder, is called.

```
void TSPAgent::TSPPrivFunc(void) {
Tcl& tcl = Tcl::instance();
tcl.eval("puts \"Message From TSPPrivFunc\"");
tcl.evalf("puts \"     Area of the Cylinder is = %f\"",
tsp_var1*tsp_var2*2*3.14);}
```

In TSPPrivFunc, 'tcl.eval' is a function that allows C++ to output to the user from the Tcl execution script. The function 'tcl.evalf' performs calculation of the surface area, and outputs the value in the interpreted hierarchy.

'Agentfile.cc' must be saved in the home directory of NS2 in a new folder, i.e. ~ns-allinone-2.27/ns2.27/newfolder.

To recompile NS2, the makefile must be edited to compile the new agent in NS2. In the 'OBJ_CC' list, 'newfolder/Agentfile.o' must be entered.

In the shell command prompt, in the NS home directory, NS must be configured and recompiled, using configure and make.

To test the new agent, a Tcl script file, 'Agent.tcl' is given:

```
# Create MyAgent (This will give two warning messages that
set myagent [new Agent/TSPAgentOtcl]

# Set configurable parameters of MyAgent
$myagent set tsp_var1_otcl 3
$myagent set tsp_var2_otcl 5.5

# Give a command to MyAgent
$myagent call-tsp-priv-func
```

It is observed that configurable parameters are set in the Tcl script, but calculation of surface area is performed in C++. 'myagent' refers to the TSPAgent in the interpreted hierarchy linked to the compiled hierarchy agent 'TSPAgent'. The command to access the private function and display the surface area is given by the last line of the code above, '$myagent call-tsp-priv-func'.

Execution of the Tcl script file is performed by executing the shell command:

```
~ns Agent.tcl
```

The resultant output in the command prompt is as follows:

```
Message From TSPPrivFunc
Area of the Cylinder is = 103.62
```

Thus, an illustration of how function can be added to NS2 is given, allowing execution of more novel research ideas.

4.3 Simulation Platform Setup

The desktop computers on which simulations were run had the following specifications:

CPU : Intel Core i7 3770 @ 3.4 GHz
RAM : 4 GB
Motherboard : Intel DH77EB Lake
Hard Drive : 2 TB Seagate Barracuda 7200 rpm HDD

The first operating system installed on the simulation PC was Linux Ubuntu version 12.04. This system was used with NS version 2.34. The first time installation of NS 2.34 proved difficult. However, various online help was available and the simulation system was installed successfully. Many tutorials were performed to familiarise with the learning curve posed by NS2. Configuration of networks, analysis of trace files, packet monitoring and addition of modules to NS2 were all practised on the simulation system. The next step was to simulate basic media streaming.

4.4 Trace Driven Media Streaming Network Simulations

The work performed in this section is based on Fitzek *et al* [48]. Research of media streaming systems has shown that transferral of video trace files between nodes in a simulated network environment is commonplace [48]. The reason for widespread use of this method is the associated difficulty of developing a uniform video source model, as many video encoding standards exist. Difficulties in modelling the behaviour of video sources with different encoding models and

different contents have elevated the utilization of network simulators for performance evaluation. Source models require multiple parameters to be taken into account, but can be very accurate for the specific situation they are developed for, as all the parameters that affect the behaviour of the model are incorporated in simulation [48].

A video trace file is a text based file that records the frame sizes of an encoded video sequence. Used in network simulators, this method has the capability of facilitating network layout, evaluating QoS and other relevant parameters. Trace files give the researcher the ability to incorporate most video source models into their simulations. Table 4-2, adapted from Fitzek *et al* [48], exhibits standard trace formats for differently encoded video sources.

Table 4-2 Trace File Formats for Various Video Codecs [48]

Group	Entries in Trace File						
MPEG-4 (new)	Frame #	Time	Type	Size [byte]	PSNR-Y	PSNR-U	PSNR-V
MPEG-4 FGS	Frame #	Type	Size [bit]	PSNR-Y	PSNR-U	PSNR-V	
H.26L	Frame #	Type	Time	Size [byte]			
H.263 H.261	Time	Type	Size [byte]				

Trace files use different column positions to present the values needed for different simulator interfaces. Simulator interfaces have to be adapted to read from these trace file formats. The NS2 simulator interface for reading trace file formats was written by Michael Savoric from the Technical University of Berlin, Germany [48].

Replication of the video trace simulation presented in Fitzek *et al* [48] was performed. In this example, a video file is transferred from a sending node to a receiving node via two routers, a client-server model. The NAM representation of this model is depicted in Figure 4-5.

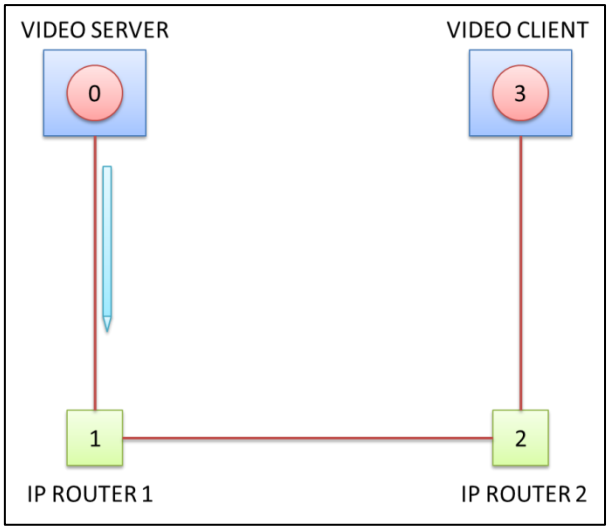

Figure 4-5 NAM Representation of Client-Server Architecture for Trace file test

The Tcl script for this simulation, 'VideoTrace.tcl', can be found in Appendix A. A standard trace file 'Verbose_Jurrasic_64' was used for this simulation. The script and NS2 modules were modified. This simulation was setup and ran on NS 2.27 on Linux Ubuntu Version 10. The transfer of a video trace representing a video source was performed. Dropping of packets and lossless transfer results were explored by adjustment of network parameters such as link bandwidth and queuing. This model was simple and employed the UDP protocol, modified to include RTP headers to transport video packets between nodes. In order to accomplish the objective of quality and scalability quantification of media streaming systems, more advanced models needed exploration and development.

4.5 Video Streaming Implementations in NS2

The work in this section is adapted from Boronat *et al* [41] and Carrascal [55].

4.5.1 RTP Based Tool-Set for Video Streaming Simulation using NS2

A simulation framework for video streaming evaluation is presented in Boronat *et al* [41], with the design of a full RTP/RTCP implementation for NS2. Multimedia tools are combined with the RTP protocol to develop a tool set for measurement of network level QoS and common video quality metrics. This research was proposed as a platform for researchers to explore source and receiver based techniques for improving the QoS in multimedia streaming services. [41]

Use of RTP (Real Time Protocol) and RTCP (Real Time Control Protocol) in Internet based media streaming is due to the timestamp and sequence number mechanisms provided by RTP. Reconstruction of the source media stream timing and detection of packet losses are provisioned for in

this protocol. Work performed by Boronat *et al* [41] aims to create a more complete implementation of RTP/RTCP in NS2 as the native NS2 implementation is limited.

Focus was placed on trace file generation from an MPEG source in Boronat *et al* [41]. Thereafter, transfer of the generated video trace file using RTP/RTCP in NS2 was performed using the framework presented in Boronat *et al* [41]. A detailed description of the setup of this framework, adapted from Carrascal [55], is discussed in this section.

The toolset provided firstly requires generation of an MPEG-4 encoded video using the 'ffmpeg' [56] tool. VLC media player is configured in this system as an RTP streamer and is opened simultaneously with the 'rtpdump' [57] tool. VLC streams the MPEG-4 encoded video on the same transport IP address that 'rtpdump' is listening. Listening tool 'rtpdump' records the RTP packets' information and generates an output packetized video file.

Using a new tool, 'RTPTracefile.exe' [41], developed by Boronat *et al* [41], the packetized video file is converted to a Traffic trace file compatible with NS2. The trace file contains packet sequence numbers, payload size and sending time for each RTP packet.

In the simulation phase, the enhanced RTP/RTCP is implemented in NS2 for transportation of the Traffic trace file. Enhancements to the native RTP implementation in NS2 include: definition of all types of RTCP packets, network-level metrics monitoring and registration in simulation time, multiple NS2 traffic pattern processing and support for multiple multicast streams on the same node.

The first part of the simulation phase is declaration of a new RTP Session instance for each simulated network node that will be streaming. RTP and RTCP agents are attached to the participating nodes. Sending and receiving functions are handled by the RTP agents, while control messages are passed by the RTCP agents. The generated traffic file is attached to the RTP agent attached to the video server node.

At the receiving node, incoming RTP packets are passed to the RTP session instance for processing and registering the sending node's statistics, such as receiving time, network delay, total packet loss and jitter amongst others. Each video receiver can record these statistics in an output RTP Receiver Trace File. An RTP Lost Trace File is created to record the sequence number of each expected RTP packet that is not received. RTCP feedback reports are exchanged during simulation between the sender and receiver.

Once the RTP reports are generated, post simulation processing can proceed. Statistics recorded in the Output Trace Files can be used to analyse network performance. The RTP Lost Trace File is used to remove data lost in simulation from the original packetized file generated by 'rtpdump'. Removal of lost packets is performed through the use of a tool developed by the researchers named

'RemoveLost.exe' [41]. This procedure results in a received file that is similar to the file obtained by the receiving node. A receiver tool 'rtpplay' [57] is then employed to transmit the packetized file to VLC which is then configured as a stream listener. The received file can be stored as an MPEG file or be played in VLC.

A full implementation of the system described above was performed. Implementation of this method was aided by Carrascal [55]. The work performed is described Section 4.5.2.

4.5.2 Implementation of RTP Based Tool-Set

NS2.27 was used as the simulation platform in Ubuntu 10 for implementation of the method described in Section 4.5.1. The RTP/RTCP module proposed in Boronat *et al* [41] was successfully added to NS2. Addition of this module is similar in nature to the method described in Section 4.2.4. The ability to send trace files using the RTP/RTCP protocol facilitated the generation of an output trace file giving statistics on network performance. A discussion of the key sections of the Tcl script, adapted from Carrascal *et al* [55], implementing the RTP/RTCP protocol in NS2 is given below. The complete Tcl script, 'RTPTrace.tcl' can be found in Appendix A.

A basic client-server topology is setup with a receiving node and a sending node each linked to the same router.

```
set node_(send1) [$ns node]
set node_(receive1) [$ns node]
set node_(router1) [$ns node]
$ns duplex-link $node_(send1) $node_(router1) 10Mb 5ms DropTail
$ns duplex-link $node_(receive1) $node_(router1) 10Mb 5ms DropTail
```

The video trace file used, 'starwars.nsformat', is generated by a trace file generator suited to the format required for the RTP protocol in this simulation.

```
set trace_file [new Tracefile]
$trace_file filename starwars.nsformat
```

The RTP agent is created as well as the RTCP agent using the newly added 'RTP_v2' and 'RTCP_v2' modules. A new RTP session instance is also started.

```
set RTP_s [new Agent/RTP_v2]
set RTCP_r [new Agent/RTCP_v2]
set self [new Session/RTP]
```

The RTP agent is attached to the sending node whereas the RTCP agent is attached to the receiving node. These agents are then connected.

```
$ns attach-agent $node_(s1) $RTP_s
$ns attach-agent $node_(s2) $RTCP_r
$ns connect $RTP_s $RTCP_r
```

The video trace file is simulated as a basic traffic trace file in NS2. This trace file is then attached to the RTP sending agent.

```
set video [new Application/Traffic/Trace]
$video attach-tracefile $trace_file
$video attach-agent $RTP_s
```

The RTP and RTCP agents are attached to the RTP session instance. A delay of 100 ms between RTCP feedback sessions is set and the packet size used by the RTP agent is set to 1064 bytes.

```
$RTCP_r session $self
$RTP_s session $self
$RTCP_r set interval_ 100ms
$RTCP_r set seqno_ 0
$RTP_s set packetSize_ 1064
```

Finally, generation of the video by the RTP agent is started along with the RTCP agent to facilitate video receipt. After stopping the agents, the simulation is complete and the resultant trace file obtained is analysed.

```
$ns at 0.0 "$video start"
$ns at 0.0 "$RTCP_r start"
$ns at $end_sim_time {
$video stop
$RTCP_r stop
$ns run
```

Information contained in the periodic RTCP-RR packets sent between the destination and source is stored in a log using the following command.

```
 >>ns RTPTrace.tcl > evaluationfile-info.log
```

Network level performance metrics such as Round Trip Time and Average Throughput versus Packet Error Rate can be obtained from analysis of this log.

Implementation of the MPEG2 sender/receiver application and RTP tools designed by the researchers in Boronat et al [41] required modification for execution in NS2. During execution of the RTP tools and the MPEG2 sender/receiver application, network level evaluation techniques were explored and manipulation of NS2 and its agents was performed.

The technique used for video analysis in Boronat *et al* [41] was used to further development of network simulation of media streams. The method of emulating real video streaming in a simulated network was performed by using video trace files generated from source video sequences. Reconstruction of the video, as seen by the receiver when streamed through the simulated network, was to be used for experimentation later in the dissertation. Reconstructing a playable version of the streamed video gives the researcher ability to measure video quality metrics and quantifiably determine network performance through these metrics.

4.6 Physical Network Setup

Use of VLC Media Player's media streaming capabilities in Boronat *et al* [41] led to an understanding of media streaming network behaviour. VLC Media Player is a free, open source, cross-platform multimedia player that incorporates many streaming protocols [47].

To obtain a qualitative physical understanding of data obtained from simulations and to verify results seen in simulations, a basic physical network was setup. This network was setup to qualitatively assess the behaviour of a basic streaming media system. A simple client-server system was prepared using an Intel i7 Quad Core desktop computer as the server. Two Intel i7 Quad Core computers and one Intel Dual Core computer were arranged as client computers. These four computers were connected in a LAN using a 100 Mbps Ethernet Router.

VLC Media Player was installed on all the computers, all of which had Microsoft Windows 7 Operating System installed. VLC on the server computer was configured as the sender and streamed video using the UDP/TCP protocol with the RTP application layer media protocol. VLC on the client computers was configured to listen to the IP port of the sender.

A standard HD video from Trace Website [58] was used for streaming to clients. The server was first configured to stream video to one client. Network bandwidth utilization was monitored on the server and was found to be 22%. Video received on the client computer had no discernible difference to the source video on the server.

Thereafter, two clients were streamed to by the server and delay, less than one second, in video receipt was experienced at the second client. Quality of received video was qualitatively observed to be poorer than the source but more degraded on the second (newer) client than the first. Network bandwidth utilization had jumped from an average of 22 % for a single client to an average of 46 % for two clients. Thus, delay and reduced quality of received video was not due to saturated uplink bandwidth at the server. Further monitoring had indicated that the server CPU was at 100 % load when streaming to two clients. This is a result of on-the-fly encoding of the streamed video, a CPU intensive task. Sending the video to two clients required encoding the source video twice.

Including three clients, network utilization increased to about 80 %, but the quality of received video degraded sharply when compared to the source video. The CPU load stayed at 100 % and delay across the network increased to almost five seconds. The addition of a third client degraded the quality of video seen by the first two clients and network jitter was observed. The bit rate of the video was variable.

The aim of utilizing a physical network was to examine the quality degradation of streamed video and delay increases, by virtue of lack of uplink bandwidth at the server. However, the server CPU became a network bottleneck before the server uplink capacity of 100 Mbps could be reached. Performance of the client-server architecture in this physical implementation indicated the hardware intensive nature of servers. An Intel i7 3770 Quad Core processor was incapable of encoding and streaming video to more than 3 client computers simultaneously. Research on client-server systems discusses the high resource requirements of media streaming servers and experimentation performed in this section with a physical system gave insight into this issue.

To solve the above problem, the network bandwidth had to be throttled i.e. to reduce the bandwidth link between the nodes. A 24 port network switch with management was used for the purpose of throttling the LAN. The same client-server configuration as above was implemented using the new network switch but with server uplink bandwidth throttled from 100 Mbps to 10 Mbps. A new lower bandwidth video, with an average bit rate of 2.5 Mbps, was also selected to lessen the server load during encoding and streaming. The server load was increased from one client to three clients, while network quality was monitored.

With one client, server uplink bandwidth utilization was an average of 30 % with the CPU load at 15 %. There was no discernible change in video quality between the source and the sender.

When streaming to two clients, server uplink bandwidth was an average of 70 % while CPU usage hovered around 40 %. Delay was experienced at the second client but received video quality remained indiscernible from the source video at both clients.

With three clients, the uplink bandwidth was saturated at 100 %, whereas CPU usage averaged 80 %. Importantly, video quality degradation across all three clients, compared to the source, was observed. Delay on the third client was longer than that experienced on the second client. The introduction of a third client also caused quality degradation for the first two clients when compared to streaming to just two clients. Blocking and jitter were seen at the first two clients and wasn't experienced prior to the addition of the third client.

Non-linear behaviour was observed with server uplink bandwidth in the physical network. As more clients were streamed to, bandwidth usage increased non-linearly for the same streaming source. It was observed that newer clients cause the quality of streamed video to decrease across all clients and cause greater non-linear delays. This non-linear behaviour observed in bandwidth usage was attributed to the router buffering protocol which operates on a FIFO basis. Non-linear increases in delay were also attributed to this method of buffering. When the buffer is cleared, increases in performance and reductions in delay are observed and the opposite is seen when the buffer is full. The first client is normally first in the queue which is why the best performance is seen for the client making the first request. Changes in delay are also caused by the variable CPU workload. These

observations are highlighted in Chapter 5 during simulation, as the same behaviour is experienced when simulating client-server systems. A physical setup of the client-server network, as described in this section, gives qualitative verification of simulation results, thereby giving credibility to further simulations for which physical network emulation would be impossible.

Chapter 5 Quality and Scalability Quantification

In this Chapter, ways in which video transmission systems are evaluated are emphasised. Merits of trace driven simulations for video streaming are discussed, including a presentation of earlier trace simulations and results obtained.

The value of analysing real-time streamed videos in simulation is supported by a few works. Attention is brought to the utilisation of this method in this dissertation, including a detailed study of the method proposed by Yang *et al* [14] and Chih-Heng *et al* [15]. Integration of NS2 and Evalvid, as addressed by Chih-Heng *et al* [15], allows for evaluation methods of video transmission using PSNR, allowing one to objectively determine video quality observed by the end user through simulation.

Discoveries made during practice with this new method are divulged. Simulations of centralized and decentralized systems are presented, including the behaviour of different video sequences in simulation.

To achieve the task of quantification of quality and scalability of media streaming network architectures, development of a quantification system and its methodology is discussed. The use of the toolset provided by Evalvid and NS2, along with modifications made to the toolset, allows for development of a QSQS for media streaming network architectures. Complete simulation structure and code development for implementation of the quantification system is given in this section.

Following development of the quantification methodology, Client-Server and P2P architectures are developed for evaluation by this system. Thereafter, a serving peer protocol is included in the P2P system to further understand decentralized architectural behaviour and demonstrate the merits of the QSQS. Results of findings from simulations are presented in Chapter 6.

5.1 Analysis of Evaluation Methods for Media Streaming Networks including PSNR as a Video Quality Metric

In Section 2.1.6, evaluation methods of media streaming systems used by various researchers were briefly discussed. Network QoS evaluation of media streaming systems is a popular method as referenced in Section 2.1.6. Determining QoS requires various performance metrics, with the most significant to media streaming being client request rejection probability. Rejection probability is often measured against system workload, number of peers in the network, peer cache size or server load. This method offers a good reflection of network performance, but there are improvements that can be made in terms of end-user perception of network performance. Measuring server stress has also been a way of gauging the scalability of a network, in addition to request rejection probability.

Employment of frame-level performance and packet loss rates for performance evaluation of CollectCast [11] and DeMSI [25] proved effective, as a better understanding of end-user perception of video quality was achieved.

Rejection probability and packet loss rates do not, however, give a comprehensive measurement of video quality. In a video sequence, different packets carry frames with different levels of importance to the quality of the video. Thus, loss of a high luminescence frame or packet of frames carrying more information has a more severe effect on end-user perception of the received video, when compared to a packet loss of lower luminescence frames. Loss of packets with frames of varying luminescence still produces the same packet loss rate.

Combination of network performance metrics with video quality metrics provides a more complete view of network performance and end-user perception. The primary focus of this dissertation is the quantification of quality and scalability in a media streaming network, and comparisons between different networks thereof. Research into the media streaming evaluation method discussed in Boronat *et al* [41], addressed in Section 4.5.1, gave an improved understanding of video quality analysis. The use of PSNR to quantifiably determine video quality received by the end-user in a media streaming network is important to this study. Further research led to the discovery of 'An Evaluation Framework for More Realistic Simulations of MPEG Video Transmission' Chih-Heng *et al* [15] and 'Design of Video Transmission and Quality Evaluation System Based on NS-2' Yang *et al* [14], works that combine NS2 with Evalvid, a framework for video analysis. Adoption of these research papers in this dissertation is discussed in detail in this section.

PSNR, a video quality based metric, is used as the primary performance indicator by Chih-Heng *et al* [15]. PSNR refers to the ratio of the peak power of a signal and the power of distorting noise that affects the integrity of its reconstruction [59]. Use of PSNR is most commonly found in the field of image compression codec evaluation [59]. PSNR is used to measure the quality of

reconstruction of lossy compression codecs, with the original data representing the signal and compression error representing noise. Importantly, PSNR can be used as a video quality metric, as long as the video type and codec are not changed between sender and receiver [60]. Higher PSNR indicates better reconstruction quality, although the codec cannot be changed in transmission. PSNR as a video quality metric is thus applicable to this dissertation.

Application of the PSNR assessment metric to video is performed on a frame-by-frame basis. A video is made up of frames, with each frame representing an image on its own. Pixels in the frame carry different colour components with varying levels of luminescence. Comparison of a reconstructed received frame to the original sent frame is the way in which PSNR operates on the image. The difference in colour components of each pixel between the sent and received frames is calculated using PSNR to determine the error in transmission. The Mean Square Error (MSE) between sent and received frames gives a definition of PSNR.

$$PSNR = 20.log_{10}\frac{MAX_I}{MSE} \qquad (5\text{-}1)$$

MAX_I is the maximum possible pixel value of the image [60]. Further details on use of PSNR for video quality analysis are given in the Section 5.2.1.

In Boronat *et al* [41], PSNR was demonstrated as a viable video quality metric in the RTP/RTCP video evaluation framework and simulation model. However, work performed by Chih-Heng *et al* [15], was based on a more general platform and was better suited for development of a quantifiable quality and scalability evaluation system as proposed in this dissertation.

5.2 A Video Quality Evaluation Framework for Video Transmission

A complete tool-set for MPEG video delivery evaluation, based on NS2, is presented in Chih-Heng *et al* [15]. Employment of the EvalVid framework, for its video evaluation capabilities, in conjunction with NS2, for its effective network simulation capabilities, allows for creation of a streamed video quality analysis framework. The connecting interfaces of EvalVid are extended to include NS2 as a replacement for the simple error simulation model that is part of EvalVid [15]. This gives researchers an ability to analyse performance of real video streams through simulation. The framework uses two important QoS metrics for video quality evaluation: PSNR and fraction of decodable frames. QoS of video delivery is an effective method of determining end-user perception of received video in comparison to network QoS techniques such as rejection probability. An analysis of this nature can be referred to as QoE.

The argument in Chih-Heng *et al* [15] on the ability of network performance metrics, such as packet loss rates and packet/frame jitter, to adequately rate end-user perception of video quality supports the work performed in this dissertation.

5.2.1 EvalVid

EvalVid provides a complete evaluation toolset for analysis of real or simulated streaming video networks. This software allows researchers to evaluate networks by methods of end-user video quality perception. EvalVid comes with a simulated environment which is a simple error model that represents corrupted or missing packets in a real network [61]. This simple error model lacks the complexity needed by researchers in evaluation of novel network designs or more realistic network scenarios [15].

The integration of NS2 within the EvalVid framework is the focus in Chih-Heng *et al* [15]. Combination of these systems for streaming network evaluation allows researchers easier development of new networks and the ability to analyse network designs in the presence of real video traffic [15]. Enhancement of video delivery services can also be achieved with NS2 and EvalVid, as researchers are able to introduce more complex network scenarios owing to the vast functionality of NS2.

The structure of EvalVid is given in Klaue et al [61]. Figure 5-1 represents the EvalVid framework and is discussed in detail.

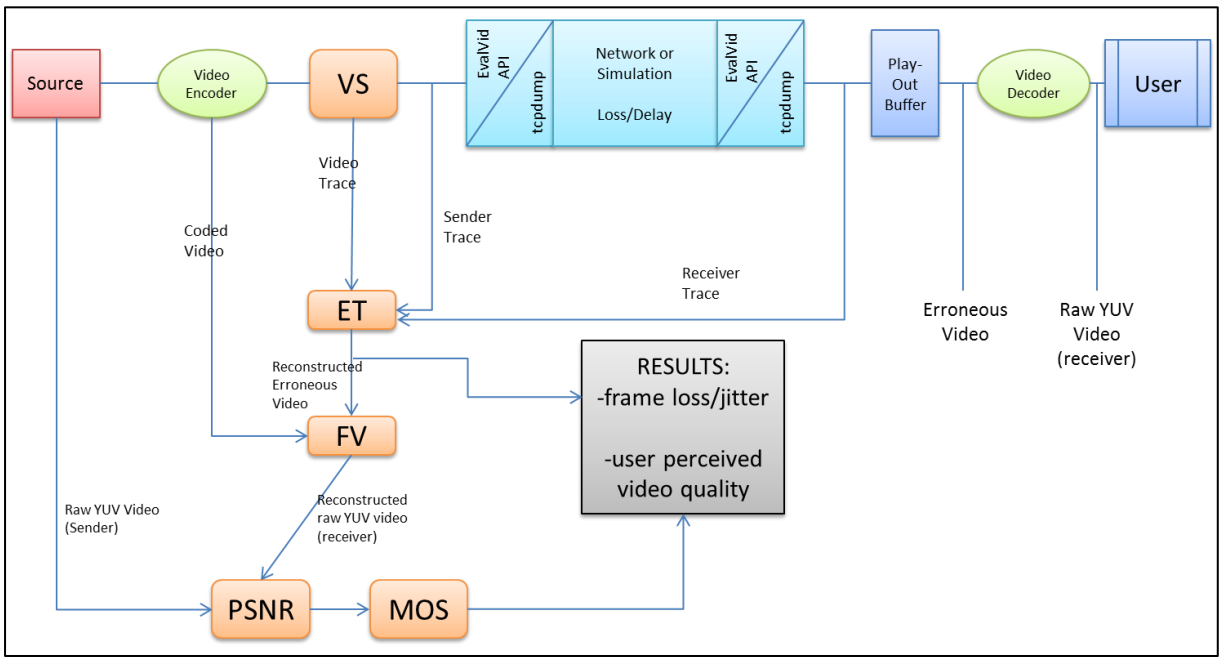

Figure 5-1 Evalvid Framework Structure [61]

The **source** can be any video in the YUV QCIF (176 x 144) or YUV CIF (352 x 288) formats [15].

MPEG-4 [62] codecs, are a method of defining compression of audio and visual digital data, and are supported by the framework developed by Chih-Heng *et al* [15]. 'ffmpeg' [56] and Xvid [63], an open source research project focusing on video compression are also supported by the framework. These codecs form the **Video Encoder and Decoder** components.

The **VS (Video Sender)** [15] component reads the encoded video file from the video encoder and performs fragmentation. Large video frames are fragmented into smaller segments and transmitted via UDP packets over the network. Each UDP packet sent is recorded in a sender trace file with the following details: timestamp, packet ID and payload size. Tcpdump provides this capability in a real network, whereas the sender records this information in a simulated network. A video trace file containing information on every frame in the real video is also generated by the VS component. Both the video and sender trace files are used in evaluation later.

Table 5-1, adapted from Chih-Heng *et al* [15], gives an example of the video trace file with Table 5-2, adapted from Chih-Heng *et al* [15], exhibiting the format of a sender trace file.

Table 5-1 Video Trace File format with sample data [15]

Frame Number	Frame Type	Frame Size	# of UDP Packets	Sender Time
0	H	29	1 segment at	33 ms
1	I	3036	4 segments at	67 ms
2	P	659	1 segment at	99 ms
3	B	357	1 segment at	132 ms
4	B	374	1 segment at	165 ms
…	…	…	…	…

Table 5-2 Sender Trace File format with sample data [15]

Time stamp (sec)	Packet ID	Packet Type	Payload Size (bytes)
0.033333	0	Udp	29
0.066666	1	Udp	1000
0.066666	2	Udp	1000
0.066666	3	Udp	1000
0.066666	4	Udp	36
0.099999	5	Udp	659
0.133332	6	Udp	357
…	…	…	…

The next important component is **ET (Evaluate Trace)** [15]. ET is used in the evaluation phase after video file transmission through a real or simulated network is complete. The receiver trace file, with the same format as the sender trace file, is sent back to the sending entity. ET takes into account the original encoded video file, video trace file, sender trace file and receiver trace file, and creates a frame/packet loss and frame/packet jitter report, thereafter generating a reconstructed video file. This recreated video file is assumed to be erroneous as it corresponds to the video found at the receiver side, seen by the end-user. ET reconstructs the erroneous video by copying each frame of the original trace file, followed by the omission of frames that the receiver trace file indicates were lost or corrupted. ET also requires the size of the play-out buffer to be set.

Assessment of video quality is performed on a frame by frame basis. **FV (Fix Video)** [15] ensures that the total number of frames in the reconstructed received video is the same as the original encoded

video. If the codec cannot handle missing frames, as is the case with the reconstructed video, then FV inserts the last successfully decoded frame in place of each lost frame to conceal the error.

The most significant portion of video quality assessment in EvalVid is the **PSNR (Peak Signal to Noise Ratio)** component. PSNR in EvalVid is defined as follows [15] [61]:

$$PSNR(n)_{dB} = 20log_{10}\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\Sigma_{i=0}^{N_{col}}\Sigma_{j=0}^{N_{row}}[Y_S(n,i,j)-Y_D(n,i,j)]^2}} \quad (5\text{-}2)$$

$V_{peak} = 2^k - 1$, where k represents the number of bits per pixel (luminance component).

Y_S and Y_D represent the source and destination pixels respectively. N_{col} and N_{row} represent pixel coordinates in the image.

As stated in Section 5.1, PSNR is used to calculate the error (difference) between a reconstructed image and an original one. Calculated on a frame by frame basis and averaged to determine a value for the full video sequence, PSNR provides a measure of the difference between the original video and the reconstructed video at the receiver. A higher PSNR value indicates receipt of a good quality video, or, in other words, one that is similar to the original video. Lower PSNR values are an indication of poor reconstruction due to a high number of lost frames. A better understanding of PSNR values is given by the Mean Opinion Square (MOS) as shown in Table 5-3.

Table 5-3 PSNR and MOS Mapping [15]

PSNR [dB]	MOS
>37	5 (Excellent)
31-37	4 (Good)
25-31	3 (Fair)
20-25	2 (Poor)
<20	1 (Bad)

The MOS table gives a representation of human perception to PSNR evaluation of video quality [15]. MOS started as a measure of quality by telephony networks to determine user perception of audio quality [64]. Use of this subjective measurement in image analysis ensures its applicability in the field of video analysis. Use of PSNR and MOS forms an important part of the QSQS presented in this dissertation.

5.2.2 Changes to EvalVid in Chih-Heng *et al* [15] to incorporate NS2

To replace the simple error simulation model in EvalVid with the more complex NS2, 3 agents are developed in Chih-Heng *et al* [15]. 'MyTrafficTrace', 'MyUDP' and 'MyUDPSink' are written to integrate NS2 into EvalVid. Requirements of these agents are to read the video trace file, and generate the data required to evaluate the quality of the delivered video. A representation of the new environment including NS2 and EvalVid is given in Figure 5-2 [15].

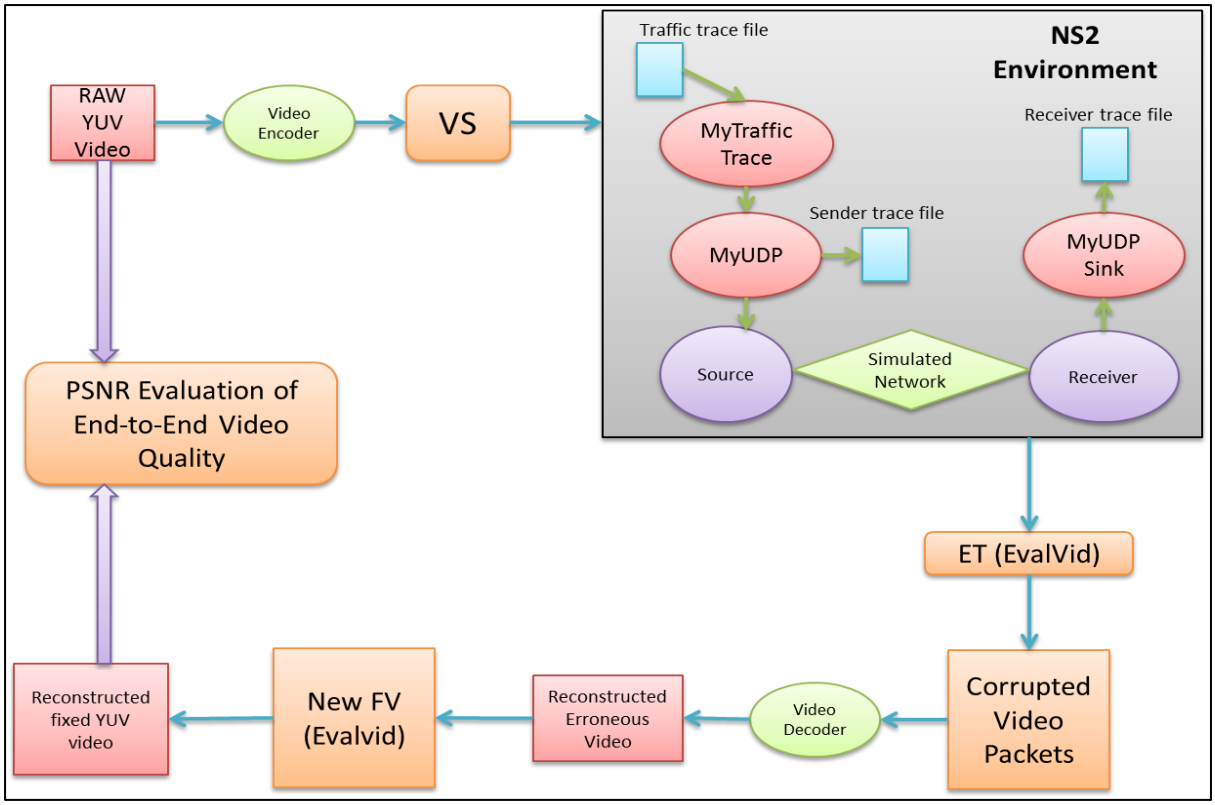

Figure 5-2 Integration of NS2 into EvalVid replacing error simulation model [15]

Explanation of the setup of this evaluation system is given in Section 5.3. The 3 agents added to NS2 each perform an important role. An explanation of the agents developed by Chih-Heng *et al* [15] follows. 'MyTrafficTrace' is responsible for frame type and frame size extraction of the video trace file from the traffic trace file generated by EvalVid's VS component. This agent also fragments the video frames into smaller segments. Segment sizes are specified by the user settings in the simulation script file. Also scripted by the user is the time at which MyTrafficTrace must send the segments to the lower UDP layer [15].

The native UDP agent in NS2 is modified and renamed 'MyUDP' agent by Chih-Heng *et al* [15]. It is tasked with allowing users to specify the output file name of the sender trace file and recording the timestamp of each transmitted packet, packet ID, and packet payload size. The role that 'tcpdump', explained in Section 5.2.1, plays in a real network is similar to that of 'MyUDP' in the simulated environment.

Normally in NS2, UDP agents connect to a sink agent, which is responsible for receiving data. In the case of this system, agent 'MyUDPSink' is the receiving agent for fragmented video packets sent by MyUDP. Whereas 'MyUDP' records the timestamp, packet ID and payload size of each sent packet in the sender trace file, 'MyUDPSink' is responsible for recording the same information, but for each received packet, in the receiver trace file.

Thereafter, ET makes use of the sender and receiver trace files, in addition to the original encoded video to produce a corrupted video. Decoding of the corrupted video commences, followed by the Fix Video component, after which a reconstructed fixed YUV video is produced. This video can then be compared to the original YUV sequence by means of PSNR, thereby evaluating both network and video quality.

One more program is developed by the researchers in Chih-Heng *et al* [15]. A problem with EvalVid's Fix Video (FV) program is defined. Video Object Plane (VOP) not coded bits exist for videos with lost frames, resulting in those frames being called vop-not-coded frames. Error concealment is dealt with by a few decoders, but decoders like ffmpeg that do not deal with it, decode all frames except for vop-not-coded frames. This is where the FV program is designed to handle vop-not-coded frames by replacing the lost frame with the last successfully decoded frame. Decoders like Xvid or NCTU may fail to decode the frames following the vop-not-coded frame, because the following frame may depend on the vop-not-coded frame, resulting in a lack of information to decode the subsequent decodable frames [15]. A new program is developed by the researchers, called 'MyFixYUV', which improves on the old FV component. The decoder output is used to fix the reconstructed erroneous video sequence. If a frame is decodable, the new FV program copies the decoded frame from the reconstructed erroneous raw video file into a temporary file. This frame is also stored in a buffer which keeps it as the last successfully decoded frame data. When a vop-not-coded frame is encountered, the new program reads its frame data from the reconstructed erroneous raw video file, but data is not stored in the temporary file as it is useless, and is discarded, with the pointer moving to the next frame in the file. Frame data in the buffer of the last successfully decoded YUV frame is written to replace vop-not-coded frames in the temporary file. Finally, when all frames have been processed, the resulting temporary file is the reconstructed fixed YUV sequence, ready for PSNR evaluation.

5.3 NS2 and EvalVid Simulation System Setup

Assistance with compilation of the components discussed in Section 5.2.2 is provided in Chih-Heng [65]. A discussion of the complete simulation system setup is given in this section.

With NS2 systems discussed in prior sections, Linux was used as the Operating System platform for development. However, the option of using either Linux Ubuntu or Windows XP was presented with the setup of this simulation system. As familiarity was gained with NS2 in Linux, use of NS2 in Windows was performed.

An Intel i7 Quad Core Desktop Computer with 4 GB RAM and 2 TB HDD was setup with Microsoft Windows XP, 32 bit, Service Pack 3 as the primary operating system.

5.3.1 Cygwin Setup

NS2 is developed primarily for UNIX based systems. Use of NS2 in Windows XP requires the installation of Cygwin. Cygwin is a collection of tools which provide a Linux environment within Windows. At its core, Cygwin is a Dynamic Link Library (DLL) which acts as a Linux Application Programming Interface (API) providing a useful range of Linux functions. It is also notable that Cygwin does not run native Linux applications on Windows. All applications must be rebuilt from source within Windows. [66]

Cygwin can be downloaded from www.cygwin.com [66]. A setup file is downloaded, which gives users an option to download the packages required instead of the whole Cygwin database. To run EvalVid and NS2 in Cygwin, the following packages were downloaded and installed, in addition to basic packages that are essential to Cygwin:

- GCC C Compiler
- GCC C++ Compiler
- GNUPlot (A command Line Graphing Utility for Linux)
- Make (Program Compilation Utility)
- More (Text Package)
- Utility Patches
- Perl
- XFree86-startup

5.3.2 NS2 in Cygwin Setup

NS2.27 all-in-one package was downloaded. The following procedure was performed to install the complete NS2.27 package in Cygwin.

Once the tarred (zipped) package was copied to the folder 'C:\cygwin\home\wayne', Cygwin was opened and the following command executed:

```
$ tar xvfz ns-allinone-2.27.tar.gz
```

This decompressed the NS2.27 packages and stored them in the folder 'ns-allinone-2.27'. After navigating into the folder, installation was started with the command:

```
$ ./install
```

Thereafter, bash scripts were configured with the relevant path information to alert the shell on the locations of NS, NAM and Xgraph. The following command is entered to start the shell environment in Cygwin so that programs like NAM can be executed:

```
$ startxwin.bat
```

Installation of NS was tested with the command:

```
$ ./ validate
```

All tests ran successfully and NS was fully operational in the Cygwin environment. An example using NAM was tested using shell command prompt:

```
$ ns firstexample.tcl
```

5.3.3 EvalVid in NS2 Setup

The required EvalVid source codes were downloaded from Chih-Heng [65]. Included in the downloaded files were the scripts for the NS2 and EvalVid connecting agents, 'MyUDP', 'MyUDPSink' and 'MyTrafficTrace'. Source codes for EvalVid programs ET (Evaluate Trace), MP4 (Video Sender), mpeg4encoder, mpeg4decoder, myfixyuv and PSNR included in the download had to be compiled.

Changes to NS native files 'agent.cc', 'agent.h' and 'packet.h' had to be made to incorporate the new agents connecting EvalVid and NS2. Changes to these files were to alert NS of the existence of the new UDP packet type and a new Traffic Trace agent. The modified files were provided by the researchers in Chih-Heng *et al* [15] and were used to overwrite the existing files found in the directory 'ns-allinone-2.27\ns2.27\common'.

Following this, a new folder 'mympeg' was created in 'ns-allinone-2.27\ns2.27\'. Source code for the new agents, 'mytraffictrace.cc', 'myudp.cc', 'myudp.h', 'myudpsink.cc' and 'myudpsink.h' were stored in 'mympeg'.

Addition of a new agent in NS usually requires setting of default parameter values in case the user does not provide these values in the Tcl simulation script. 'ns-default.tcl' found in the directory 'ns-allinone-2.27\ns-2.27\tcl\lib', is edited to include the following:

```
Agent/myUDP set packetSize_ 1000
Tracefile set debug_ 0
```

Thereafter, recompilation of NS is required to install the new agents and allow communication with EvalVid. Before recompilation, 'Makefile', found in the NS home directory, must be edited to include the following:

```
In OBJ_CC=\
mympeg/myudp.o mympeg/myudpsink.o mympeg/mytraffictrace.o \
```

Finally, in the Cygwin environment, the following commands must be executed to rebuild NS with the new features in the directory 'ns-allinone-2.27\ns-2.27':

```
$ make clean
$ make
```

Once recompilation is complete, NS2 can now be used with EvalVid to perform video quality analysis.

In Section 5.4, this evaluation system is used extensively to understand the behaviour of centralized and decentralized media streaming architectures. Thereafter, a complete explanation of development of the final QSQS is given, based on the understanding gained in Section 5.4.

5.4 Understanding Centralized and Decentralized Media Streaming Architectures by Means of Video Quality Evaluation

Experience had to be gained in utilisation of the Video Quality Evaluation system setup in Section 5.3. It was evident that a QSQS for media streaming architectures could be developed using NS2 and EvalVid, but setup of such a system required more groundwork to be performed.

5.4.1 Testing the Video Quality Evaluation System

It was decided that the same client-server topology (single client), which was used in earlier trace driven simulations, would be used for the first test. A standard raw YUV format test video from YUV Traces [58] was downloaded for this example, 'foreman_qcif.yuv', 400 frames long with resolution 176 x 144.

Firstly, the raw YUV sequence must be encoded to mpeg format. This is achieved using the video encoder 'mpeg4encoder.exe' from MPEG [62], by executing the command:

```
$ mpeg4encoder.exe example.par
```

'example.par' is a parameter file containing information needed by the encoder on the raw YUV video that is to be encoded. 'foreman_qcif.yuv', 14.85 MB large, is compressed to a 'cmp' (mpeg encoded file) file 324 KB in size. A screenshot of the encoding procedure is shown in Figure 5-3.

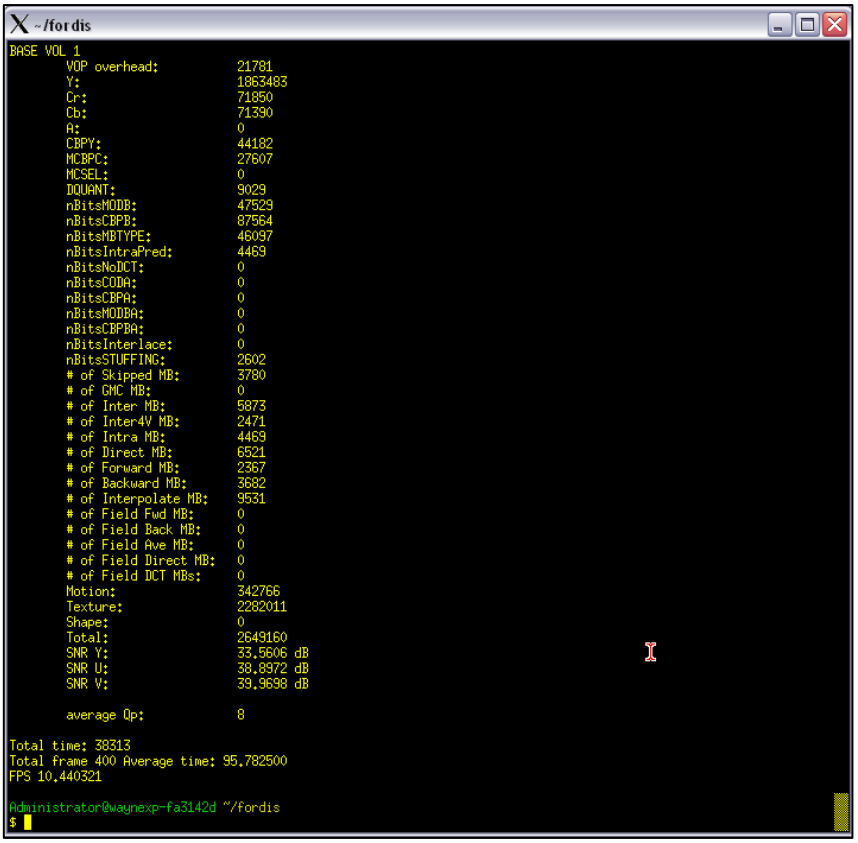

Figure 5-3 MPEG Encoding Shell Output

The tracefile sender is recorded by EvalVid Video Sender component 'MP4.exe', using the following command:

```
$ MP4.exe –send 224.1.2.3 5555 1000 foreman_qcif.cmp > st
```

The Video Sender fragments the encoded video frames into 1000 byte sized packets, thereafter transmitting these packets to the specified address. Text file 'st' records this information to form the traffic trace file, for input to NS2. In this test example, MP4.exe outputs that 549 packets were sent to the address via UDP.

Once the traffic trace file 'st' has been generated, a Tcl script (Test.tcl) specifying the media streaming network in NS2 must be written. An example script is discussed here highlighting the use of the new agents.

The client-server topology is generated by the following lines:

```
set s1 [$ns node]
set r1 [$ns node]
set r2 [$ns node]
set d1 [$ns node]

$ns duplex-link $s1 $r1 10Mb    1ms DropTail
$ns duplex-link $r1 $r2 0.18Mb 10ms DropTail
$ns duplex-link $r2 $d1 10Mb    1ms DropTail
```

The server is represented by ‘s1’ with a single client ‘d1’ and two routers ‘r1’ and ‘r2’. UDP packet size is set to 1028 bytes with 8 bytes for the UDP header and 20 bytes for the IP address header. The queue limit between the routers is also set to 10.

Attaching new agents MyUDP and MyUDPSink to the server and client is performed by the following lines:

```
set udp1 [new Agent/myUDP]
$ns attach-agent $s1 $udp1
$udp1 set packetSize_ $packetSize
$udp1 set_filename sd

set null1 [new Agent/myUdpSink]
$ns attach-agent $d1 $nulll1
$ns connect $udp1 $null1
$null1 set_trace_filename rd
```

‘udp1’ is an instance of agent MyUDP and is attached to server ‘s1’. The name of the sender video trace file is also specified by the user as ‘sd’. Likewise, ‘null1’ is an agent of MyUDPSink, attached to client ‘d1’ with receiver trace file ‘rd’.

The next important phase is to create a trace file in NS2 that copies the traffic trace file ‘st’ sent by EvalVid component ‘MP4.exe’. It is then required to extract the required information, frame number, type, size and length from ‘st’. The following lines in Tcl perform this task [65]:

```
set original_file_name st
set trace_file_name video1.dat
set original_file_id [open $original_file_name r]
set trace_file_id [open $trace_file_name w]

set frame_count 0
set last_time 0

while {[eof $original_file_id] == 0} {
      gets $original_file_id current_line
      scan $current_line "%d%s%d%s%s%s%d%s" no_ frametype_ length_ tmp1_
tmp2_ tmp3_ tmp4_ tmp5_
      set time [expr 1000 * 1000/30]

      if { $frametype_ == "I" } {
                set type_v 1
                  }
      if { $frametype_ == "P" } {
                set type_v 2
                  }
      if { $frametype_ == "B" } {
                set type_v 3
                  }
      if { $frametype_ == "H" } {
               set type_v 1
                  }
```

```
        puts $trace_file_id "$time $length_ $type_v $max_fragmented_size
        incr frame_count
}

close $original_file_id
close $trace_file_id
set end_sim_time [expr 1.0 * 1000/30 * ($frame_count + 1)  / 1000]
                            puts "$end_sim_time"

set trace_file [new Tracefile]
$trace_file filename $trace_file_name
set video1 [new Application/Traffic/myTrace]
$video1 attach-agent $udp1
$video1 attach-tracefile $trace_file
```

'video1.dat' is written with information from each frame: time, length, type and maximum fragmented size. Simulation time is also calculated based on the number frames read from the traffic file 'st', and the fact that the video will be played back at 30 frames per second. Thereafter, 'video1' is a tracefile created as an instance of the myTrafficTrace agent, and all the information from 'video1.dat' is stored in this sending trace file. Sender trace file 'video1' is then attached to agent 'udp1'.

The video is now ready to be sent from server to client via UDP. NS is then instructed on times at which to start and end the simulation.

```
$ns at 0.0 "$video1 start"
$ns at $end_sim_time "$video1 stop"
$ns at [expr $end_sim_time + 1.0] "$null1 closefile"
$ns at [expr $end_sim_time + 1.0] "finish"
$ns run
```

This script is executed by the command:

```
$ ns test.tcl
```

Once the simulation run has completed, two files, 'sd' (sender trace file) and 'rd' (receiver trace file) are generated by NS. These files are then sent to EvalVid component ET (Evaluate Trace) for processing. Trace evaluation is executed with the following command:

```
$ et.exe sd rd st foreman_qcif.cmp error.cmp 1
```

Figure 5-4 shows the output of ET.

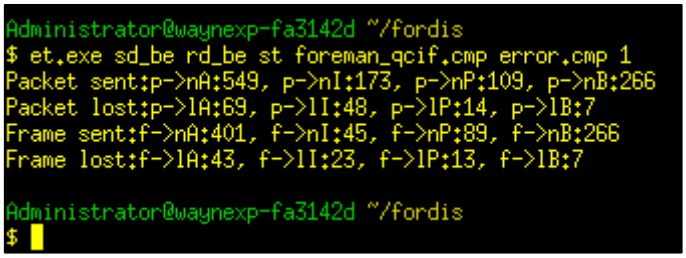

Figure 5-4 Output from Evaluation of Sender and Receiver Trace files, with generation of corrupted video sequence error.cmp

Figure 5-4 illustrates the number of lost packets and frames including the types of packets and frames lost, as evaluated by ET. 'error1.cmp' is the encoded reconstruction of the received video.

Decoding of the video can now be carried out to create a raw erroneous YUV image using the 'mpeg4decoder' program, from MPEG [62].

```
$ mpeg4decoder.exe error.cmp error 176 144 > df
```

Decoding status is sent to a file where information on vop-not coded frames is stored. Finally, the decoded YUV sequence is fixed with 'myfixyuv.exe' to replace vop-not-coded frames with last successfully decoded frames.

```
$ myfixyuv.exe df qcif 400 error.yuv fixed_video.yuv
```

The original video 'foreman_qcif.yuv' can now be compared to the received video 'fixed_video.yuv' by means of PSNR:

```
$ psnr.exe 176 144 420 foreman_qcif.yuv fixed_video.yuv > psnr_fixed_video
```

Output file 'psnr_fixed_video' stores the output of PSNR for every frame of the decoded video. The output results, plotted with Xgraph, are seen in Figure 5-5.

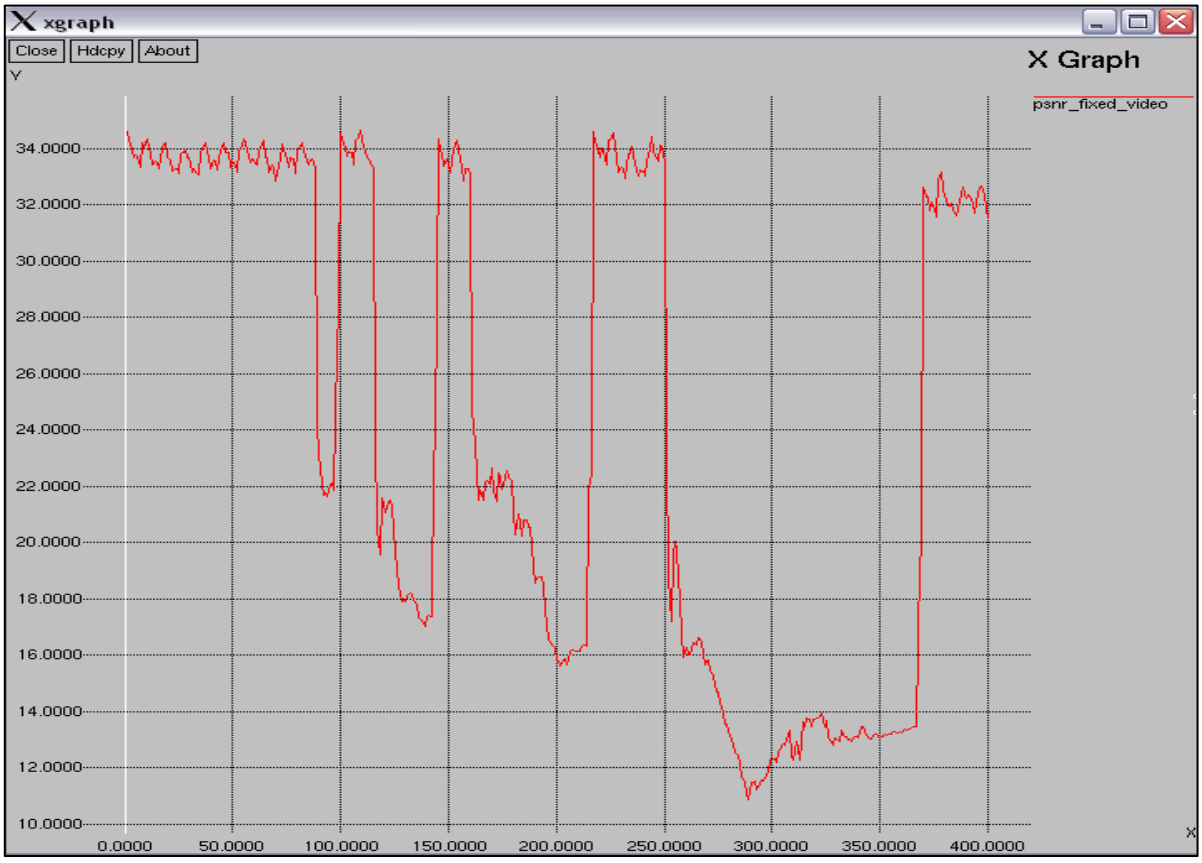

Figure 5-5 PSNR of Transmitted Foreman Video, with X-axis depicting frame numbers and Y-axis depicting PSNR in dB

Figure 5-5 shows the quality drop in PSNR for some frames of the video due to low bandwidth. Observations of reductions in PSNR below 22 dB indicate very poor quality as the video progresses. A downward trend in PSNR as the video progresses is observed with spikes above 30 dB. This can be attributed to router buffer overflow and low streaming bandwidth. When the buffer clears, video quality increases as seen by the PSNR spikes.

5.4.2 Determining Optimal Bandwidth for Different Standard Videos

Another video, 'akiyo_qcif.yuv', was downloaded from YUV Traces [58]. Tests on this video were carried out similarly to the way in which 'foreman_qcif.yuv' was tested. The aim in this section for both videos was to determine the optimal bandwidth at which both videos could be streamed in a single client-server network without any losses.

Table 5-4 Optimal Bandwidth of Test Videos

Test Video	**Optimal (Lossless) Bandwidth (Mbps)**
Akiyo_qcif.yuv	0.10
Foreman_qcif.yuv	0.25

Determination of this bandwidth between routers is used for development of topologies in later sections. The optimal bandwidth for 'foreman_qcif.yuv' was found to be higher than 'akiyo_qcif.yuv' as it had more movement and thus required a higher average streaming rate. PSNR of the received 'akiyo' video is shown in Figure 5-6.

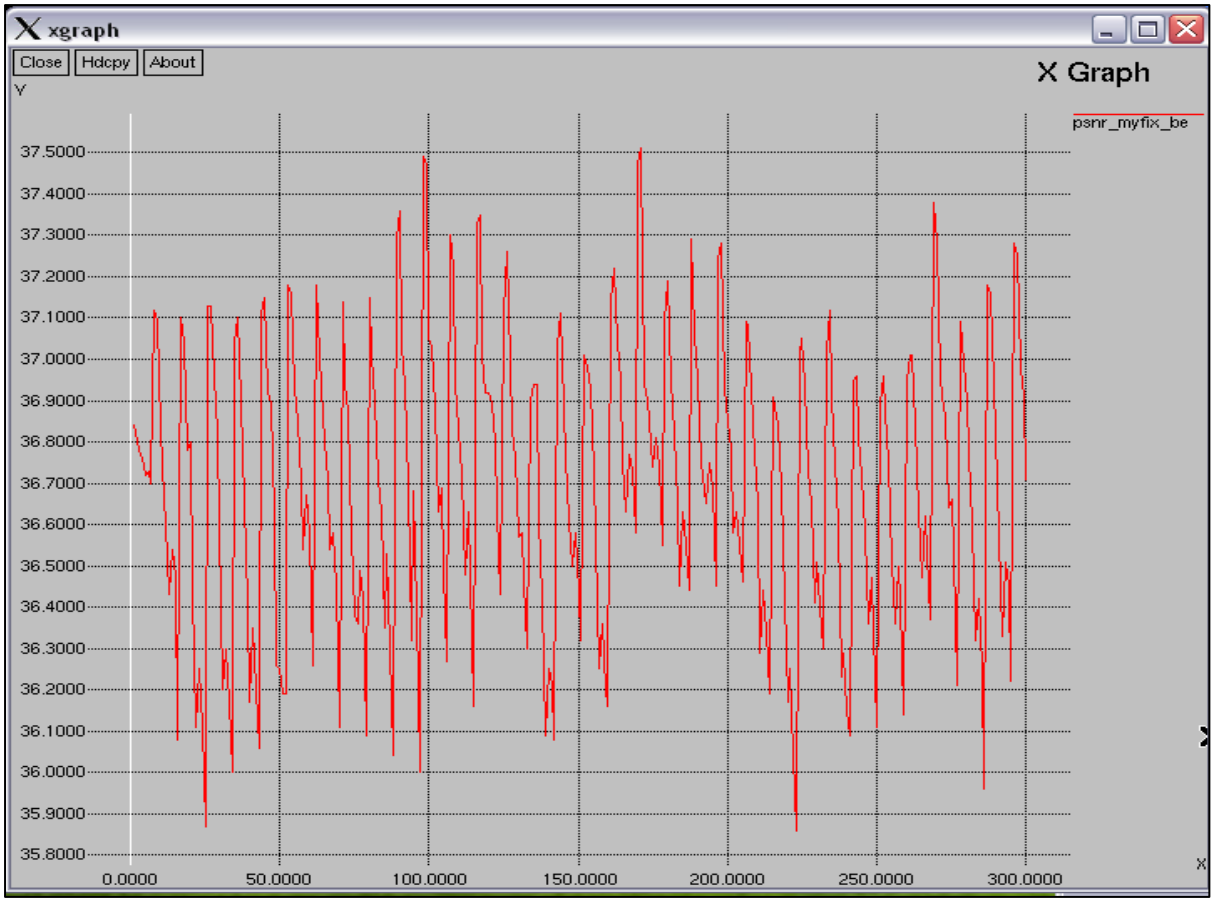

Figure 5-6 PSNR of transmitted Akiyo video with Optimal Bandwidth, with X-axis depicting Frame Numbers and Y-axis depicting PSNR in dB

PSNR values of the video range from 35 dB to 37.5 dB, which is representative of excellent quality video as outlined by the subjective MOS rating. An increase in PSNR indicates higher quality video streaming than that observed in Figure 5-5, due to the increase in bandwidth. The router buffer does not overflow, causing minimal loss of data packets.

5.4.3 Increasing number of clients in the Central Simulated Network

Number of clients in the client-server topology is increased to 2 for the 'akiyo' video, with a bandwidth of 0.10 Mbps. During simulation, both clients are served simultaneously. However, large frame losses are experienced. Figure 5-7 shows the PSNR of received video at each client, with the first client in red and second client in green.

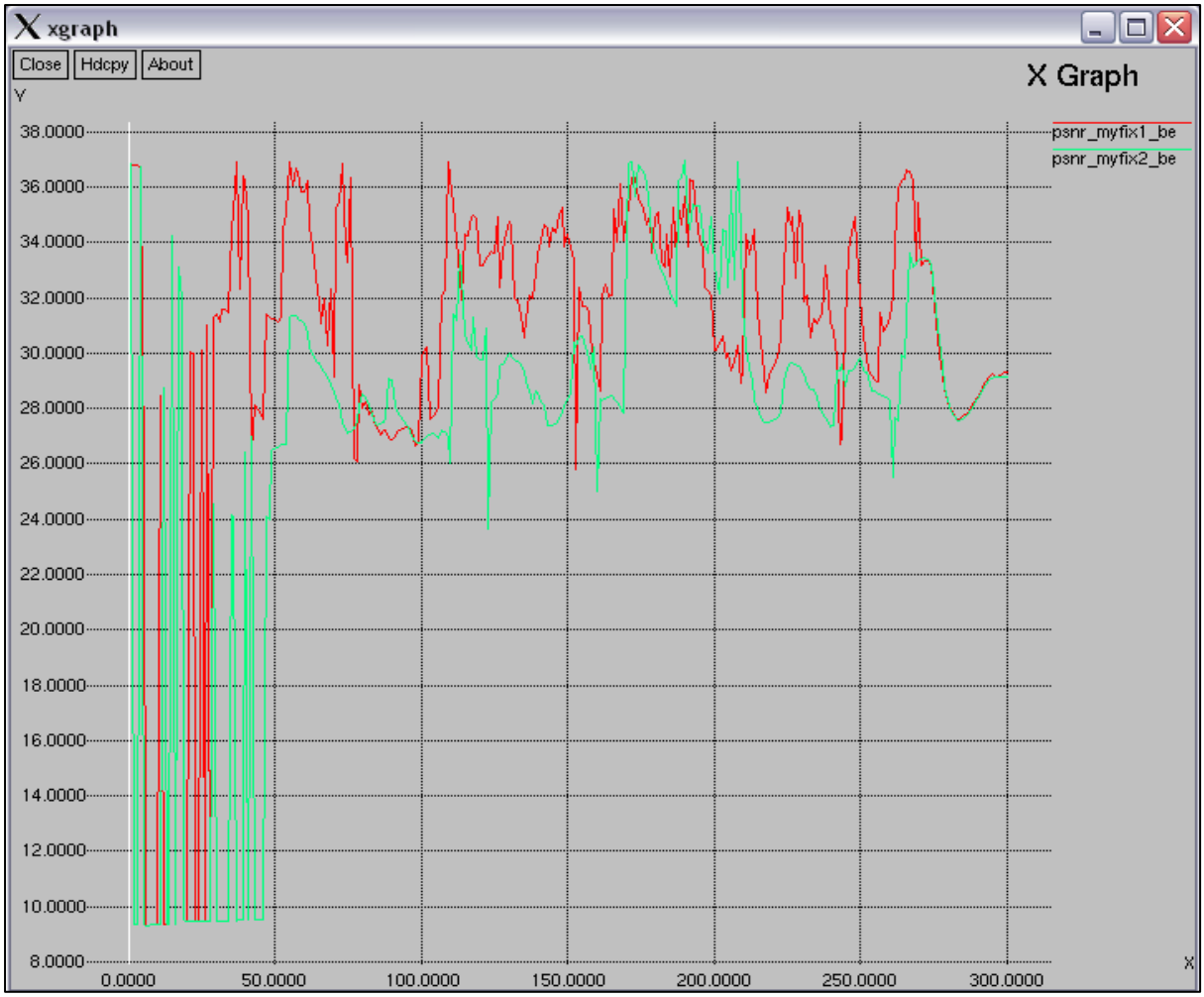

Figure 5-7 Comparison of received video quality at each client, with X-axis representing frame numbers and Y-axis representing PSNR in dB

Figure 5-7 depicts the drop in quality experienced by both clients with a server that possesses the same uplink bandwidth for optimal delivery to a single client. PSNR values now range from about 9 dB to 37 dB, indicating a maximum quality drop in transmission of about 25 dB when compared to optimal transmission of the 'akiyo' video. Thus, it is apparent that introduction of a new client caused a drop in quality, as the server's uplink bandwidth was insufficient to serve two clients. Table 5-5 depicts the observations made and a discussion of experimentation follows.

Table 5-5 Changes in Centralized Bandwidth Requirements for Increasing Clients Requesting Akiyo Video

Bandwidth (Mbps)	**1 client**	**2 clients**	**3 clients**
0.10	No Losses	High Losses	Very High Losses
0.20	No Losses	Medium Losses	High Losses
0.25	No Losses	No Losses	Medium Losses
0.57	No Losses	No Losses	No Losses

Following this, examination of the optimal bandwidth for two clients was performed. It was assumed that 0.20 Mbps uplink bandwidth for two clients would be sufficient. However, losses were still experienced in the network with a linear increase in bandwidth in terms of number of clients.

Trial and error led to 0.25 Mbps bandwidth as optimal for distribution to two clients. At this bandwidth, no losses were experienced in transmission and quality of the received video was indiscernible from the original source video.

Non-linear behaviour of optimal bandwidth was experienced with the 'foreman' video as well. It is noted that during physical network implementation, discussed in Section 4.6, similar non-linear bandwidth usage was observed. Observation of this behaviour in both simulated and physical networks verifies the validity of simulations performed. This non-linear behaviour is attributed to the FIFO queuing protocol.

Thereafter, three clients were streamed the 'akiyo' video with a server uplink bandwidth of 0.25 Mbps. As expected, high frame loss rate was experienced and PSNR values dropped significantly. In this experiment, the first client experienced slightly degraded results compared to the previous example with two clients. However, received video at the second client was worse than the previous example, with the third client's received video having poor quality.

A pattern emerged which showed that introduction of new clients had a negative impact on quality experienced by existing clients, with new clients having the worst quality, as expected. The order in which clients are served gives them the status of first client to new client. Even though all clients make requests in these scenarios at the same time, the client declared first in the script is given priority over one declared after.

Serving the 'akiyo' video to three clients without losses required a server uplink bandwidth of 0.57 Mbps. The non-linearity of optimal uplink bandwidth requirement against number of clients served was exhibited.

The architecture experimented with meets specifications of the basic topology layout of a client-server system and the standard way in which videos are streamed by that system.

5.4.4 Development of Decentralized Topology and Comparison to Centralized Topology

The first attempt of decentralized topology development was a three node system, consisting of two sending peers and one receiving peer. Each peer was connected to a central router, with a duplex link bandwidth of 0.25 Mbps between each peer and the router, and each link queue set at 10. In this system, one peer sends the 'akiyo' video while the other peer simultaneously sends the 'foreman' video to the receiving peer. Performance of this system, based on received video quality, was good as a small fraction of frames were lost during simultaneous transmission of both videos.

Optimal bandwidth was found to be 0.28 Mbps for the receiving peer link with the other peer links at 0.25 Mbps. Table 5-6 depicts the results from centralized and decentralized simulation topologies.

Table 5-6 Serving Akiyo and Foreman Videos to a Single Client for Both Topologies

Bandwidth (Mbps)	Centralized Topology	Decentralized Topology
0.25	High Losses	Low Losses
0.28	Medium-High Losses	No Losses
0.37	No Losses	No Losses

At this stage, a centralized and decentralized topology had been developed. The same request model as the decentralized system above was applied to a single server, single client centralized topology. This was to determine the optimal uplink bandwidth for a single server serving both 'akiyo' and 'foreman' videos simultaneously to a client. With a server uplink bandwidth of 0.28 Mbps, found to be the optimal bandwidth of the decentralized topology, high frame losses for both videos was observed. Lossless transmission was achieved with an uplink bandwidth of 0.37 Mbps. The higher bandwidth requirement of a central server in this comparison was the first observation of increased efficiency associated with decentralized topology, which is based on the division of responsibility principle.

A revised method of testing the decentralized topology is application of the division of responsibility principle to both bandwidth and storage amongst peers. To achieve this, the combined storage space of all peers in a decentralized topology should match the storage space of the server in a centralized topology which it is being compared to. For example, if a server was streaming the 'foreman' video, 14.85 MB large, it is assumed to have a storage capacity of 14.85 MB. In a decentralized network with two peers serving one peer, each serving peer should have storage capacity approximately half that of the central server, i.e. 7.43 MB. Storage division can be achieved through fragmentation of the video being served.

The source video was fragmented, using EvalVid software, into groups of frames. In the file specifying video parameters to the MPEG4 encoder, a video can be split by encoding the frames in separate batches. For example, in the case of 'akiyo', which is 300 frames long, one parameter file would specify the first 150 frames of the source video to be encoded, while the other parameter file specifies frames 151-300 for encoding. From this point, each batch of frames is treated as an individual video within EvalVid and NS2, with recombination of the constituent received video files to form the final output video.

To test the performance of division of storage, a decentralized topology with three serving peers and one receiving peer is scripted. The 'foreman' video, which has higher bandwidth requirements, is chosen. The video is split into three pieces of 133 frames each. Each serving peer

sends one piece to the receiving peer. Fine tuning of the decentralized topology had to be performed at this point to find the optimal bandwidth. Results obtained are depicted in Table 5-7 and a discussion follows.

Table 5-7 Decentralized topology performance serving Foreman Video split into 3 parts with an increasing number of clients

Bandwidth (Mbps)	Queue Size	1 Client	2 Clients
0.25	10	High Losses	High Losses
0.70	10	No Losses	No Losses
0.50	100	No Losses	No Losses

Using 0.25 Mbps links between each peer and the central router, performance was poor as the link to the receiving peer was overloaded, receiving three times the content that a central server would have sent in the same time period. The decentralized system sends the complete video to the receiving peer at a fraction of the time the central server takes to send the complete video, as various pieces of the video are sent simultaneously by different peers.

With an increase in bandwidth links of all peers to 0.70 Mbps, and a queue size of 10, there were no losses in the decentralized topology. This bandwidth was lossless but higher than the optimal bandwidth. It was also discovered that maintaining 0.50 Mbps links from each peer to the router, with a queue size of 100 for each link, resulted in lossless transmission. However, queue size adjustment was not performed as the focus was kept on bandwidth.

With no changes to the specifications of the decentralized topology, the number of receiving peers was increased to two. A queue size of 10 and bandwidth of 0.70 Mbps was maintained for all links. Evaluation had shown that no frame losses had occurred, showing that the decentralized topology was able to handle twice the load with the same level of performance achieved when serving a single receiving client.

In order to validate the performance observed in the decentralized topology, a centralized topology with the same load was scripted. A single server had to serve the 'foreman' video to two clients simultaneously. A comparison of centralized and decentralized topologies is given in Table 5-8 for an increasing client base with the same 0.70 Mbps bandwidth links.

Table 5-8 Centralized and Decentralized Performance Comparison for an Increasing Client Base with the same bandwidth links

Number of Clients	Centralized Topology	Decentralized Topology
2	No Losses	No Losses
3	Medium-High Losses	Low Losses
4	High Losses	Medium Losses

With 0.70 Mbps bandwidth links and a queue size of 10 between the server and clients, no losses were observed in the centralized topology. Thereafter, increasing the load to three clients resulted in a fairly sharp decline in performance. This performance was evaluated so that a comparison could be made with a decentralized topology bearing the same load. Evaluation of a decentralized system with three serving peers and three receiving peers was then performed, maintaining 0.70 Mbps bandwidth links and queue sizes of 10. Lower losses were incurred in this topology compared to the performance of the centralized topology with a matched load. It was observed that frame types of low priority were lost in the decentralized topology, compared to loss of high priority frames in the centralized topology. This equates to higher information loss in the centralized system, thereby resulting in lower PSNR values.

An increase to four receiving clients for both topologies resulted in a greater amount of information lost by the centralized topology and lower quality video. Results from PSNR analysis showed higher PSNR values under the same load were observed for a decentralized topology over a centralized topology.

5.5 Development of a QSQS for Comparison of Media Streaming Architectures

Work presented in Section 5.4 led to the development of a system that fully tests the requirements and performance of centralized and decentralized media streaming architectures. It is important to consider that the aim of developing such a system fits the main aim of this dissertation: To quantify the difference in quality and scalability of centralized and decentralized media streaming architectures. Employing the Video Quality Evaluation System presented in Section 5.2, a comparison between architectures can be made. Throughout this dissertation, a QSQS was discussed. However, this system was borne out of making a comparison between centralized and decentralized architectures, as will be discussed in this section. A methodology emerged in making the comparison that developed into a system that could be used for different media streaming architectures.

The P2P VoD Media Streaming system presented in Chapter 3 requires evaluation by this method. Evaluation of a purely P2P network employing some of the protocols in the full-scale P2P VoD system are evaluated. RTP with the UDP/TCP protocol is applied along with the division of responsibility principle. In addition to this, a Serving Peer protocol is evaluated while applying replication and buffering principles discussed in Section 3.

5.5.1 Comparing Centralized and Decentralized Media Streaming Architectures

Determination of the difference between centralized and decentralized architectures requires two architectures that can be fairly compared. Hardware criteria are established to make this

comparison. Two basic architectures, a purely centralized media streaming architecture and a purely decentralized one are scripted in Tcl. A discussion of the architectural designs and scripts for each system follows.

5.5.2 System Architecture and Hardware Equality

A generic centralized architecture, Figure 5-8, is shown with a fixed server storage capacity and arbitrary bandwidth links, in this case 1 Mbps for clients and 2 Mbps for the server, to the router. Speeds are chosen based on the quality of video that is required for transmission. Higher speeds are used for higher bandwidth requirement videos. This bandwidth was chosen here as it was sufficient for the chosen video for experimentation.

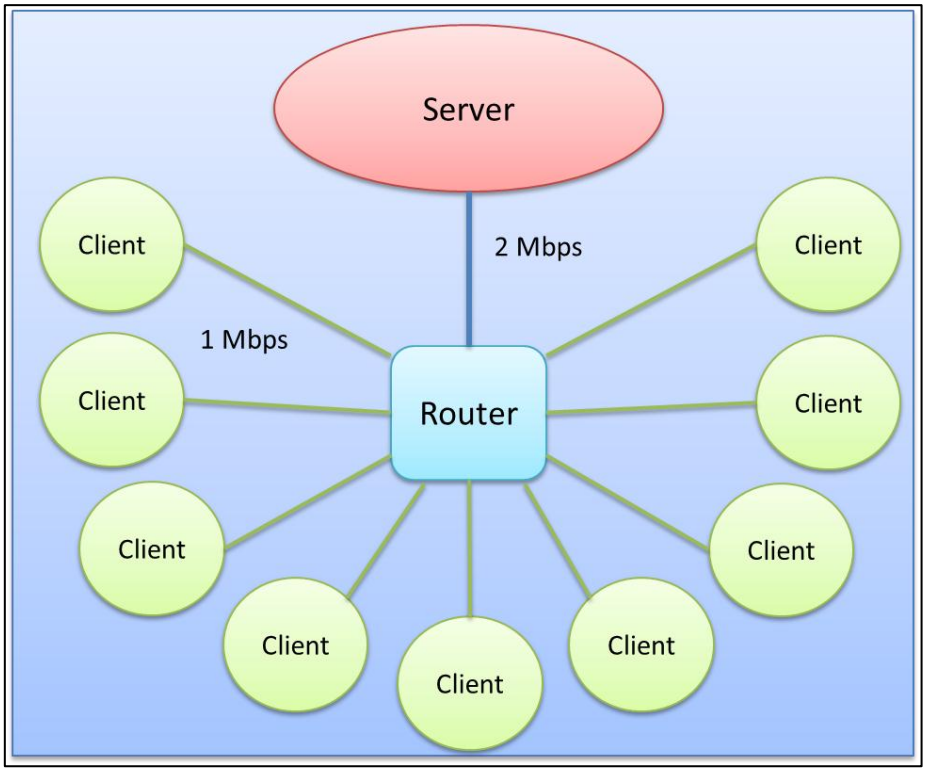

Figure 5-8 Topology Layout of Basic Centralized Architecture

A complete video is stored in the server for transmission to clients. For this architecture, server storage capacity is assumed to be the size of the video which is being streamed to clients. A Droptail queuing protocol, operating on a First-in-First-out basis, is employed in the system to process user requests.

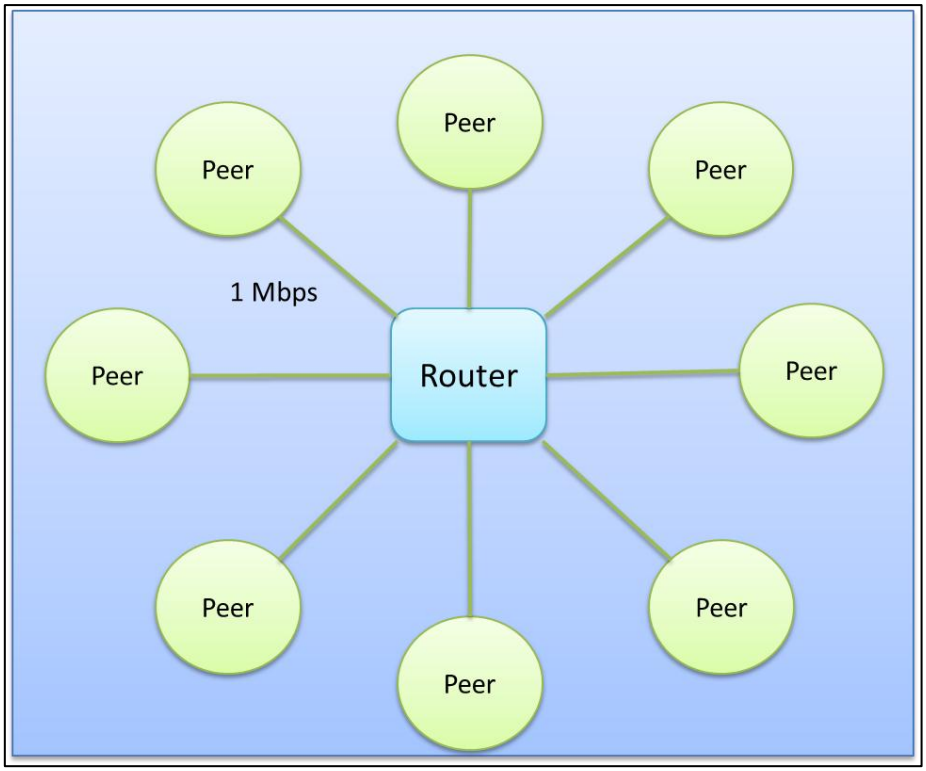

Figure 5-9 Topology Layout of Basic Decentralized Architecture

Figure 5-9 is a depiction of a decentralized analogue to the centralized architecture in Figure 5-8. Each peer has a 1 Mbps duplex link to the central router, thereby allowing each peer to be directly connected to every other peer. Storage capability in this system is distributed equally through the peers of the system. In other words, the same video streamed by the central server is segmented into a number of parts, equal to the number of network peers present. The concept of segmentation to maintain hardware equality decentrally was discussed in Section 5.4.4. Storage capacity of each peer is assumed to be the size of the video segment stored on them, ensuring that the combined storage of all peers in the decentralized architecture is equivalent to the central server capacity. To further enhance the division of responsibility principle decentrally, each peer request is served equally by every other peer.

5.5.3 System Development and Methodology for Quantifying Quality and Scalability of Each System

Poisson Request Generation

With the basic centralized and decentralized architectures specified, it was important to develop a request model for implementation in NS2 such that accurate network loads could be realised. A Poisson random request generator was developed to mimic user request behaviour. With Lambda as the mean, a typical Poisson distribution for four values of lambda is shown in Figure 5-10 [67].

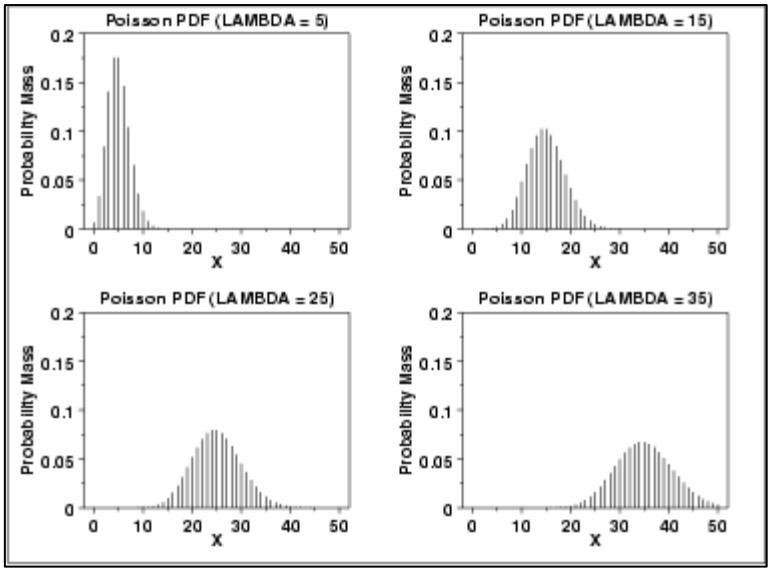

Figure 5-10 Poisson Probability Mass Functions

As demonstrated by Figure 5-10, the probability mass of X (number of user requests in this case) is heavily weighted toward lambda, the mean value. Hence, if Lambda is set equal to the number of clients centrally, or the number of peers decentrally, then the likelihood of generating one request per node per request cycle is high, i.e. number of requests generated is equal to number of nodes in the network. A new agent, named 'newagent3.cc' is written to add this functionality to NS2. Important sections of the C++ code are discussed here, while the complete code for 'newagent.cc' can be found in Appendix A. As with most agents in NS2, parameters are passed from the Tcl script to C++. In this case, 'Lambda' for the Poisson distribution is defined in the Tcl script as the number of clients or peers in the network. The following code extract shows how Poisson values are distributed based on input value Lambda.

```
int seed = time(NULL);
srand(seed);
```

Time is set as the seed for random number generation to ensure that each random value generated is distinct from every other random number. Function 'rand' is a uniform random number generator, with 'R' a uniform random number between 0 and 1. Thereafter, the Poisson number is generated using the uniform random number 'R' as the seed:

```
float R;
R= (float)(rand()*-1)/(float)(RAND_MAX+1);
int k=0;
const int max_k = 1000;
double P = exp(-lambda);
double sum = P;
if (sum>=R) result=0;

for (k=1; k<max_k; ++k) {
      P*=lambda/(double)(k);
      sum+=P;
```

```
        if (sum>=R) break;
        }
result=k;
```

The Poisson number generated by the above code is derived from the mathematical expression given by:

$$P\ (X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad (5\text{-}3)$$

Following this calculation, if $\sum_{k=1}^{1000} P(k) > R$, then output 'k'. Thus, it can be seen that 'k', the random output variable, is most often equal to Lambda, λ. The Poisson value generated is stored in 'example.txt', from which the Tcl script files reads this value.

```
tcl.eval("puts \"Message from Random Poisson Generator\"");
tcl.evalf("puts \" Poisson Number = %d\"",result);
ofstream myfile;
myfile.open ("example.txt");
myfile << result;
myfile.close();
```

In Tcl, the following script initialises the Poisson number generation agent:

```
set myagent [new Agent/TSPAgentOtcl]

#Set configurable parameters of MyAgent
$myagent set mean_otcl 5
$myagent set lambda_otcl $numnodes
```

Mean is input as an arbitrary value for uniform random number generation. '$numnodes' refers to the number of peers or clients in the network, and is set as Lambda.

Tcl Scripting of Centralized and Decentralized Media Streaming Architectures

Tcl scripting of the centralized and decentralized architectures outlined in Section 5.5.2 is discussed in this section. Full scripts of both central, 'central.tcl' and decentral, 'decentral.tcl' architectures can be found in Appendix A.

Scripts are written with reusability in mind so that parameters such as number of nodes can be adjusted to create new networks. Each simulation consists of ten runs, with each run generating a new Poisson request number. Ten is an arbitrarily high number, but is chosen based on hardware capabilities of the simulation workstation and improved accuracy of results obtained. A higher number of runs will give finer results, smoothing out variations and more credibly simulating real network behaviour.

The number of requests generated determines the number of nodes served. For example, if the network consists of ten nodes, and the Poisson request number is nine, then nodes one to nine are each served a single copy of the requested video. If the Poisson number is greater than the number of nodes generated, then nodes at the beginning of the set are served again, i.e. if twelve requests are

generated and ten nodes exist in the system, then nodes one to ten are each served the requested video, with nodes one and two served requests eleven and twelve after a certain time period during that run of the simulation. This aims to replicate the case where users make new requests while viewing their first request, thereby increasing network load.

Scripting of the decentralized architecture proved to be more difficult than the centralized architecture as video segmentation was troublesome to solve on a larger scale than that presented in Section 5.4. For the centralized architecture, a single traffic trace file, generated by EvalVid video sender component 'MP4.exe', had to be read from. Thereafter, video sender and video receiver trace files were generated for each request and stored on the simulation workstation for processing. The same had to be performed for the decentralized architecture, with a difference in the number of traffic trace files read in by NS2. In this case, number of peers in the network specified how many video segments existed, thereby directly influencing number of traffic trace files for the network. A traffic trace file can be thought of as each peer's stored content as it is representative of the video that is requested from that peer. Attachment of the traffic trace file to the 'myUDP' agent entails generation of the video sender trace file.

Each peer in the decentralized network has a piece of the video requested by them, as each video is segmented into equal parts amongst the number of peers in the network. Traffic trace files are thus attached chronologically to the peers, with the first video segment stored in the first peer, and so on. For example, if a peer in a network with ten peers makes a request, then it takes receipt of nine video segments. Receiver trace files are generated by agent 'MyUDPSink', attached to the requesting peer, or client in the case of centralized architecture. It is said that scripting of the decentralized architecture introduces further complexity, as is apparent with the case of 'MyUDP' sending agents, which are only attached to the server for each request, but attached to every peer in the decentralized network.

A high number of sender and receiver trace files are output from each simulation for processing with EvalVid components. However, a far higher number of these files are generated by decentralized networks, as each peer is involved in both sending and receiving. This created complexities with naming, which had to be solved dynamically.

Full Tcl scripts in Appendix A give a better understanding to the discussion in this Section.

Methodology for Quantifying Quality and Scalability

Batch processing had to be performed on generated sender and receiver trace files from simulations. To achieve this, bash scripting was employed along with simple modifications to perform post simulation processing with the enhanced EvalVid toolkit from Chih-Heng *et al* [15]. Bash scripts for post simulation processing are given in Appendix A, with slightly different scripts for

centralized and decentralized architectures. Figure 5-11 depicts the final evaluation system for network wide video quality analysis.

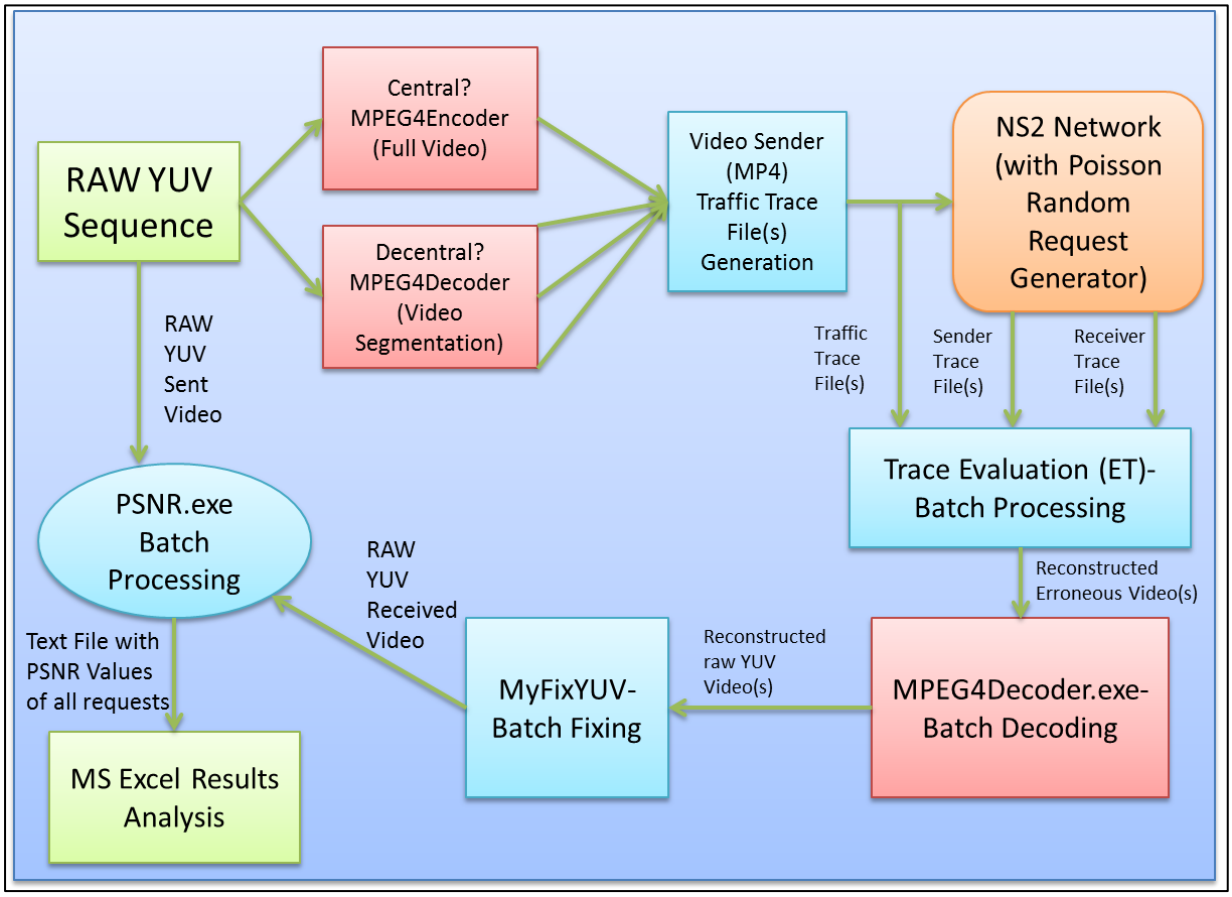

Figure 5-11 Final Evaluation System for Network-Wide PSNR Analysis

For a given simulation, each request has an associated sender and receiver trace file. After evaluation and generation of a fixed received raw YUV sequence, both sent and received YUV videos are sent to 'PSNR.exe' for evaluation. 'PSNR.exe' calculates the PSNR of each frame in the video and outputs an average PSNR representative of quality of the complete video. A modification was made to 'PSNR.exe' to record values to a text file.

For each centralized architecture simulation, a PSNR value for each request was generated and recorded in a text file. With PSNR values for video segments in the case of decentralized systems, PSNR of each segment constituting a video was recorded, and thereafter averaged accordingly to produce a PSNR value for the complete video streamed decentrally.

At the completion of processing, each simulation has a text file containing the request number and resulting PSNR value indicative of quality of that transmission. Microsoft Excel is used to

average the PSNR values from the text file to observe average video quality delivery across the network.

At this stage, it is possible to make inferences about quality of network transmission for different numbers of requests, etc. However, comparison of centralized and decentralized networks in terms of quality and scalability, requires a more controlled process.

Using the system described above, it is possible to accurately measure the scalability of decentralized systems over centralized systems using video quality as the metric. This method is outlined in Steps (a) to (e).

a) A simulation runs ten times where ten discrete random Poisson requests are generated. Each request generation occurs at a time equal to less than the complete time it takes to serve the full video in the case of the centralized system, and less than the time it takes to serve each piece of the video in the case of the decentralized system. This intrinsically increases system load as it attempts to mimic the unpredictable behaviour of users in the network.

b) Run the simulation for the centralized system starting with 2 clients. Once average PSNR for the network is obtained, run the simulation again with 3 clients. Continue to increase the number of clients in the network with each simulation until the PSNR value drops below a certain arbitrary value. This value was chosen to be 23 dB as Mean Opinion Square (MOS-subjective assessment) suggests that video quality would be poor at this point.

c) Apply the same principle as described in (b) to the decentralized system. For each simulation run, divide video frames evenly across peers in the network. This ensures that division of responsibility is applied fairly and total decentralized storage capacity does not exceed that of the centralized system. Run simulations from 2 peers to the same number of peers reached by the centralized system and record average PSNR for each simulation.

d) Repeat steps (b) and (c) at least two more times to average out results and obtain the highest possible accuracy for measurement in both systems.

e) An arbitrary PSNR value of 28 dB is chosen as MOS suggests this to be the threshold between good and fair quality. Using this PSNR value, determine the number of users served in each network. The difference between these numbers would be a good reflection of the difference in scalability between the centralized and decentralized architectures.

This five step method is referred to as the QSQS in this dissertation, and is employed to differentiate centralized and decentralized architectures. A discussion of results from application of this method is discussed in Chapter 6.

5.5.4 Serving Peer Protocol Implementation

Principles of the P2P VoD Media Streaming Architecture are applied to the basic decentralized architecture evaluated in Section 5.5.3. However, some of the protocols and agents employed by the P2P VoD system cannot be tested as time does not permit. Evaluation of quality and scalability gave an indication of performance of the P2P VoD system in terms of its decentralized nature, ability to control peers, video segmentation and division of storage. This section presents the development and evaluation of a serving peer protocol, as suggested in Section 3.11. An extended increase in the user base, or a large influx of requests, reduces quality of decentralized architecture video delivery. A 'central' nature appears in the decentralized system, as peer resources are overwhelmed by high request numbers, thereby requiring them to act as 'servers'. This problem was addressed in the proposal of the P2P VoD Media Streaming System, with a serving peer protocol suggested. The idea was adopted from Yifeng *et al* [28], where a 'Helper' protocol was proposed. Theory of serving peer protocols was discussed in Section 3.11. Most importantly, this strategy relies on content replication. A discussion of protocol implementation in NS2 follows.

This protocol makes the assumption that there are peers available with spare uplink bandwidth and the same stored content as those peers which have saturated uplink bandwidth. When the system load exceeds one request per peer during a simulation period, this protocol is deployed by the Control Server. A peer with spare uplink is then required to stream the video segment requested from the overloaded peer to the requesting peer.

Tcl scripting of this protocol required many changes to the Tcl script for the standard decentralized architecture presented in Section 5.5.2. A full Tcl script for employment of the serving peer protocol in the decentralized architecture is provided in Appendix A. Replication of content is performed in this script by creation of fictitious peers which store video segments. The number of fictitious peers created in the system is equal to half the number of actual peers in the system. Each fictitious peer stores two segments, thus creating a second copy of the complete video between these peers. When the number of requests for a simulation period exceeds the number of actual peers in the network, requests up to the number of actual peers are served by those actual peers, while serving peers are required to serve the requests over that threshold. For example, in a network of ten peers, five fictitious peers are created, with each newly created peer storing two segments. If less than ten requests are generated, then the original peers are required to serve each other. However, once the number of requests crosses the threshold of ten, then fictitious (serving) peers are activated, and

required to serve requesting peers with the remaining requests from that request generation. This method is not purely a serving peer protocol based on deficit bandwidth and can perhaps be regarded as a pro-active way of dealing with peer overload. However, results from implementation of the QSQS on the decentralized architecture in Section 5.5.3 validate this method as a deficit bandwidth implementation, as a threshold of one request per user per simulation period is apt, with a crossing of that threshold resulting in significant quality degradation.

There are possibly many other ways of performing replication and employing a serving peer strategy but this one was developed as it best suited available resources. Evaluation of the serving peer protocol is performed by adopting the Quality Quantification portion of the QSQS. Steps (a) to (c) evaluation of the serving peer protocol:

a) Run a normal decentralized architecture simulation for 6, 8 and 10 peers. Each simulation must be carried out for a medium load and a high load. A medium load is observed when Lambda, Poisson Random Request Generator parameter, is set to one and a half times the number of peers in the network.

b) Perform (a) again, but with the implementation of the serving peer protocol.

c) Once average PSNR values for each network have been obtained from the Quality Quantification System, plot medium load network performances for the normal decentralized architecture and compare these results to those of the decentralized architecture employing the serving peer protocol. Thereafter, make a plot of values obtained from high load networks of both architectures.

Firstly, an inference on effectiveness of the serving peer protocol can be made by observation of its effect under medium and high network loads. Secondly, difference in quality observed for high network loads with and without a serving peer protocol implementation is, in effect, a quantification of the protocol's ability to alleviate overloaded peers, and furthermore, on its ability to improve video delivery quality in the network. Results obtained from the method described here are presented and analysed in Chapter 6.

Chapter 6 Evaluation System Results

Analysis of results obtained from Sections 5.5.3 and 5.5.4 is performed in this chapter. A 2000 frame video standard video was chosen from YUV Traces [58]. A video with a high number of frames was chosen as this made segmentation simpler with an increase in number of peers in the decentralized architecture. Microsoft Excel was used to process PSNR data obtained from all simulations. The primary objective of this dissertation was quantification of scalability of centralized and decentralized media streaming architectures and a comparison between results thereof. Using video quality (PSNR) as a metric, a quantification of scalability was achieved with fairly accurate results. Use of the quantification system in evaluation of a serving peer protocol also produces results of interest which are analysed.

6.1 Quality of Centralized Architecture

Following the methodology in Section 5.5.3, average network wide PSNR values were obtained for an increasing number of nodes. Three iterations of simulations for each particular number of nodes are carried out, with each simulation consisting of ten iterations. Thus, each data point obtained is an average of multiple data points, allowing for finer results to be obtained.

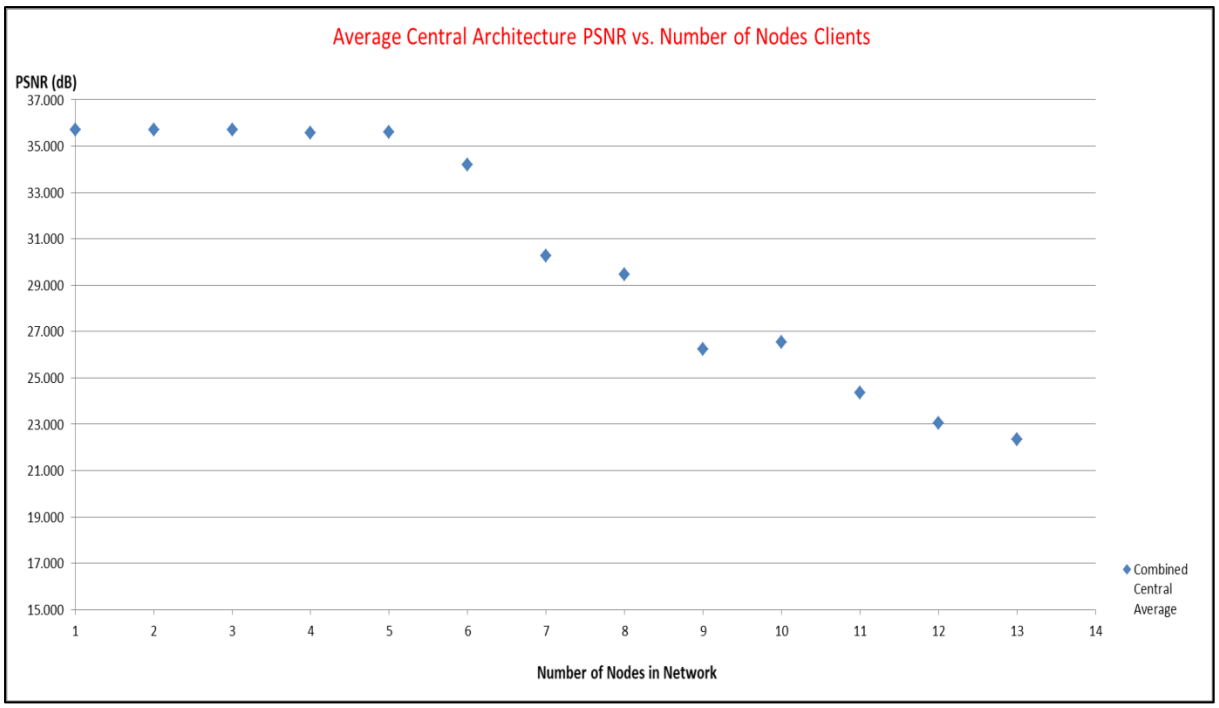

Figure 6-1 Network Averaged PSNR for Increasing Clients in Centralized Architecture

Figure 6-1 shows that server performance is good when serving up to 6 nodes, i.e. video quality reception at the end-user is indiscernible from source video content. However, as the load starts to increase, i.e. by increasing the number of clients requesting from the server, quality of video received steeply declines. This drop in video quality network wide is a direct result of saturated uplink bandwidth at the server, as packets containing important frames are lost or delayed. Further observation of Figure 6-1 shows that video quality network wide with nine clients is about 26 dB, considered fair to poor quality by MOS standards. The chosen value of 26 dB is important to the proposed Quantification of Scalability System discussed in Section 5.5.3.3. Beyond nine clients, video quality delivery by the centralized architecture tends to decline at a constant rate with an increase in the number of nodes. A tipping point, where the server falls over due to overloading, was not observed in the centralized architecture, but PSNR values below 23 dB are representative of video quality that is extremely poor and difficult to view. Figure 6-2 is a comparison of source video to received video, as the number of clients is increased. 'YUVViewer.exe' was used to play raw YUV videos, downloaded as part of the EvalVid toolkit.

Source Video Quality

Video Quality with 5 Clients

Video Quality with 8 Clients

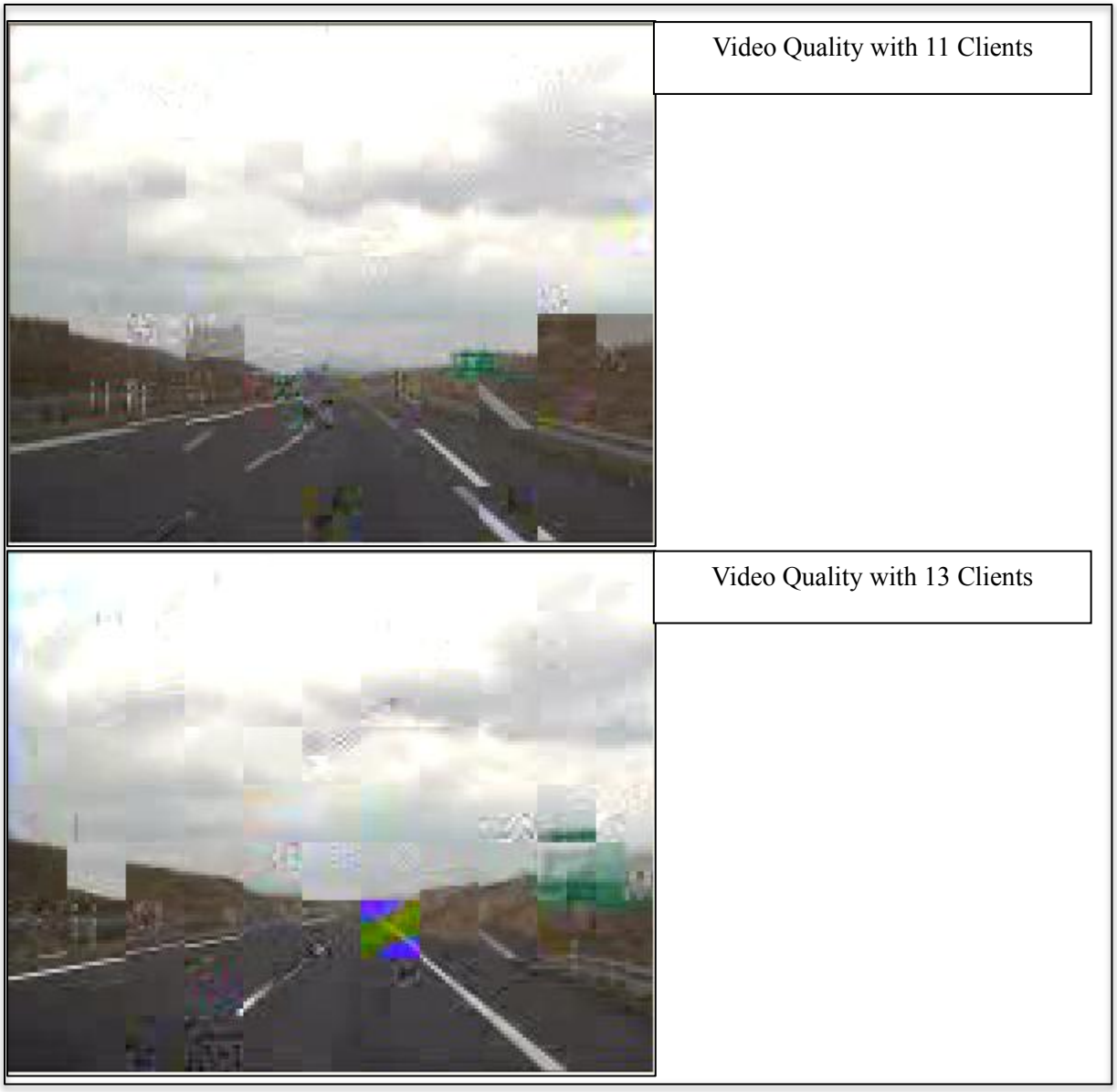

Figure 6-2 Centralized Video Quality Perception with Scaling

Figure 6-2 is an accurate representation of the average end-user's perception of received video as more clients come onto the network. Quality degradation can be perceived subjectively, and is validated by PSNR values from Figure 6-1. Subjective quality assessment, based on MOS, can rate video quality as degrading from excellent to very poor.

6.2 Quality of Decentralized Architecture

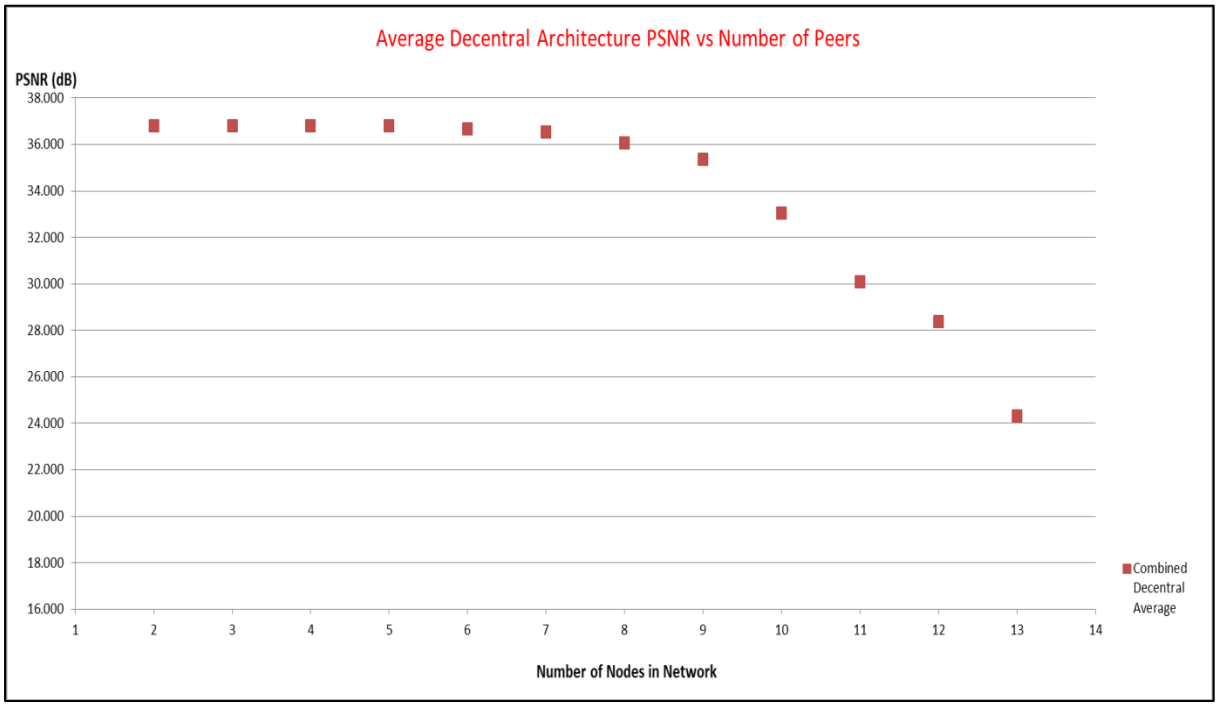

Figure 6-3 Network Averaged PSNR for Increasing Peers in Decentralized Architecture

The standard decentralized architecture maintains excellent video quality, by use of MOS rating, delivery as network load increases for an extended period. Maintenance of high video quality delivery for increasing loads is a result of adoption of the division of responsibility principle, in terms of storage and bandwidth, by the decentralized architecture. When the number of networked peers increases beyond ten, some video quality degradation is observed, with a slow degradation rate. Beyond 12 peers, video quality delivery of this system drops from good to fair, with a further increase resulting in poor quality. Figure 6-4 is a depiction of received video quality in the network for an increasing number of peers, as seen by the average end-user.

Source Video Quality

Video Quality with 5 requesting Peers

Video Quality with 8 requesting Peers

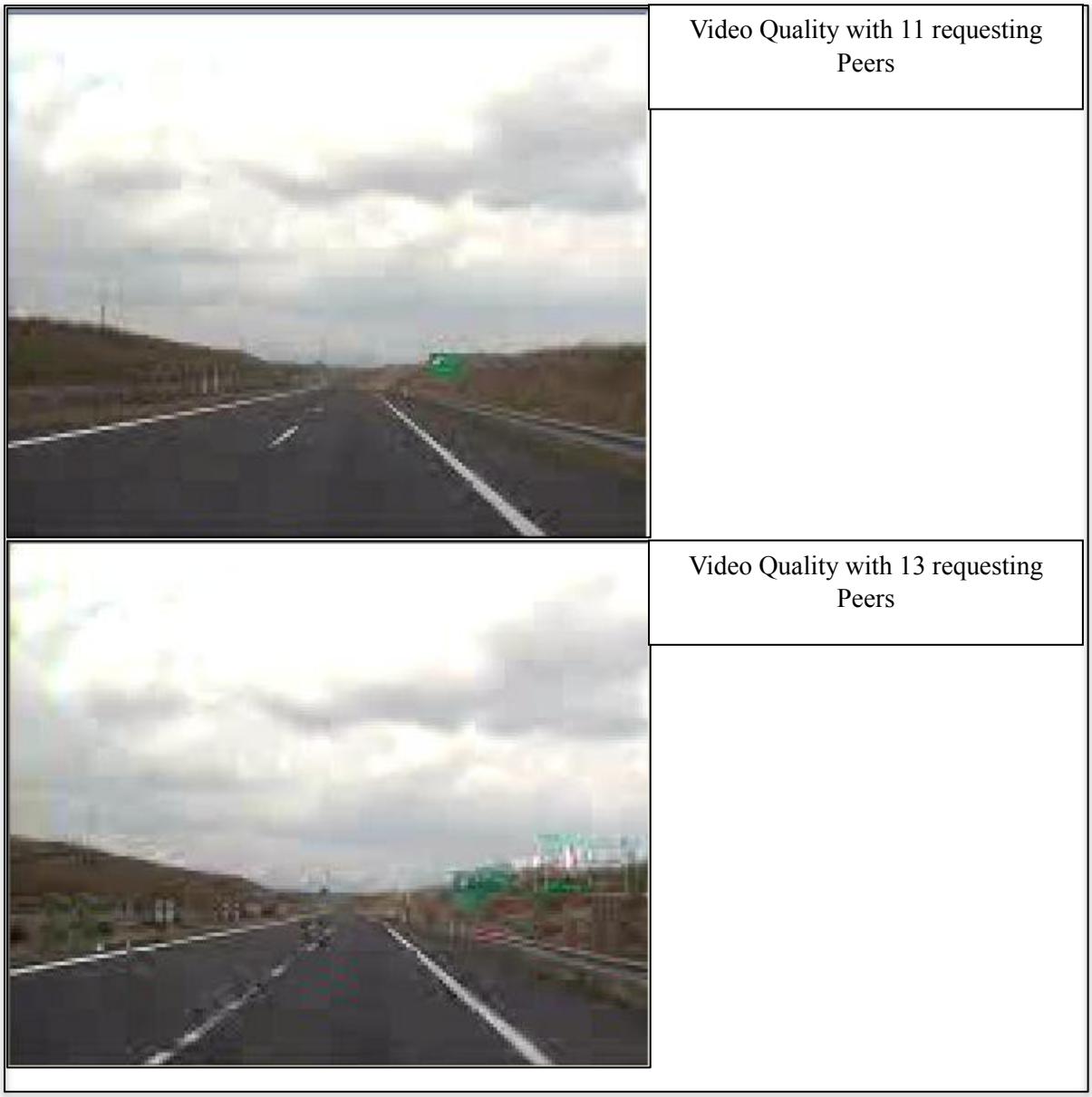

Figure 6-4 Decentral Video Quality Perception with Scaling

Observation of excellent quality video delivery is seen up to eight nodes, with a drop to good quality reception by eleven nodes. Validation of PSNR results is achieved by observation of the correlation between relative MOS interpretation of those PSNR values and a subjective assessment of received YUV videos, as depicted by Figure 6-4.

6.3 Comparison of Centralized and Decentralized Architectures in terms of Quality and Scalability

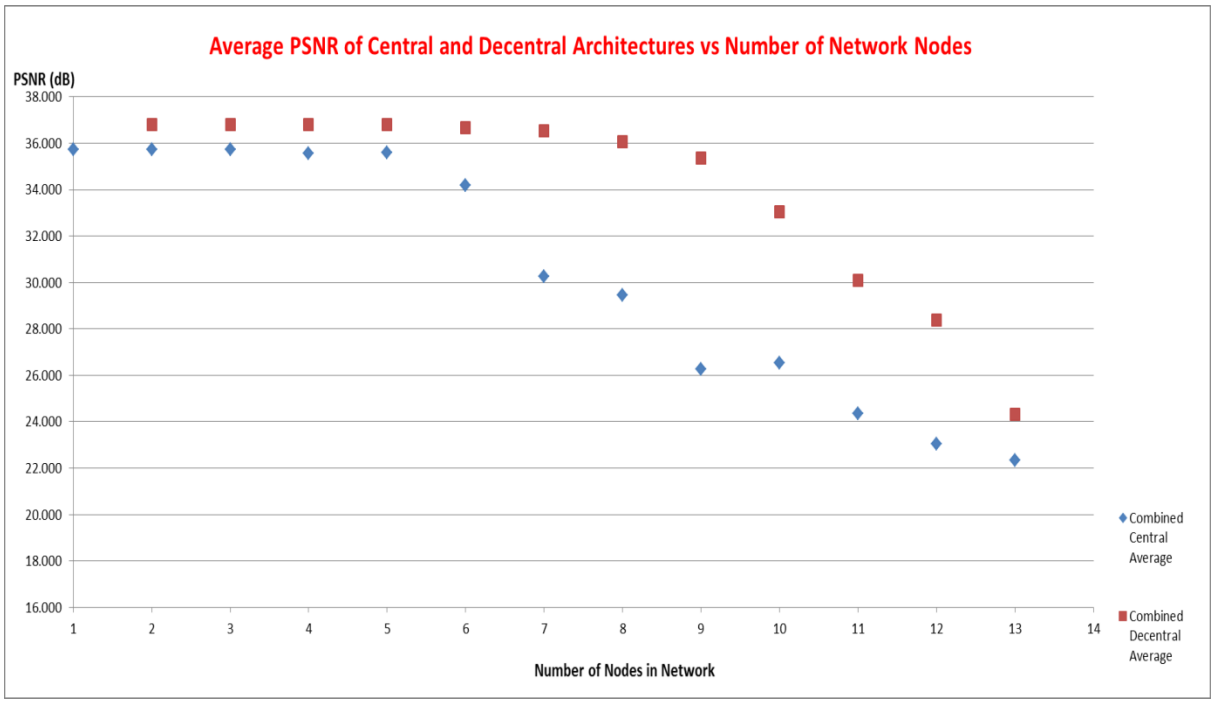

Figure 6-5 Direct Comparison of Centralized and Decentralized Network Averaged PSNR

Figure 6-5 is a comparison of average PSNR values of centralized (Figure 6-1) and decentralized (Figure 6-3) architectures for increasing nodes. Rates of quality degradation observed for both architectures indicate a steeper drop in video quality delivery of the centralized system when scaling up. When number of connected nodes is low, indicative of a low network load, both architectures are found to perform similarly. However, the decentralized architecture maintains excellent quality for an extended increase in number of nodes compared to the centralized architecture.

A higher PSNR value is indicative of higher quality video delivery. Drawing on this, Figure 6-6 is a representation of increase in quality of decentralized over centralized architecture.

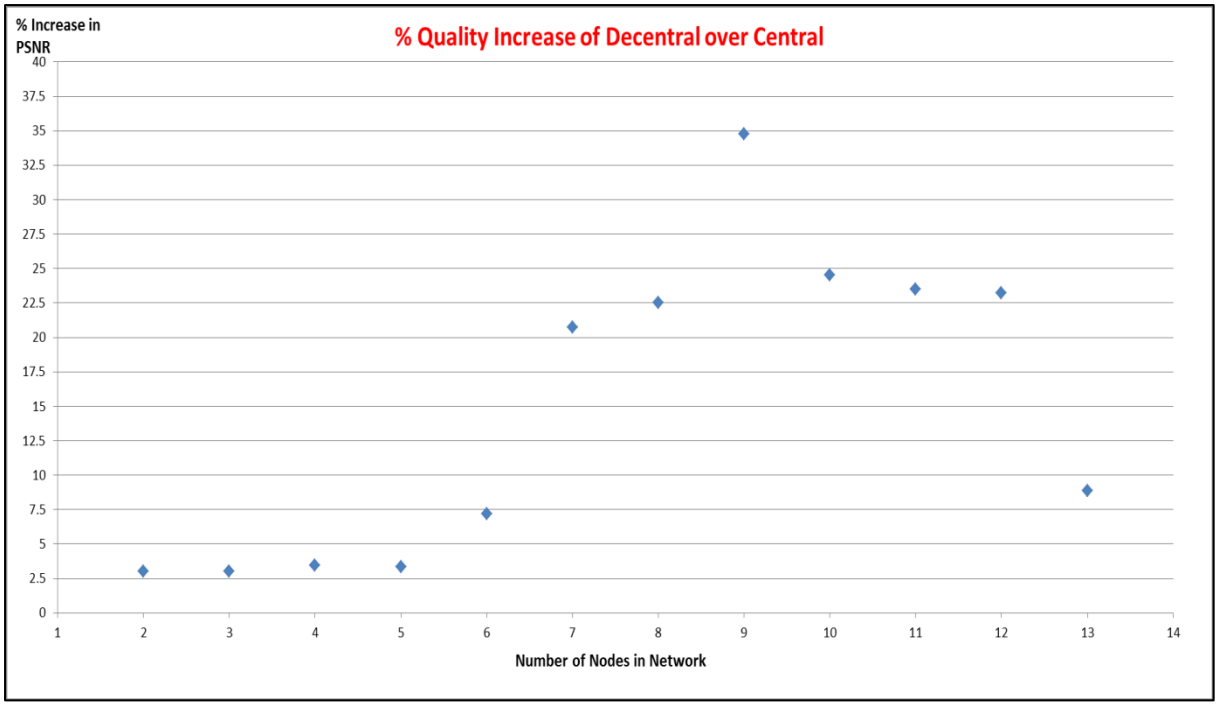

Figure 6-6 Delivered Video Quality Difference between Architectures

On average, 15 % higher video quality (PSNR) is observed in the decentralized architecture. As mentioned in Section 6.1, at nine nodes, the centralized architecture has an average video quality delivery of 26 dB, with the decentral average at 35.4 dB. Figure 6-6 reveals a peak quality increase of 34 % at this load.

An inference about scalability can be made using 26 dB as the tipping point, i.e. where end-users may be unhappy with received video quality. The QSQS, proposed in Section 5.5.3.3, suggests finding the point at which the decentralized architecture quality drops to 26 dB and finding the number of nodes served. Figure 6-5 shows that the decentralized architecture serves between 12 and 13 nodes at 26 dB. This value is compared to the centralized architecture serving nine nodes at this quality rating. It can thus be inferred that the decentralized architecture is around 35 % more scalable than the centralized architecture when serving fair (MOS rating) quality videos.

Increases in quality and scalability of the decentralized architecture decline with a further increase in workload. As the user base grows beyond 12 users, the quality improvement held decentrally drops off quickly, with both architectures exhibiting similar performance with 13 connected nodes. Video quality is thus hampered for both systems at very high loads. This is expected with the centralized architecture, as the server is overloaded. However, decentrally, addition of peers is indicative of an increase in network resources, thus rendering observed quality degradation for higher nodes unexpected. Reduction in quality of the decentralized architecture under very high loads can be explained, however, by the emergence of a ‘central nature’ in the decentralized system. This behaviour was discussed in Section 3 where the suggestion of a serving peer protocol was first

made. Significant network load increases result in demands on each peer to serve high volumes of data. However, peers do not have high resources, which are usually available to central servers, but are still expected to serve a considerable number of other peers. Thus, high loss rates are experienced in the decentralized architecture. Emergence of this behaviour in simulation further validates the integrity of data obtained from simulation in this dissertation.

6.4 Serving Peer Protocol Performance

Results of the method proposed in Section 5.5.4 are presented in this section. Suggestion of this protocol was due to the drop in quality experienced in the decentralized architecture under very high loads. Figure 6-7 and Figure 6-8 present results from evaluation, by the Quality Quantification System, of the serving peer protocol, under medium to high network loads.

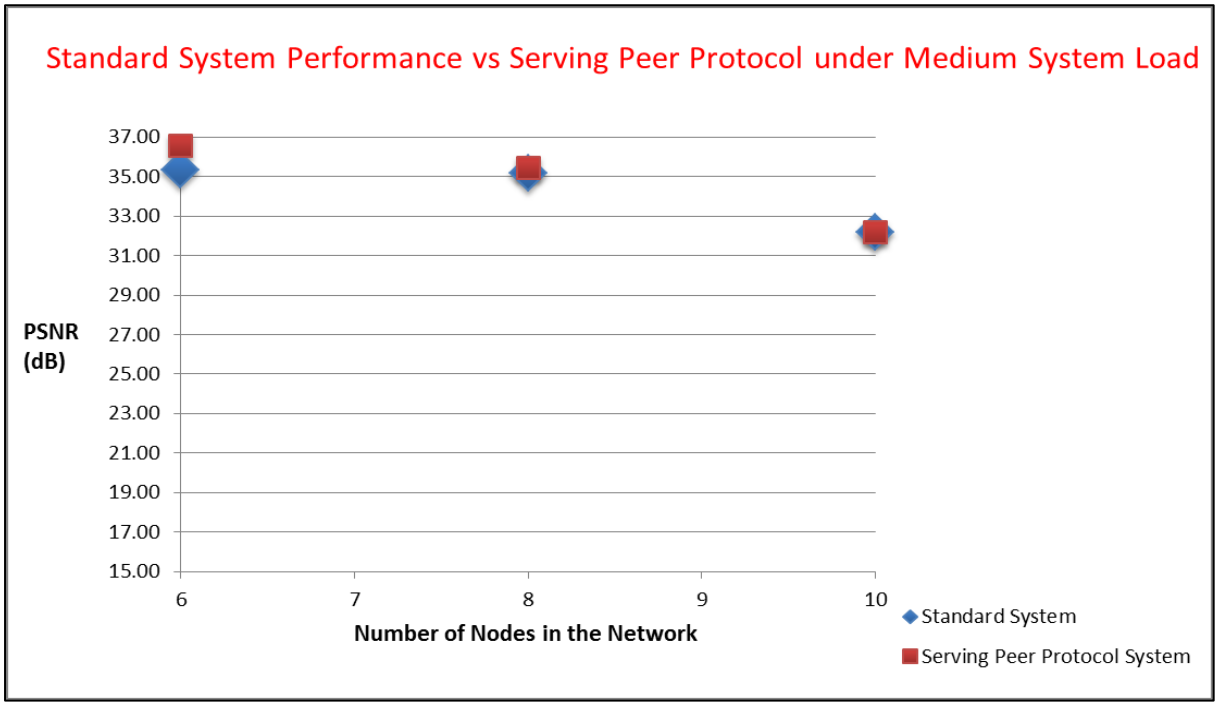

Figure 6-7 Effect of Serving Peer Protocol implementation at Medium System Load

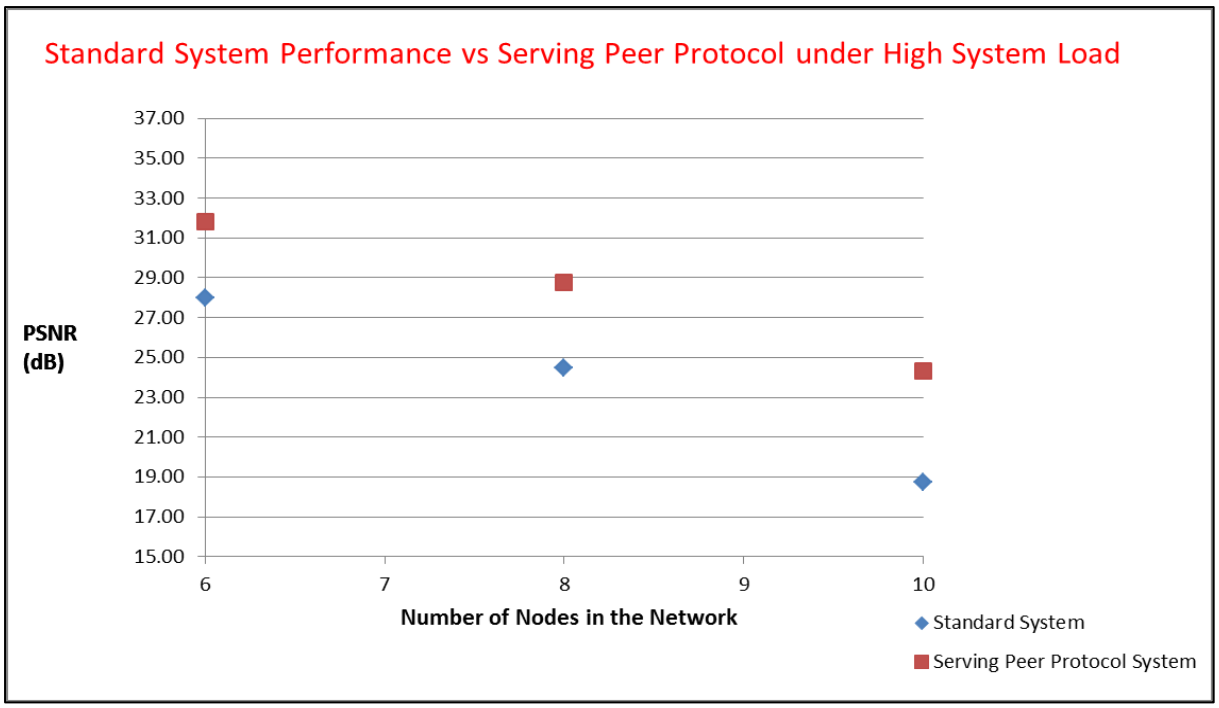

Figure 6-8 Effect of Serving Peer Protocol Implementation at High System Load

At medium system loads, Figure 6-7, effect of the serving peer protocol is less pronounced. This is in line with theory discussed in Section 3.11, as peers are able to serve contents when their uplink bandwidth is not saturated. Improvement in quality by means of the serving peer protocol is observed in Figure 6-8, where a high deficit bandwidth exists in the decentralized architecture amongst peers. Peers in the standard decentralized architecture with over demanded contents are unable to effectively serve requests, dropping packets as a result of saturated uplink bandwidth. Figure 6-8 shows the high rate of quality degradation in the standard network. Introduction of the serving peer protocol under high loads shows an average increase in video quality network-wide of about 30 % for a range of nodes.

Effectiveness of the serving peer protocol is highlighted by comparing Figure 6-7 and Figure 6-8. However, many P2P protocols exist for the purpose of improving streaming capabilities and quality in P2P systems, with many suggested in Chapter 3. The importance of evaluation of the serving peer protocol in this section is exhibition of the QSQS's capabilities, primarily the Quality Quantification System in this case with protocol evaluation. The evaluation system's to quantifiably evaluate new protocols in terms of quality or scalability, including full scale media streaming architectures, has been validated with the series of simulations performed and results obtained in this section.

Chapter 7 Conclusion

Knowledge of media streaming network architectures was gained from research into a vast and growing active field. A survey of the vast body of research revealed many areas of interest. Quick expansion of P2P media streaming systems was one such area.

Ability of P2P systems to scale better than client-server systems have made this an active area of research. Each of the three primary streaming media delivery architectures, Client-Server, CDNs and P2P, were analysed with a weighing of the benefits and drawbacks of each architecture. This propositioned the proposal of a hybrid media streaming architecture, which sought to gain from the advantages of each architecture, and purge associated disadvantages. In light of this, a P2P VoD STB Media Streaming architecture was proposed. Feeding off a CDN run by a media streaming organization, this system replaces a central server with a decentralized network of STBs controlled by a media organization. STBs allow for control and restriction of user behaviour, thereby alleviating many of the reliability issues associated with P2P architectures. A full scale system is specified, with integration of multiple P2P protocols that address buffering, video fragmentation and replication, amongst others. Development of this system exhibits the understanding of media streaming architectures gained through research, and assists in furthering understanding of methods of evaluation.

The proposal of a media streaming architecture was evaluated by the use of NS2, which is widely used amongst researchers in the field of media streaming. A comparison between NS2 and other network simulators is made to determine the best choice of toolset for achieving the main objective of this dissertation, which is a comparison of scalability between centralized and decentralized network architectures. Network metrics, such as rejection probability and packet loss rate, that proved common in evaluation of many media streaming systems researched, was deemed to not give an accurate view of end-user perception of video quality. Further research of network evaluation methods led to the discovery of an EvalVid toolset for video quality evaluation, utilising NS2 as a network simulation platform. PSNR, a video quality metric, was utilized to perform analysis of delivered video.

A QSQS was proposed using a modified EvalVid toolset and NS2 to evaluate media streaming networks by means of PSNR applied network wide. Development of this system is discussed in detail. Research found that decentralized architectures are acknowledged as having a superior ability to scale than centralized architectures. Application of the QSQS to basic centralized and decentralized architectures found that on average, for medium to high network loads, the decentralized architecture delivers 15 % higher video quality to peers than a centralized architecture with the same storage and bandwidth constraints. It was inferred, by the proposed methodology of the Scalability Quantification system, that the decentralized architecture was about 35 % more scalable than the centralized architecture when delivered video quality was matched.

A serving peer protocol was designed to negate the effects of a 'central' nature discovered in the decentralized architecture during research and subsequently observed during simulation, when very high loads were placed on the network. Evaluation of this protocol, which was proposed as part of the P2P VoD STB Media Streaming system, was performed by utilising the Quality Quantification system, a subsection of the complete evaluation system. A video quality increase of almost 30 % was observed network wide under very high loads, when this protocol was implemented in the decentralized architecture. Important constituents of the proposed P2P VoD STB Media Streaming System were tested, including the serving peer protocol.

Successful numerical evaluation of basic centralized and decentralized architectures, including a useful P2P protocol, by the QSQS highlights the potential of this system in formalising new experiential quality testing methods for media streaming systems. Focussing on developments of other works in the field of media streaming, it is possible to envisage this measurement system being employed as a performance indicator. Development of this system allowed successful determination of scalability and quality differences between centralized and decentralized media streaming architectures, allowing the primary objective of this dissertation to be achieved.

References

[1] D. Kerr, “Video Streaming is on the rise with Netflix dominating,” 14 May 2013. [Online]. Available: http://news.cnet.com/8301-1023_3-57584535-93/video-streaming-is-on-the-rise-with-netflix-dominating/. [Accessed 19 August 2013].

[2] D. Marpe, H. H. Inst., T. Wiegand and G. Sullivan, “The H.264/MPEG4 advanced video coding standard and its applications,” *IEEE Communications Magazine,* vol. 44, no. 8, pp. 134 - 143, August 2006.

[3] E. C. Laura Ricci, “Distributed Virtual Environments: From Client Server to Cloud and P2P Architectures,” in *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, Madrid, 2012.

[4] P. A. C. K. Sripanidkulchai, “Distributing Streaming Media Content Using Cooperative Networking,” in *Proceedings of the 12th international workshop on Network and*, 2002.

[5] G. X. Charilaos Stais, “Realistic Media Streaming over BitTorrent,” in *Future Network and MobileSummit 2012 Conference Proceedings*, 2012.

[6] J. E. Berkes, “Decentralized Peer-to-Peer Network Architecture: Gnutella and Freenet,” University of Manitoba, Manitoba, 2003.

[7] P. R. ,. L. M. Nikolaos Laoutaris, “ECHOS: Edge Capacity Hosting Overlays of Nano Data Centers,” *ACM SIGCOMM Computer Communication Review,* 2008.

[8] M. S. M. L. Nafaa A., “Analysis of a large-scale VOD architecture for broadband operators: a P2P-based solution,” *IEEE Communications Magazine,* vol. 46, no. 12, pp. 47-55, December 2008.

[9] D. C. Kyoungwon Suh, J. Kurose and L. Massoulie, “Push-to-Peer Video-on-Demand System: Design and Evaluation,” *Selected Areas in Communications, IEEE Journal on ,* vol. 25, no. 9, pp. 1706-1716, 2007.

[10] K. S. J. K. D. T. Y Guo, “A peer-to-peer on-demand streaming service and its performance evaluation,” in *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on (Volume:2)*, 2003.

[11] A. H. D. X. B. B. B. B. Mohamed Hefeeda, “CollectCast: A peer-to-peer service for media streaming,” *Multimedia Systems,* 01 11 2005.

[12] K. S. J. K. a. D. T. Yang Guo, “P2Cast: Peer-to-peer Patching Scheme for VoD Service,” in *WWW '03 Proceedings of the 12th international conference on World Wide Web*, New York, 2003.

[13] S. T. E. L. Juan Pedro Muñoz-Gea, “Modeling and Evaluation of Multisource Streaming Strategies in P2P VoD Systems,” *Consumer Electronics, IEEE Transactions on ,* vol. 58, no. 4, pp. 1202-1210, November 2012.

[14] Z. W. L. L. Su Yang, “Design of video transmission and quality evaluation system based on NS-2,” in *Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on*, XianNing, 2011.

[15] C.-K. S. W.-S. H. A. Z. Chih-Heng Ke, “An Evaluation Framework for More Realistic Simulations of MPEG Video Transmission,” *Journal of Information Science and Engineering,* vol. 24, no. 2, pp. 425-440, 2008.

[16] O. M. Maurizio Agelli, “MediaDART: A Decentralized Framework for Sharing Multimedia Content,” in *Sixth International Conference on Creating, Connecting and Collaborating through Computing (c5 2008)*, 2008.

[17] S. L. ,. R. B. ,. B. B. ,. A. S. Suman Banerjee, “Scalable resilient media streaming,” in *Proc. of NOSSDAV*, 2004.

[18] Z. Peng, H. Ting-lei, L. Cai-xia and X. Wang, “Research of P2P architecture based on cloud computing,” in *Intelligent Computing and Integrated Systems (ICISS), 2010 International Conference on*, Guilin.

[19] R. Y. ,. S. M. Sachin Yadav, “Comparison between Centralized & Decentralized Overlay Networks for Media Streaming,” *International Journal of Computer Applications,* vol. 5, no. 5, August 2010.

[20] H. Deshpande, M. Bawa and H. Garcia-Molina, “Streaming Live Media over a Peer-to-Peer Network,” Stanford InfoLab Publication Server, 2008.

[21] S.-J. Kim and M. Choi, “A genetic algorithm for server location and storage allocation in multimedia-on-demand network,” in *Mobile Future and Symposium on Trends in Communications, 2003. SympoTIC '03. Joint First Workshop on*, 2003.

[22] K. Xiong and H. Perros, “Service Performance and Analysis in Cloud Computing,” in *Services - I, 2009 World Conference on* , 2009.

[23] H. Khazaei, J. Misic and V. Misic, “Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems,” *Parallel and Distributed Systems, IEEE Transactions on ,* vol. 23, no. 5, May 2012.

[24] S. Chellouche, D. Negru, Y. Chen and M. Sidibe, “Home-Box-assisted content delivery network for Internet Video-on-Demand services,” in *Computers and Communications (ISCC), 2012 IEEE Symposium on*, Cappadocia, 2012.

[25] R. B. Alan Kin Wah Yim, “Decentralized media streaming infrastructure (DeMSI): An adaptive and high-performance peer-to-peer content delivery network,” *Journal of Systems Architecture,* vol. 52, no. 12, pp. 737-772, December 2006.

[26] I. Bermudez, M. Mellia and M. Meo, “Investigating Overlay Topologies and Dynamics of,” *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS,* vol. 29, no. 9, October 2011.

[27] V. Janardhan and H. Schulzrinne, “Peer assisted VoD for set-top box based IP network,” in *P2P-TV '07 Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV*, New York, 2007.

[28] Y. He and L. Guan, “Solving Streaming Capacity Problems in P2P VoD Systems,” *Circuits and Systems for Video Technology, IEEE Transactions on ,* vol. 20, no. 11, pp. 1638-1642, November 2010.

[29] K. Sato, K. Yagishita, M. Katsumoto and T. Miki, “Peer-to-Peer Based Mobile Video On-Demand with Layered Video Distribution,” in *Complex, Intelligent and Software Intensive Systems, 2007. CISIS 2007. First International Conference on* , 2007.

[30] J. Munoz-Gea, S. Traverso and E. Leonardi, “Modeling and Evaluation of Multisource Streaming Strategies in P2P VoD systems,” *IEEE Transactions on Consumer Electronics,* vol. 58, no. 4, pp. 1202-1210, November 2012.

[31] A. H. D. X. B. K. B. a. B. B. Mohamed Hefeeda, “CollectCast: A Peer-to-Peer service for media streaming,” *Multimedia Syst.,* vol. 11, 2005.

[32] H. X. Tao, Z. Hua and L. Z. Ding, “A Network Coding Solution on P2P Streaming Media On-demand System,” in *Web Information Systems and Mining, 2009. WISM 2009. International Conference on* , 2009.

[33] H. Schwarz, D. Marpe and T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard,” *Circuits and Systems for Video Technology, IEEE Transactions on ,* vol. 17, no. 9, pp. 1103-1120, September 2007.

[34] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems,” in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, 2001.

[35] P. Maymounkov and D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric,” in *IPTPS '01 Revised Papers from the First International Workshop on Peer-to-Peer Systems*.

[36] D. Yang, H. Wang, Y. Zhao and Y. Gao, “A Real-time Streaming Media File Sharing Mechanism Based on P2P and SIP,” in *Pervasive Computing and Applications, 2006 1st International Symposium on*, Urumqi, 2006.

[37] A. N. E. W. a. C. A. C. Chamil Jayasundara, “Localized P2P VoD Delivery Scheme with Pre-fetching for Broadband Access Networks,” in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Houston, 2011.

[38] N. A. M. P. P. S. Raghav Sethi, “Bounced — Improving data availability through replication in P2P networks,” in *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, Bangalore, 2013.

[39] R. E. K. M. H. S. Takaya Fujimoto, “Video-Popularity-Based Caching Scheme for P2P Video-on-Demand Streaming,” in *Advanced Information Networking and Applications (AINA), 2011 IEEE*

International Conference on, Singapore, 2011.

[40] W. Wu and J. C. Lui, "Exploring the Optimal Replication Strategy in P2P-VoD Systems: Characterization and Evaluation," *Parallel and Distributed Systems, IEEE Transactions on ,* vol. 23, no. 8, 06 December 2011.

[41] F. Boronat, M. Montagud and V. Vidal, "Enhanced RTP-based Tool-Set for Video Streaming Simulation using NS-2," in *MoMM '10 Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*, New York, 2010.

[42] H. Schulzrinne, G. Fokus, S. Casner and R. Frederick, "IETF RFC Search," January 1996. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1889.txt. [Accessed 02 09 2013].

[43] Z. Q. L. Y. Zheng Chaomei, "A streaming media congestion control mechanism based on UDP," in *Networking and Digital Society (ICNDS), 2010 2nd International Conference on* , 2010.

[44] D. Rubenstein, J. Kurose and D. Towsley, "Detecting shared congestion of flows via end-to-end measurement," *Networking, IEEE/ACM Transactions on ,* vol. 10, no. 3, pp. 381-395, June 2002.

[45] F. Liu, B. Li, B. Li and H. Jin, "Peer-Assisted On-Demand Streaming: Characterizing Demands and Optimizing Supplies," *Computers, IEEE Transactions on ,* vol. 62, no. 2, pp. 351-361, February 2013.

[46] T. Ge and S. Manoharan, "Mitigating Free-Riding on BitTorrent Networks," in *Digital Telecommunications (ICDT), 2010 Fifth International Conference on*, Athens, 2010.

[47] Organization, VideoLAN, "VLC Media Player," 2013. [Online]. Available: http://www.videolan.org/vlc/index.html. [Accessed 21 August 2013].

[48] P. S. a. M. R. F. H. P. Fitzek, "Using Network Simulators with Video Traces," Acticom Mobile Networks, 2003.

[49] T. d. Souza-Daw, T. D. Nguyen, T. M. Hoang and L. H. Ngoc, "Integration of an open-source network simulator (NS) and a video encoder-decoder (JM) for network video streaming in an educational platform," in *Global Engineering Education Conference (EDUCON), 2012 IEEE*, Marrakech, 2012.

[50] "PlanetLab," 6 March 2013. [Online]. Available: http://en.wikipedia.org/wiki/PlanetLab. [Accessed 21 August 2013].

[51] W. Lab, "NS2 or OMNet ++," WiCIP, 10 August 2012. [Online]. Available: http://www.wicip.ca/index.php/wicip-resources/82-wicip-main/wicip-resource-articles/74-wicip-ns2-or-omnetpp. [Accessed 2013 August 2013].

[52] T. Issariyakul and E. Hossain, Introduction to Network Simulator NS2, Springer, 2009.

[53] M. Greis, "Tutorial for the Network Simulator 'ns'," [Online]. Available: http://www.isi.edu/nsnam/ns/tutorial/index.html. [Accessed 21 August 2013].

[54] Pradeepkumar, "Creating a New Agent in Network Simulator 2," 12 December 2011. [Online]. Available: http://nsnam.com/2011/12/creating-new-agent-in-network-simulator.html. [Accessed 21 August 2013].

[55] V. Carrascal, “NS-2 code developed by Víctor Carrascal,” 10 March 2009. [Online]. Available: http://gridnet.upc.es/~maguilar/ns2_code_victor/ns2codevictor. [Accessed 21 August 2013].

[56] FFmpeg, “About FFmpeg,” [Online]. Available: http://www.ffmpeg.org/about.html. [Accessed 02 September 2013].

[57] H. Schulzrinne, P. Pan, A. Tsukamoto, D. Sisalem and S. Casner., “RTP Tools (Version 1.20),” 10 09 2013. [Online]. Available: http://www.cs.columbia.edu/irt/software/rtptools/. [Accessed 20 November 2013].

[58] Trace Education, “YUV Video Sequences,” 2013. [Online]. Available: http://trace.eas.asu.edu/yuv/. [Accessed 21 August 2013].

[59] Wikipedia, “Peak signal-to-noise ratio,” 22 August 2013. [Online]. Available: http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio. [Accessed 22 August 2013].

[60] Q. Huynh-Thu and M. Ghanbari, “Scope of validity of PSNR in image/video quality assessment,” *Electronic Letters,* vol. 44, no. 13, pp. 800-801, 27 June 2008.

[61] J. Klaue, B. Rathke and A. Wolisz, “EvalVid - A Framework for Video Transmission and Quality Evaluation,” in *Proc. of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Illinois.

[62] MPEG, “MPEG Decoder,” [Online]. Available: http://megaera.ee.nctu.edu.tw/mpeg/. [Accessed 2013 January 20].

[63] “Xvid codec,” Xvid, 2013. [Online]. Available: http://www.xvid.org/Project-Info.46.0.html. [Accessed 15 November 2013].

[64] Wikipedia, “Mean opinion score,” 25 June 2013. [Online]. Available: http://en.wikipedia.org/wiki/Mean_opinion_score.

[65] C.-H. Ke, “EvalVid in NS2,” [Online]. Available: http://140.116.164.80/~smallko/ns2/Evalvid_in_NS2.htm. [Accessed 22 August 2013].

[66] Cygwin, “Cygwin,” 2013. [Online]. Available: http://www.cygwin.com/. [Accessed 22 August 2013].

[67] “Poisson Distribution,” [Online]. Available: http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm.

[68] D. Kerr, “Video streaming is on the rise with Netflix dominating,” May 2013. [Online]. Available: http://news.cnet.com/8301-1023_3-57584535-93/video-streaming-is-on-the-rise-with-netflix-dominating/.

[69] C.-K. S. W.-S. H. a. A. Z. Chih-Heng Ke, “An Evaluation Framework for More Realistic Simulations of MPEG Video Transmission,” no. 24, 2008.

[70] K. S. J. K. a. D. T. Yang Guo, “P2Cast: Peer-to-Peer Patching Scheme for VoD Service,” in *WWW '03 Proceedings of the 12th international conference on World Wide Web*, New York, 2003.

[71] K. S. J. K. D. T. Yang Guo, “A peer-to-peer on-demand streaming service and its performance evaluation,” in *ICME '03 Proceedings. 2003 International Conference on Multimedia and Expo*,

2003.

[72] M. B. H. G.-M. H Deshpande, “Streaming Live Media over a Peer-to-Peer Network,” Stanford InfoLab Publication Server, 2008.

[73] K. Xiong and H. Perros, “Service Performance and Analysis in Cloud Computing,” in *2009 World Conference on Services - I*, 2009.

[74] H. Khazaei, J. Misic and V. Misic, “Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems,” *IEEE Transactions on Parallel and Distributed Systems,* vol. 23, no. 5, May 2012.

[75] “The Network Simulator - ns-2,” [Online]. Available: http://www.isi.edu/nsnam/ns/. [Accessed 27 June 2013].

[76] [Online]. Available: http://www.item.ntnu.no/~arnelie/Evalvid-RA.htm. [Accessed 27 June 2013].

[77] L. G. Yifeng He, “Solving Streaming Capacity Problems in P2P VoD Systems,” *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 20, no. 11, 2010.

[78] “YUV Video Sequences,” [Online]. Available: http://trace.eas.asu.edu/yuv/. [Accessed 10 June 2013].

Appendix A

All programs that were discussed in the dissertation are attached in this Appendix.

1. **'Agentfile.cc'**

```
#include "agent.h"
class TSPAgent : public Agent {
public:
TSPAgent();
protected:
int command(int argc, const char*const* argv);
private:
int    tsp_var1;
double tsp_var2;
void   TSPPrivFunc(void);
};

static class TSPAgentClass : public TclClass {
public:
TSPAgentClass() : TclClass("Agent/TSPAgentOtcl") {}
TclObject* create(int, const char*const*) {
return(new TSPAgent());
}
} class_tsp_agent;

TSPAgent::TSPAgent() : Agent(PT_UDP) {
bind("tsp_var1_otcl", &tsp_var1);
bind("tsp_var2_otcl", &tsp_var2);
}
int TSPAgent::command(int argc, const char*const* argv) {
if(argc == 2) {
if(strcmp(argv[1], "call-tsp-priv-func") == 0) {
TSPPrivFunc();
return(TCL_OK);
}
}
return(Agent::command(argc, argv));
}

void TSPAgent::TSPPrivFunc(void) {
Tcl& tcl = Tcl::instance();
tcl.eval("puts \"Message From TSPPrivFunc\"");
tcl.evalf("puts \"     Area of the Cylinder is = %f\"",
tsp_var1*tsp_var2*2*3.14);
}
```

2. **'VideoTrace.tcl'**

```
set ns [new Simulator]
$ns color 1 Blue
set nf [open out.nam w]
$ns namtrace-all $nf

Agent/UDP set nam_tracevar_ true
Agent/UDP set tracevar_ true

set send_node [$ns node]
set router_node_1 [$ns node]
set router_node_2 [$ns node]
set recv_node [$ns node]

$ns duplex-link $send_node $router_node_1 1Mb 10ms DropTail
$ns duplex-link $router_node_1 $router_node_2 1Mb 10ms DropTail
$ns duplex-link $router_node_2 $recv_node 1Mb 10ms DropTail

$ns duplex-link-op $router_node_1 $router_node_2 queuePos 0.5
$ns duplex-link-op $router_node_2 $recv_node queuePos 0.5
$ns queue-limit $router_node_1 $router_node_2 10
$ns queue-limit $router_node_2 $recv_node 10

set udp [new Agent/UDP]
$udp set fid_ 1
$ns attach-agent $send_node $udp

set snk [new Agent/Null]
$snk set fid_ 1
$ns attach-agent $recv_node $snk

$ns connect $udp $snk

set original_file_name Verbose_Jurassic_64.dat
set trace_file_name video.dat
set original_file_id [open $original_file_name r]

set trace_file_id [open $trace_file_name w]
set last_time 0

while {[eof $original_file_id] == 0} {

  gets $original_file_id current_line
  if {[string length $current_line] == 0 ||[string compare [string index
  $current_line 0] "#"] == 0} {
  continue
  }

  scan $current_line "%d%s%d" next_time type length
  set time [expr 1000*($next_time-$last_time)]
  set last_time $next_time
  puts -nonewline $trace_file_id [binary format "II" $time $length]
  }
  close $original_file_id
  close $trace_file_id

  set end_sim_time [expr 1.0*$last_time/1000+0.001]
```

```
set trace_file [new Tracefile]
$trace_file filename $trace_file_name
set video [new Application/Traffic/Trace]
$video attach-agent $udp
$video attach-tracefile $trace_file

# start the simulation:
$ns at 0.0 {
$send_node label "VIDEO-SERVER"
$router_node_1 label "IP-ROUTER 1"
$router_node_2 label "IP-ROUTER 2"
$recv_node label "VIDEO-CLIENT"
$video start
}

# stop the simulation:
$ns at $end_sim_time {
      finish
}

proc finish {} {
      global ns nf
      $ns flush-trace
      close $nf
      exec nam out.nam &
      exit 0
      }

$ns run
```

3. 'RTPTrace.tcl'

```
set ns [new Simulator]
set end_sim_time 10.0

set nf [open final.nam w]
set tf [open final.tr w]
$ns namtrace-all $nf
$ns trace-all $tf

proc finish {} {
      global ns nf tf
      $ns flush-trace
      close $nf
      close $tf

      #Execute nam on the trace file
      exec nam final.nam &

      exit 0

}

set node_(s1) [$ns node]
set node_(s2) [$ns node]
set node_(r1) [$ns node]

$ns duplex-link $node_(s1) $node_(r1) 10Mb 5ms DropTail
$ns duplex-link $node_(s2) $node_(r1) 10Mb 5ms DropTail

set trace_file [new Tracefile]
$trace_file filename starwars.nsformat
set RTP_s [new Agent/RTP_v2]
set RTCP_r [new Agent/RTCP_v2]
set self [new Session/RTP]

$ns attach-agent $node_(s1) $RTP_s
$ns attach-agent $node_(s2) $RTCP_r
$ns connect $RTP_s $RTCP_r
set video [new Application/Traffic/Trace]
$video attach-tracefile $trace_file
$video attach-agent $RTP_s

$RTCP_r session $self
$RTP_s session $self
$RTCP_r set interval_ 100ms
$RTCP_r set seqno_ 0
$RTP_s set packetSize_ 1064
$ns at 0.0 "$video start"
$ns at 0.0 "$RTCP_r start"

$ns at $end_sim_time {
            $video stop
            $RTCP_r stop
            $RTP_s stop

      finish

}

$ns run
```

4. **'newagent.cc'**

```
#include <stdio.h>
#include <iostream>
#include <string.h>
#include <cstdlib>
#include <cmath>
#include <ctime>
#include "agent.h"
#include <tcl.h>
#include <fstream>

using namespace std;

class TSPAgentRandom:public Agent {
public:
      TSPAgentRandom();
protected:
      int command(int argc, const char*const* argv);
private:
      double mean;
      double lambda;
      int result;
      void TSPPrivFuncRandom(void);
};

static class TSPAgentRandomClass : public TclClass {
public:
      TSPAgentRandomClass() : TclClass("Agent/TSPAgentOtcl"){}
      TclObject* create(int, const char*const*){
            return(new TSPAgentRandom());
      }
} class_tsp_agent;

TSPAgentRandom::TSPAgentRandom() : Agent(PT_UDP) {
      bind("mean_otcl",&mean);
      bind("lambda_otcl",&lambda);
      bind("result_otcl",&result);
}

int TSPAgentRandom::command(int argc, const char*const* argv)
      {
      if (argc == 2){
            if (strcmp(argv[1],"call-random-poisson")==0)
                  {
                        TSPPrivFuncRandom();
                        return(TCL_OK);
                  }
            }
            return(Agent::command(argc,argv));
      }

void TSPAgentRandom::TSPPrivFuncRandom()
      {
            Tcl& tcl = Tcl::instance();

    int seed = time(NULL);
            srand(seed);

        float R;
            R= (float)(rand()*-1)/(float)(RAND_MAX+1);
```

```
            //cout << R << endl;
            int k=0;
            const int max_k = 1000;
            double P = exp(-lambda);
            //cout << P << endl;
            double sum = P;
            //cout << sum << endl;
            if (sum>=R) result=0;
            for (k=1; k<max_k; ++k) {
                  //cout << P << endl;
                  P*=lambda/(double)(k);
                  sum+=P;
                  if (sum>=R) break;
            }
    result=k;

            tcl.eval("puts \"Message from Random Poisson Generator\"");
            tcl.evalf("puts \" Poisson Number = %d\"",result);

            ofstream myfile;
            myfile.open ("example.txt");
            myfile << result;
            myfile.close();

}
```

5. **'Central.tcl'**

```
set ns [new Simulator]

set nd [open out.tr w]
$ns trace-all $nd

set nf [open final.nam w]
$ns namtrace-all $nf

set numnodes 10
set simcount 10
set total_req 0

set fo [open poisson.txt w]
set sent_files [open sent.txt w]
set received_files [open received.txt w]

#Generate Poisson array
for {set count 0} {$count < $simcount} {incr count} {

      #Create MyAgent
      set myagent [new Agent/TSPAgentOtcl]

      #Set configurable parameters of MyAgent
      $myagent set mean_otcl 5
      $myagent set lambda_otcl $numnodes

      #Give a command to my agent
      $myagent call-random-poisson
      #Read from C++ output for random number
      set f [open example.txt]

      while {[eof $f] == 0} {
    gets $f current_line
    scan $current_line "%d" no_
      }

      set poisson_array($count) $no_
      set total_req [expr $total_req+$no_]
      puts -nonewline $fo $no_
      puts $fo "\n"
      after 500
      close $f
}

close $fo

puts $total_req
#Display elements for Poisson request array
for {set h 0} {$h < $simcount} {incr h} {
      puts $poisson_array($h)
}

set max_fragmented_size   1000
#add udp header(8 bytes) and IP header (20bytes)
set packetSize    1028
```

```
#Central server
set c0 [$ns node]
#Router nodes
set r1 [$ns node]
set r2 [$ns node]

for {set i 0} {$i < $numnodes} {incr i} {
      set d($i) [$ns node]
}

$ns duplex-link $c0 $r1 1Gb 1ms DropTail
$ns duplex-link $r1 $r2 100Mb 10ms DropTail

for {set i 0} {$i < $numnodes} {incr i} {
 $ns duplex-link $r2 $d($i) 100Mb 10ms DropTail
 }

set qr1r2 [[$ns link $r1 $r2] queue]
$qr1r2 set limit_ 1000

set original_file_name st
      set trace_file_name video1.dat
      set original_file_id [open $original_file_name r]
      set trace_file_id [open $trace_file_name w]


      set frame_count 0
      set last_time 0

      while {[eof $original_file_id] == 0} {
      gets $original_file_id current_line

      scan $current_line "%d%s%d%s%s%s%d%s" no_ frametype_ length_ tmp1_
tmp2_ tmp3_ tmp4_ tmp5_
      #puts "$no_ $frametype_ $length_ $tmp1_ $tmp2_ $tmp3_ $tmp4_ $tmp5_"

      # 30 frames/sec. if one want to generate 25 frames/sec, one can use
set time [expr 1000*1000/25]
      set time [expr 1000 * 1000/30]

      if { $frametype_ == "I" } {
            set type_v 1
      }

      if { $frametype_ == "P" } {
            set type_v 2
      }

      if { $frametype_ == "B" } {
            set type_v 3
      }

      if { $frametype_ == "H" } {
            set type_v 1
      }
```

```
      puts  $trace_file_id "$time $length_ $type_v $max_fragmented_size"
      incr frame_count
      }

close $original_file_id
close $trace_file_id
set end_sim_time [expr 1.0 * 1000/30 * ($frame_count + 1)  / 1000]
puts "$end_sim_time"

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req < $numnodes} {
            puts "yes"
            for {set j 0} {$j < $num_req} {incr j} {
                  set udp($j,$num_sim) [new Agent/myUDP]
                  $ns attach-agent $c0 $udp($j,$num_sim)
                  $udp($j,$num_sim) set packetSize_ $packetSize
                  $udp($j,$num_sim) set_filename sd($j)be_$num_sim
                  set name sd
                  append name $j "_" $num_sim
                  puts $sent_files $name
                  file rename -force sd($j)be_$num_sim $name
                  set null($j,$num_sim) [new Agent/myUdpSink2]
                  $ns attach-agent $d($j) $null($j,$num_sim)
                  $ns connect $udp($j,$num_sim) $null($j,$num_sim)
                  $null($j,$num_sim) set_trace_filename rd($j)be_$num_sim
                  set name rd
                  append name $j "_" $num_sim
                  puts $received_files $name
                  file rename -force rd($j)be_$num_sim $name
            }
       for {set k 0} {$k < $num_req} {incr k} {
                  set trace_file($k,$num_sim) [new Tracefile]
                  $trace_file($k,$num_sim) filename $trace_file_name
                  set video($k,$num_sim) [new Application/Traffic/myTrace2]
                  $video($k,$num_sim) attach-agent $udp($k,$num_sim)
                  $video($k,$num_sim)                        attach-tracefile
$trace_file($k,$num_sim)
        }
       for {set a 0} {$a < $num_req} {incr a} {$ns at [expr 60.0*$num_sim]
"$video($a,$num_sim) start"}
       for {set b 0} {$b < $num_req} {incr b} {$ns at [expr 60.0*$num_sim +
$end_sim_time] "$video($b,$num_sim) stop"}
       for {set c 0} {$c < $num_req} {incr c} {$ns at [expr 60.0*$num_sim +
$end_sim_time + 1.0] "$null($c,$num_sim) closefile"}
      } elseif {$num_req >= $numnodes} {
            puts "no"
            for {set j 0} {$j < $num_req} {incr j} {
                  set udp($j,$num_sim) [new Agent/myUDP]
                  $ns attach-agent $c0 $udp($j,$num_sim)
                  $udp($j,$num_sim) set packetSize_ $packetSize
                  $udp($j,$num_sim) set_filename sd($j)be_$num_sim
                  set name sd
                  append name $j "_" $num_sim
                  puts $sent_files $name
                  file rename -force sd($j)be_$num_sim $name
                  set null($j,$num_sim) [new Agent/myUdpSink2]
```

```
                if {$j < $numnodes} {
                      $ns attach-agent $d($j) $null($j,$num_sim)
                } elseif {$j >= $numnodes} {
                      set z [expr $j-$numnodes]
                      $ns attach-agent $d($z) $null($j,$num_sim)
                }
                $ns connect $udp($j,$num_sim) $null($j,$num_sim)
                $null($j,$num_sim) set_trace_filename rd($j)be_$num_sim
                set name rd
                append name $j "_" $num_sim
                puts $received_files $name
                file rename -force rd($j)be_$num_sim $name
           }
       for {set k 0} {$k < $num_req} {incr k} {
                set trace_file($k,$num_sim) [new Tracefile]
                $trace_file($k,$num_sim) filename $trace_file_name
                set video($k,$num_sim) [new Application/Traffic/myTrace2]
                $video($k,$num_sim) attach-agent $udp($k,$num_sim)
                $video($k,$num_sim)                      attach-tracefile
$trace_file($k,$num_sim)
        }
       for {set a 0} {$a < $num_req} {incr a} {
       if { $a < $numnodes} {
                $ns at [expr 60.0*$num_sim] "$video($a,$num_sim) start"
       } elseif {$a >= $numnodes} {
                $ns at [expr 60.0*$num_sim + 60.0] "$video($a,$num_sim)
start"
       }
       }
       for {set b 0} {$b < $num_req} {incr b} {
       if { $b < $numnodes} {
                $ns    at    [expr    60.0*$num_sim    +    $end_sim_time]
"$video($b,$num_sim) stop"
       } elseif {$b >= $numnodes} {
                $ns    at    [expr    60.0*$num_sim    +    $end_sim_time*2]
"$video($b,$num_sim) stop"
       }
       }
       for {set c 0} {$c < $num_req} {incr c} {
       if { $c < $numnodes} {
                $ns  at  [expr  60.0*$num_sim  +  $end_sim_time  +  1.0]
"$null($c,$num_sim) closefile"
       } elseif {$c >= $numnodes} {
                $ns  at  [expr  60.0*$num_sim  +  $end_sim_time*2  +  2.0]
"$null($c,$num_sim) closefile"
       }
       }
      }
}

close $sent_files
close $received_files

proc finish {} {
        global ns nd nf
        $ns flush-trace
        close $nd
```

```
        close $nf
        #exec nam final.nam &
        exit 0
}

$ns at [expr $end_sim_time*$total_req + 3.0] "finish"

$ns run
```

6. ‘Decentral.tcl’

```
set ns [new Simulator]

set nd [open out.tr w]
$ns trace-all $nd

set nf [open final.nam w]
$ns namtrace-all $nf

set numnodes 10
set simcount 10
set total_req 0

set sent_files [open sent.txt w]
set received_files [open received.txt w]

#Generate Poisson array
for {set count 0} {$count < $simcount} {incr count} {

      #Create MyAgent
      set myagent [new Agent/TSPAgentOtcl]

      #Set configurable parameters of MyAgent
      $myagent set mean_otcl 5
      $myagent set lambda_otcl $numnodes

      #Give a command to my agent
      $myagent call-random-poisson
      #Read from C++ output for random number
      set f [open example.txt]

      while {[eof $f] == 0} {
    gets $f current_line
    scan $current_line "%d" no_
      }

      set poisson_array($count) $no_
      set total_req [expr $total_req+$no_]
      after 500
      close $f
}
puts $total_req
#Display elements for Poisson request array
for {set h 0} {$h < $simcount} {incr h} {
      puts $poisson_array($h)
}

set max_fragmented_size   1000
#add udp header(8 bytes) and IP header (20bytes)
set packetSize    1028

set rc [$ns node]

for {set i 0} {$i < $numnodes} {incr i} {
      set d($i) [$ns node]
}
```

```
for {set i 0} {$i < $numnodes} {incr i} {
      set r($i) [$ns node]
}

for {set j 0} {$j < $numnodes} {incr j} {
      $ns duplex-link $r($j) $rc 100Mb 10ms DropTail
}

for {set k 0} {$k < $numnodes} {incr k} {
      set qrcr($k) [[$ns link $r($k) $rc] queue]
      $qrcr($k) set limit_ 1000
}

for {set a 0} {$a < $numnodes} {incr a} {
      $ns duplex-link $d($a) $r($a) 100Mb 10ms DropTail
}

for {set b 0} {$b < $numnodes} {incr b} {
      set original_file_name($b) st$b
      set trace_file_name($b) video($b).dat
      set original_file_id($b) [open $original_file_name($b) r]
      set trace_file_id($b) [open $trace_file_name($b) w]
      set frame_count($b) 0
      set last_time($b) 0

      while {[eof $original_file_id($b)] == 0} {
      gets $original_file_id($b) current_line

      scan $current_line "%d%s%d%s%s%s%d%s" no_ frametype_ length_ tmp1_
tmp2_ tmp3_ tmp4_ tmp5_
      #puts "$no_ $frametype_ $length_ $tmp1_ $tmp2_ $tmp3_ $tmp4_ $tmp5_"

      # 30 frames/sec. if one want to generate 25 frames/sec, one can use
set time [expr 1000*1000/25]
      set time [expr 1000 * 1000/30]

      if { $frametype_ == "I" } {
            set type_v 1
      }

      if { $frametype_ == "P" } {
            set type_v 2
      }

      if { $frametype_ == "B" } {
            set type_v 3
      }

      if { $frametype_ == "H" } {
            set type_v 1
      }

      puts          $trace_file_id($b)     "$time     $length_     $type_v
$max_fragmented_size"
      incr frame_count($b)
      }
```

```
close $original_file_id($b)
close $trace_file_id($b)
set end_sim_time($b) [expr 1.0 * 1000/30 * ($frame_count($b) + 1)  / 1000]
puts "$end_sim_time($b)"
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req < $numnodes } {
            puts "yes"
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        set udp($k,$j,$num_sim) [new Agent/myUDP]
                        $ns attach-agent $d($k) $udp($k,$j,$num_sim)
                        $udp($k,$j,$num_sim) set packetSize_ $packetSize
                        $udp($k,$j,$num_sim)                    set_filename
sd($k)be_($j)be_$num_sim
                        set name sd
                        append name $k "_" $j "_" $num_sim
                        puts $sent_files $name
                        file rename -force sd($k)be_($j)be_$num_sim $name
                        set  null($k,$j,$num_sim)  [new  Agent/myUdpSink2]

                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if { $k == $j } {
                              for {set z 0} { $z < $numnodes } {incr z} {
                                    if {$z == $j} {
                                          continue
                                    }
                              $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
    }
    }
    for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns          connect          $udp($k,$j,$num_sim)
$null($k,$j,$num_sim)
                        $null($k,$j,$num_sim)            set_trace_filename
rd($k)be_($j)be_$num_sim
                        set name rd
                        append name $k "_" $j "_" $num_sim
                        puts $received_files $name
                        file rename -force rd($k)be_($j)be_$num_sim $name
                        set trace_file($k,$j,$num_sim) [new Tracefile]
                        $trace_file($k,$j,$num_sim)               filename
$trace_file_name($k)
                        set           video($k,$j,$num_sim)           [new
Application/Traffic/myTrace2]
                        $video($k,$j,$num_sim)                  attach-agent
$udp($k,$j,$num_sim)
```

```
                        $video($k,$j,$num_sim)              attach-tracefile
$trace_file($k,$j,$num_sim)
             }
            }
            }
            } elseif { $num_req > $numnodes } {
                  puts "no"
                  for {set j 0} {$j < $num_req} {incr j} {
                  for {set k 0} {$k < $numnodes} {incr k} {
                  set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                              set udp($k,$j,$num_sim) [new Agent/myUDP]
                              $ns attach-agent $d($k) $udp($k,$j,$num_sim)
                              $udp($k,$j,$num_sim)      set      packetSize_
$packetSize
                              $udp($k,$j,$num_sim)              set_filename
sd($k)be_($j)be_$num_sim
                              set name sd
                              append name $k "_" $j "_" $num_sim
                              puts $sent_files $name
                              file  rename  -force  sd($k)be_($j)be_$num_sim
$name
                              set         null($k,$j,$num_sim)          [new
Agent/myUdpSink2]
                        } elseif { $j >= $numnodes } {
                                set udp($k,$j,$num_sim) [new Agent/myUDP]
                                set aa [expr $j-$numnodes]
                                    $ns         attach-agent         $d($aa)
$udp($k,$j,$num_sim)
                                    $udp($k,$j,$num_sim)   set   packetSize_
$packetSize
                                    $udp($k,$j,$num_sim)        set_filename
sd($k)be_($j)be_$num_sim
                                    set name sd
                                    append name $k "_" $j "_" $num_sim
                                    puts $sent_files $name
                                    file            rename            -force
sd($k)be_($j)be_$num_sim $name
                                    set      null($k,$j,$num_sim)      [new
Agent/myUdpSink2]
                        }
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k==$j || $k==$tempvar} {
                        if { $j < $numnodes } {
                              for {set z 0} { $z < $numnodes } {incr z} {
                                    if {$z == $j} {
                                          continue
                                    }
                              $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
```

```
        if { $j >= $numnodes } {
            set aa [expr $j-$numnodes]
            for {set z 0} { $z < $numnodes } {incr z} {
                  if {$z == $aa} {
                        continue
                  }
                  $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
      }
    }
    }
    for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        $ns           connect           $udp($k,$j,$num_sim)
$null($k,$j,$num_sim)
                        $null($k,$j,$num_sim)             set_trace_filename
rd($k)be_($j)be_$num_sim
                        set name rd
                        append name $k "_" $j "_" $num_sim
                        puts $received_files $name
                        file rename -force rd($k)be_($j)be_$num_sim $name
                        set trace_file($k,$j,$num_sim) [new Tracefile]
                        $trace_file($k,$j,$num_sim)                 filename
$trace_file_name($k)
                        set            video($k,$j,$num_sim)            [new
Application/Traffic/myTrace2]
                        $video($k,$j,$num_sim)                  attach-agent
$udp($k,$j,$num_sim)
                        $video($k,$j,$num_sim)              attach-tracefile
$trace_file($k,$j,$num_sim)
             }
            }
            }
            }
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req < $numnodes } {
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns at [expr 6.0*$num_sim] "$video($k,$j,$num_sim)
start"
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns  at  [expr  6.0*$num_sim  +  $end_sim_time($k)]
"$video($k,$j,$num_sim) stop"
                  }
            }
```

```
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns  at  [expr  6.0*$num_sim  +  $end_sim_time($k)]
"$null($k,$j,$num_sim) closefile"
                  }
            }
            }
      }
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req >= $numnodes } {
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                                    $ns     at     [expr     6.0*$num_sim]
"$video($k,$j,$num_sim) start"
                        } elseif { $j >= $numnodes } {
                              $ns    at    [expr    6.0*$num_sim    +    6.0]
"$video($k,$j,$num_sim) start"
                        }
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                         $ns  at  [expr  6.0*$num_sim  +  $end_sim_time($k)]
"$video($k,$j,$num_sim) stop"
                  } elseif { $j >= $numnodes } {
                              $ns     at     [expr     6.0*$num_sim     +
$end_sim_time($k)*2] "$video($k,$j,$num_sim) stop"
                  }
             }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                         $ns at [expr 6.0*$num_sim + $end_sim_time($k) +
1.0] "$null($k,$j,$num_sim) closefile"
                  } elseif { $j >= $numnodes } {
                              $ns     at     [expr     6.0*$num_sim     +
$end_sim_time($k)*2 + 2.0] "$null($k,$j,$num_sim) closefile"
                  }
             }
            }
            }
```

```
      }
}

close $sent_files
close $received_files

proc finish {} {
        global ns nd nf
        $ns flush-trace
        close $nd
        close $nf
        exec nam final.nam &
        exit 0
}

$ns at [expr $end_sim_time(0)*$total_req + 1000.0] "finish"

$ns run
```

7. Bash Script for Central Architecture Batch Processing

```
#!/bin/bash

i=1;

FILE=sent.txt

echo "###################"

k=1

while read sdline;do

echo "Line # $k: $sdline"

a='r'

rdline=$a${sdline:1}

et.exe $sdline $rdline st highway_qcif.cmp err${sdline:2}.cmp 1

mpeg4decoder.exe err${sdline:2}.cmp err${sdline:2} 176 144 > df${sdline:2}

myfixyuv.exe df${sdline:2} qcif 2000 err${sdline:2}.yuv
myfix${sdline:2}.yuv

psnr.exe 176 144 420 highway_qcif.yuv myfix${sdline:2}.yuv >
psnr${sdline:2}

((k++))

done < $FILE

echo "Total number of lines in file:$k"
```

8. Bash Script for Decentral Architecture

```
#!/bin/bash
i=1;
FILE=sent.txt
echo "##################"
k=1
while read sdline;do
echo "Line # $k: $sdline"
aa='r'
rdline=$aa${sdline:1}
partnum=${sdline:2:1}
bb='highway'
cc='_qcif.cmp'
vidnum=$bb$partnum$cc
dd='st'
tracenum=$dd$partnum
et.exe $sdline $rdline $tracenum $vidnum err${sdline:2}.cmp 1
temp1='err'
temp2=${sdline:2}
temp3='.cmp'
errorname=$temp1$temp2$temp3
mpeg4decoder.exe $errorname err${sdline:2} 176 144 > df${sdline:2}
temp4='df'
dfname=$temp4$temp2
temp5='.yuv'
erroryuv=$temp1$temp2$temp5
myfixyuv.exe $dfname qcif 200 $erroryuv myfix${sdline:2}.yuv
temp6='myfix'
myfixname=$temp6$temp2$temp5
temp7='_qcif.yuv'
videoyuv=$bb$partnum$temp7
psnr.exe 176 144 420 $videoyuv $myfixname > psnr${sdline:2}
((k++))
done < $FILE
echo "Total number of lines in file:$k"
```

9. ‘DecentralWithServingPeer.tcl’

```
set ns [new Simulator]

set nd [open out.tr w]
$ns trace-all $nd

set nf [open final.nam w]
$ns namtrace-all $nf

set numnodes 6
set simcount 10
set total_req 0
set stime 10.0

set sent_files [open sent.txt w]
set received_files [open received.txt w]

#Generate Poisson array
for {set count 0} {$count < $simcount} {incr count} {

      #Create MyAgent
      set myagent [new Agent/TSPAgentOtcl]

      #Set configurable parameters of MyAgent
      $myagent set mean_otcl 5
      $myagent set lambda_otcl $numnodes

      #Give a command to my agent
      $myagent call-random-poisson
      #Read from C++ output for random number
      set f [open example.txt]

      while {[eof $f] == 0} {
    gets $f current_line
    scan $current_line "%d" no_
      }

      set poisson_array($count) $no_
      set total_req [expr $total_req+$no_]
      after 500
      close $f
}
puts $total_req
#Display elements for Poisson request array
for {set h 0} {$h < $simcount} {incr h} {
      puts $poisson_array($h)
}

set max_fragmented_size   1000
#add udp header(8 bytes) and IP header (20bytes)
set packetSize    1028

set rc [$ns node]

for {set i 0} {$i < $numnodes} {incr i} {
      set d($i) [$ns node]
```

```
}

for {set i 0} {$i < $numnodes} {incr i} {
      set r($i) [$ns node]
}

for {set j 0} {$j < $numnodes} {incr j} {
      $ns duplex-link $r($j) $rc 1Mb 10ms DropTail
}

for {set k 0} {$k < $numnodes} {incr k} {
      set qrcr($k) [[$ns link $r($k) $rc] queue]
      $qrcr($k) set limit_ 10000
}

for {set a 0} {$a < $numnodes} {incr a} {
      $ns duplex-link $d($a) $r($a) 1Mb 10ms DropTail
}

for {set b 0} {$b < $numnodes} {incr b} {
      set original_file_name($b) st$b
      set trace_file_name($b) video($b).dat
      set original_file_id($b) [open $original_file_name($b) r]
      set trace_file_id($b) [open $trace_file_name($b) w]
      set frame_count($b) 0
      set last_time($b) 0

      while {[eof $original_file_id($b)] == 0} {
      gets $original_file_id($b) current_line

      scan $current_line "%d%s%d%s%s%s%d%s" no_ frametype_ length_ tmp1_
tmp2_ tmp3_ tmp4_ tmp5_
      #puts "$no_ $frametype_ $length_ $tmp1_ $tmp2_ $tmp3_ $tmp4_ $tmp5_"

      # 30 frames/sec. if one want to generate 25 frames/sec, one can use
set time [expr 1000*1000/25]
      set time [expr 1000 * 1000/30]

      if { $frametype_ == "I" } {
            set type_v 1
      }

      if { $frametype_ == "P" } {
            set type_v 2
      }

      if { $frametype_ == "B" } {
            set type_v 3
      }

      if { $frametype_ == "H" } {
            set type_v 1
      }

      puts          $trace_file_id($b)      "$time     $length_      $type_v
$max_fragmented_size"
      incr frame_count($b)
```

```
      }
close $original_file_id($b)
close $trace_file_id($b)
set end_sim_time($b) [expr 1.0 * 1000/30 * ($frame_count($b) + 1)  / 1000]
puts "$end_sim_time($b)"
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req < $numnodes } {
            puts "yes"
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        set udp($k,$j,$num_sim) [new Agent/myUDP]
                        $ns attach-agent $d($k) $udp($k,$j,$num_sim)
                        $udp($k,$j,$num_sim) set packetSize_ $packetSize
                        $udp($k,$j,$num_sim)                    set_filename
sd($k)be_($j)be_$num_sim
                        set name sd
                        append name $k "_" $j "_" $num_sim
                        puts $sent_files $name
                        file rename -force sd($k)be_($j)be_$num_sim $name
                        set   null($k,$j,$num_sim)   [new   Agent/myUdpSink2]

                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if { $k == $j } {
                              for {set z 0} { $z < $numnodes } {incr z} {
                                    if {$z == $j} {
                                          continue
                                    }
                              $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
    }
    }
    for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns           connect           $udp($k,$j,$num_sim)
$null($k,$j,$num_sim)
                        $null($k,$j,$num_sim)             set_trace_filename
rd($k)be_($j)be_$num_sim
                        set name rd
                        append name $k "_" $j "_" $num_sim
                        puts $received_files $name
                        file rename -force rd($k)be_($j)be_$num_sim $name
                        set trace_file($k,$j,$num_sim) [new Tracefile]
                        $trace_file($k,$j,$num_sim)                filename
$trace_file_name($k)
                        set            video($k,$j,$num_sim)            [new
Application/Traffic/myTrace2]
```

```
                        $video($k,$j,$num_sim)                  attach-agent
$udp($k,$j,$num_sim)
                        $video($k,$j,$num_sim)              attach-tracefile
$trace_file($k,$j,$num_sim)
             }
            }
            }
            } elseif { $num_req > $numnodes } {
                  puts "no"
                  for {set j 0} {$j < $num_req} {incr j} {
                  for {set k 0} {$k < $numnodes} {incr k} {
                  set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                              set udp($k,$j,$num_sim) [new Agent/myUDP]
                              $ns attach-agent $d($k) $udp($k,$j,$num_sim)
                              $udp($k,$j,$num_sim)      set      packetSize_
$packetSize
                              $udp($k,$j,$num_sim)              set_filename
sd($k)be_($j)be_$num_sim
                              set name sd
                              append name $k "_" $j "_" $num_sim
                              puts $sent_files $name
                              file  rename  -force  sd($k)be_($j)be_$num_sim
$name
                              set          null($k,$j,$num_sim)          [new
Agent/myUdpSink2]
                        } elseif { $j >= $numnodes } {
                                set udp($k,$j,$num_sim) [new Agent/myUDP]
                                if { $j < [expr $numnodes*2] } {
                                set aa [expr $j-$numnodes]
                                } elseif { $j >= [expr $numnodes*2] } {
                                set aa [expr $j-$numnodes*2]
                                }
                                    $ns         attach-agent         $d($aa)
$udp($k,$j,$num_sim)
                                    $udp($k,$j,$num_sim)   set   packetSize_
$packetSize
                                    $udp($k,$j,$num_sim)        set_filename
sd($k)be_($j)be_$num_sim
                                    set name sd
                                    append name $k "_" $j "_" $num_sim
                                    puts $sent_files $name
                                    file            rename            -force
sd($k)be_($j)be_$num_sim $name
                                    set      null($k,$j,$num_sim)      [new
Agent/myUdpSink2]
                        }
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k==$j || $k==$tempvar} {
                        if { $j < $numnodes } {
                              for {set z 0} { $z < $numnodes } {incr z} {
```

```
                                    if {$z == $j} {
                                          continue
                                    }
                                $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
        if { $j >= $numnodes } {
            set aa [expr $j-$numnodes]
            for {set z 0} { $z < $numnodes } {incr z} {
                  if {$z == $aa} {
                        continue
                  }
                  $ns attach-agent $d($k) $null($z,$j,$num_sim)
            }
        }
      }
    }
    }
    for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        $ns           connect           $udp($k,$j,$num_sim)
$null($k,$j,$num_sim)
                        $null($k,$j,$num_sim)             set_trace_filename
rd($k)be_($j)be_$num_sim
                        set name rd
                        append name $k "_" $j "_" $num_sim
                        puts $received_files $name
                        file rename -force rd($k)be_($j)be_$num_sim $name
                        set trace_file($k,$j,$num_sim) [new Tracefile]
                        $trace_file($k,$j,$num_sim)                 filename
$trace_file_name($k)
                        set            video($k,$j,$num_sim)            [new
Application/Traffic/myTrace2]
                        $video($k,$j,$num_sim)                  attach-agent
$udp($k,$j,$num_sim)
                        $video($k,$j,$num_sim)              attach-tracefile
$trace_file($k,$j,$num_sim)
             }
            }
            }
            }
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req < $numnodes } {
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns         at         [expr         $stime*$num_sim]
"$video($k,$j,$num_sim) start"
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
```

```
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns at [expr $stime*$num_sim + $end_sim_time($k)]
"$video($k,$j,$num_sim) stop"
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
                  if {$k != $j} {
                        $ns at [expr $stime*$num_sim + $end_sim_time($k)]
"$null($k,$j,$num_sim) closefile"
                  }
            }
            }
      }
}

for {set num_sim 0} {$num_sim < $simcount} {incr num_sim} {
 set num_req $poisson_array($num_sim)
      if { $num_req >= $numnodes } {
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                                    $ns    at    [expr    $stime*$num_sim]
"$video($k,$j,$num_sim) start"
                        } elseif { $j >= $numnodes && $j < [expr
$numnodes*2] } {
                              $ns  at  [expr  $stime*$num_sim  +  $stime]
"$video($k,$j,$num_sim) start"
                        } elseif { $j >= [expr $numnodes*2] } {
                              $ns at [expr $stime*$num_sim + $stime*2.0]
"$video($k,$j,$num_sim) start"
                        }
                  }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                         $ns at [expr $stime*$num_sim + $end_sim_time($k)]
"$video($k,$j,$num_sim) stop"
                  } elseif { $j >= $numnodes && $j < [expr $numnodes*2] } {
                              $ns    at    [expr    $stime*$num_sim    +
$end_sim_time($k)*2] "$video($k,$j,$num_sim) stop"
                  } elseif { $j >= [expr $numnodes*2] } {
                              $ns    at    [expr    $stime*$num_sim    +
$end_sim_time($k)*3] "$video($k,$j,$num_sim) stop"
                  }
             }
            }
            }
            for {set j 0} {$j < $num_req} {incr j} {
```

```
            for {set k 0} {$k < $numnodes} {incr k} {
             set tempvar [expr $j-$numnodes]
                  if { $k!=$j && $k!=$tempvar} {
                        if { $j < $numnodes } {
                         $ns at [expr $stime*$num_sim + $end_sim_time($k) +
1.0] "$null($k,$j,$num_sim) closefile"
                  } elseif { $j >= $numnodes && $j < [expr $numnodes*2] } {
                              $ns     at     [expr     $stime*$num_sim     +
$end_sim_time($k)*2 + 2.0] "$null($k,$j,$num_sim) closefile"
                  } elseif { $j >= [expr $numnodes*2] } {
                          $ns      at      [expr      $stime*$num_sim      +
$end_sim_time($k)*3 + 2.0] "$null($k,$j,$num_sim) closefile"
                  }
             }
            }
            }
      }
}

close $sent_files
close $received_files

proc finish {} {
        global ns nd nf
        $ns flush-trace
        close $nd
        close $nf
        exec nam final.nam &
        exit 0
}

$ns at [expr $end_sim_time(0)*$total_req + 1000.0] "finish"

$ns run
```

Appendix B

Journal Paper submitted to IEEE Transactions on Circuits and Systems for Video Technology, currently under review.

Quantifying Scalability of Decentral Media Streaming Networks over Central Networks, including Empirical Evaluation of a Peer-to-Peer Protocol, by use of Peak Signal to Noise Ratio

W. Frederick, *Member, IEEE*, A.L.Leigh Jarvis

***Abstract*—The performance of central and decentral media streaming architectures is examined in this paper. Peak Signal to Noise Ratio (PSNR) is used to quantify the performance of both architectures, with quality and scalability used as performance criteria. Both central and decentral architectures are implemented with the same hardware characteristics in Network Simulator 2. Performance analysis of network results is achieved with modifications to the Evalvid framework. Experimental results indicate that the decentral system has on average 15 % higher PSNR than the central system and 34 % higher scalability under similar operating conditions. Decentral system drawbacks under very high loads are examined with possible solutions mentioned. Performance of the quality and scalability quantification system prompts testing of a serving peer protocol to improve decentral system performance under very high loads. Results show a 30 % increase in PSNR under high system loads.**

***Index Terms*—Central Architecture, Client-Server, Decentral Architecture, Peer-to-Peer, Performance Evaluation, Scalability.**

INTRODUCTION

MEDIA streaming has attracted much attention in recent years and is responsible for the highest traffic on the internet [68]. As a result of this expansion, several streaming technologies have since emerged that address the issue of serving clients. From an architectural perspective, these can be divided into two main categories, central and decentral networks.

The first and simplest solution was to use a central server which serves client requests directly. In this system, clients are served videos directly and with minimal delay. However, video data requires higher bandwidth as the quality of video and unique connections increase. This translates to the server having to serve large amounts of data to multiple clients simultaneously.

This research has been supported by Allied Technologies Limited (ALTECH) bursary grant.

W. Frederick and A.L.Leigh Jarvis are both based in the School of Engineering at the University of KwaZulu Natal, (e-mail: 208504648@stu.ukzn.ac.za and jarvis@ukzn.ac.za)

A.L.L Jarvis is with the Engineering Faculty of the University of KwaZulu Natal, KwaZulu Natal, South Africa (e-mail: jarvis@ukzn.ac.za).

For example, a typical Blu-Ray disc using the MPEG4 AVC video codec requires 18 Mbps for streaming a 1080p Full HD video. Maintaining a persistent 18 Mbps upstream rate for a video, with duration of about 2 hours, for a number of clients, is hardware intensive and economically inefficient for a modest client base. As the client base grows, the problem with centralized streaming networks emerges. Central servers become overloaded quickly with an increase in demand and tend to fall over when relative demand from users is high. Alternatives to central architecture developed as a result of the scalability issue with central servers.

Peer-to-Peer overlay networks on a central backbone network were the first to improve on the scalability of the client-server model. In these networks, the server uses the caching and streaming ability of its clients during times of high demand to reduce its workload. As a client is streaming from a server, it caches the part of the video it has already watched so that the next client that requests the same video can stream from the first client. In this way, the server can deal with an increased load and more clients. However, this approach was very

limited and still degraded, as a large demand was still place on the central server for videos uncommonly requested.

Server duplication in order to handle increasing user bases is an effective approach to dealing with the scalability problem of central systems. These networks are usually referred to as content delivery networks or data centers. Servers are duplicated across the user population and become known as "edge" servers. They all connect to a main central content server. As the user base grows, more duplication is needed and costs start to significantly increase. This approach is expensive but is in constant development.

A Peer-to-Peer (P2P) network is the least expensive but most unreliable approach to media streaming. In a pure P2P network, no server exists and the network uses the storage and streaming capabilities of network clients to allow streaming. This approach was borne out of decentralized file sharing systems such as Gnutella and BitTorrent. P2P Live and P2P Video on Demand are two P2P streaming systems under constant development. Since the mid 2000's, these approaches have been steadily replacing client-server systems and seem to be an improved solution for scalability and less prone to single point of failure problems. P2P systems are intrinsically capable of large scaling unlike client-server based systems as new users bring with them new hardware and thus higher storage and streaming bandwidth into the network. Although scalability in these systems may seem plausible, it does however have drawbacks. Most significantly, decentral architecture algorithms have to deal with a highly dynamic system where peer status constantly changes and there is minimal service guarantee to users.

Similar to the nature in which organizations such as Internet Service Providers (ISPs) have to provide a Quality of Service (QoS) guarantee to users, media streaming organizations need to provide the same QoS guarantees. In most of the works researched on these systems, performance is measured in terms of blocking probability i.e. whether a client request was served or not. This is an effective measure of network performance but it may not adequately address the Quality of Experience (QoE) of users in the network, i.e. how users perceive visual quality of streamed video. For example, a request may be served but the quality of video received might be very poor. We thus propose a new scheme in which comparisons can be made between networks on a QoE basis.

Based on Network Simulator 2 (NS2), the EvalVid framework for network video transmission evaluation, and by use of Peak Signal to Noise Ratio (PSNR), this paper aims at quantifying the scalability of a decentral network over a central network.

For the video quality measurement and evaluation systems, modifications were made to the work performed in Chih-Heng *et al* [69]. A pure client-server network and peer-to-peer network is developed for implementation in the quality measurement system.

Performance of the quality measurement system is tested further by the introduction of a serving peer protocol in the decentral system. Development of this protocol arose as a result of the degradation in performance of the pure peer-to-peer system under very high loads.

The remainder of this paper is organized as follows. Section 2 discusses related work including current media streaming systems and evaluation methods used in their development. Section 3 outlines the development of fair and comparable pure central and decentral media streaming networks. The quality measurement system is discussed in Section 4 highlighting the methodology in quantifying scalability of the systems. Section 5 discusses results obtained from the proposed system. A discussion on implementation of a serving peer protocol in the decentral architecture is also given with the subsequent performance evaluation of the protocol. Future improvements to the system and upcoming work are discussed, including a brief conclusion of the findings in this paper in Sections 6 and 7.

RELATED WORKS

Many techniques exist to address the scalability problem of Client-Server systems. Numerous types of systems have been developed which employ the peer-to-peer technique. P2Cast is one such system, which cooperatively streams video using patching techniques to construct an overlay network [70]. Simulations performed show that P2Cast is more scalable than typical client-server architectures through use of the client rejection probability metric. Average network load and server load are included as further performance metrics. These metrics show the performance improvement of their system over client server systems but do not give a good indication as to how this improvement in network performance translates into improved video quality seen by the end user.

DirectStream is another system which is evaluated with the overall rejection probability of user requests against server workload [71]. Although results show good scaling for the system, there is little indication of video reception quality at the end user as the user base increases. Evaluation of CollectCast is achieved through frame level performance analysis, performed by evaluating packet loss rate against time [31].

This is similar to the system we deploy in that frame level performance is observed. However, being a video delivery system, it is important to go further and look at how each frame PSNR drops as network load increases which is the goal of our system evaluation method.

SpreadIt is a system that uses client resources for streaming high bandwidth video. This architecture spreads bandwidth requirements across the network where maintenance of QoS is high priority [72]. Empirical methods are employed to evaluate video quality against the number of peers streaming. Performance analysis on this basis ties in with our system development as we propose a method that goes a step further and numerically evaluates video quality observed by all peers in the network.

The work in this paper stresses the importance of multiple performance criteria in assessment of network systems to ensure that users receive the best possible service while maintaining the minimal use of network resources. Xiong *et al* [73] and Khazaei *et al* [74] highlight the fact that in order to develop optimal systems, accurate modeling of a system is critical, as in the case with cloud architectures. Monitoring service performance in cloud computing has helped to develop better cloud architectures as these works have shown [73]. The same principle is adhered to in this paper.

In their work, Munoz-Gea *et al* [30] developed a mathematical model for evaluating Peer-to-Peer Video on Demand systems. Blocking probability (request rejection probability) is used as the main performance indicator. This model has proven very effective as the results from the model correlate with simulation. Our system method of quality evaluation can be integrated with this model to provide both QoS and QoE evaluation methods for new media streaming architectures to perform system optimization.

In our work, we have established a media streaming research system by use of the work developed in Chih-Heng *et al* [69] for video quality analysis in conjunction with NS2 [75] platform and Evalvid [76] framework. Hence, a method for analyzing PSNR of videos transmitted through a simulated network was designed. Modifications were made to the established system to allow network-wide PSNR analysis to determine the quality of received video across a complete user base.

Drawbacks of Peer-to-Peer systems under high loads are discussed in Yifeng *et al* [77]. Over-demanded content from a single peer reveals the lack of streaming capacity in a decentralized architecture. Content replication is a way of dealing with the issue of over demanded content as the same content is placed in more locations allowing for more access. In the event of a peer with saturated upload capacity, a 'helper' protocol is deployed. It employs another peer, with the same segments as the strained peer, to serve the requesting peer [77]. We finally evaluated the effect of a serving peer protocol, similar to the helper protocol [77], by use of our quality measurement system.

MEDIA STREAMING NETWORK ARCHITECTURE DESIGN

This paper aims to compare central and decentral media systems with a focus on scalability. NS2 was used as the simulation platform for developing the network architectures. Making a fair comparison between the two architectures, hardware criteria were established.

A generic central architecture is shown in Figure 1, with a fixed server storage capacity and arbitrary bandwidth links, in this case 1 Mbps for clients and 2 Mbps for the server, to a router. Speeds are dependent on the quality of video being viewed. The server stores a complete video for transmission to client nodes. Client requests are served as they come and the router employs a simple Droptail queuing protocol which operates on a First-In-First-Out basis.

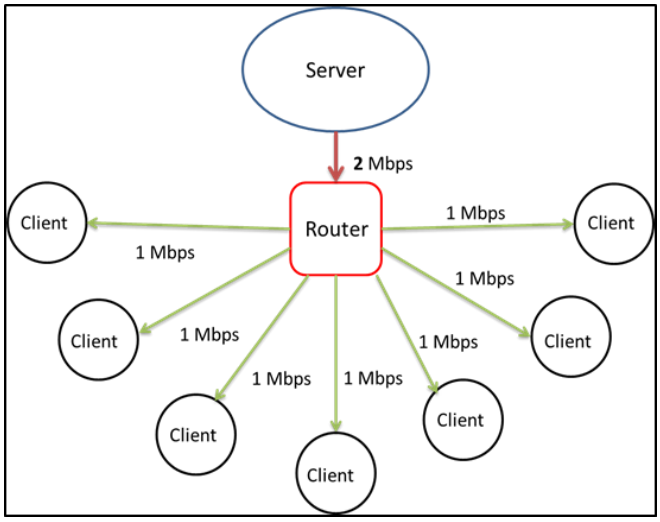

Fig. 9. Central System Architecture, with Arbitrary Bandwidth Links.

In Figure 2, all decentral peers all have the same duplex links to the router as clients of the central architecture. Storage capacity in this architecture is distributed equally through the peer nodes. Droptail queuing protocol is followed in this architecture. To maintain even distribution of responsibility, each peer request is served equally by every other peer in the network.

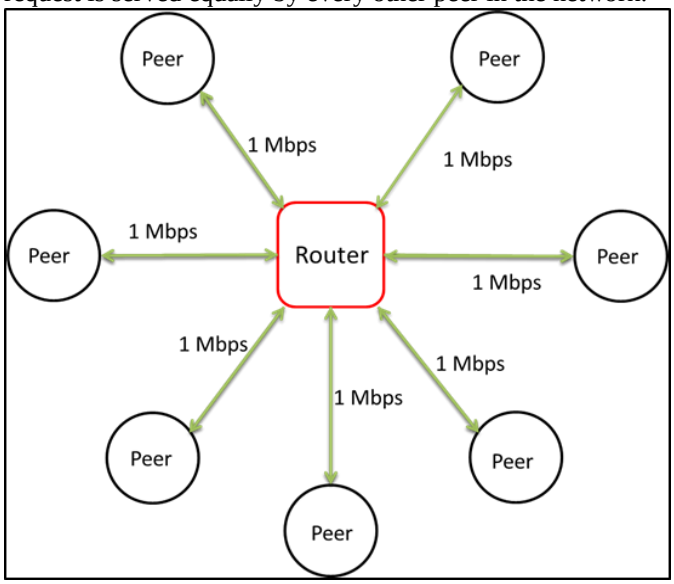

Fig. 10. Decentral System Architecture

Full duplex 1 Mbps Ethernet standards are used for both systems in simulations. Finally, a standard test video, from YUV Sequences [78], 2000 frames long, was chosen and stored in each node decentrally, i.e. divided by the number of nodes in the system.

SYSTEM DEVELOPMENT AND METHODOLOGY FOR QUANTIFYING THE SCALABILITY OF EACH ARCHITECTURE

System Development

A request model had to be developed to ensure that network loads would be realistic during simulations. It was decided that a Poisson Random Request Generator would be developed in order to serve requests. A typical Poisson distribution has a probability mass function heavily weighted towards the mean value, or lambda.

Thus, in a system with *n* nodes, setting the mean of the Poisson generator equal to the number of nodes in the network, i.e. $\lambda=n$, will ensure that on average, n requests are generated each simulation period. Logically, this means that each user requests one video every simulation period, but this number may increase or decrease dependent on the output of the request generator. Intuitively, requests generated are proportional to the number of nodes in the network. This allows both network architectures to have fair request arrival policies which generate variable loads.

Work performed by Chih-Heng *et al* [69] is used as the basis for developing the quality evaluation system, briefly discussed below. This evaluation system was modified to enable batch PSNR processing of video files to determine the average PSNR network wide. Figure 3, adapted from [69], depicts the flow of system operations.

The RAW YUV sequence refers to the 2000 frame video, taken from standard YUV sequences [78], used in this comparative analysis, although any YUV video can be used. The important modification to the system proposed in Chih-Heng et al [69], is the recording and averaging of PSNR network wide. PSNR values are stored for each receiving node as the simulation is run. Data is then analysed to find where in the system quality drops and at which point the highest loss rate and thus poorest video quality can be found. However, it was decided to average video quality results across the network as this method can gauge video quality observed by all users at any time in the network. PSNR is widely accepted as the best objective video quality metric [69]. PSNR is calculated using the following formula:

$$PSNR(n)_{dB} = 20log_{10}\frac{V_{peak}}{\sqrt{\frac{1}{N_{col}N_{row}}\sum_{i=0}^{N_{col}}\sum_{j=0}^{N_{row}}[Y_S(n,i,j)-Y_D(n,i,j)]^2}}, \quad (1)$$

$V_{peak} = 2^k - 1$, where k represents the number of bits per pixel (luminance component).

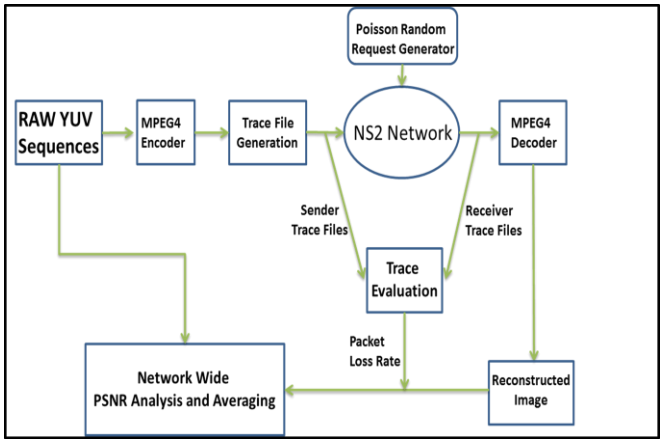

Fig. 3. Simulation System Batch Processing Structure adapted from Chih-Heng *et al* [69]

The denominator of the logarithmic argument represents the Mean Square Error (MSE) between the received frame, Y_d, and the sent frame, Y_s. A higher PSNR generally implies higher received video quality. An important scale that uses the PSNR metric is the Mean Opinion Square (MOS), which is a subjective representation of PSNR values. Table 1 depicts MOS and PSNR mapping as judged by critics [69].

TABLE I
MOS AND PSNR MAPPING

PSNR(dB)	MOS
>37	**5(Excellent)**
31~37	**4(Good)**
25-31	**3(Fair)**
20~25	**2(Poor)**
<20	**1(Bad)**

Mean Opinion Square is important as it links human perception to PSNR values. Our work requires that PSNR values obtained from simulations be interpreted in order to make judgments about the user's network experience.

Methodology for Quantification of Scalability

From the system described above, it is possible to accurately measure the scalability of decentral architectures over central architectures using video quality as the metric. The method can be described step-by-step as follows:

a) A simulation runs ten times where ten discrete random Poisson requests are generated. Each request generation occurs at a time equal to less than the complete time it takes to serve the full video in the case of the central system, and less than the time it takes to serve each piece of the video in the case of the decentral system. This intrinsically increases the system load as it attempts to mimic the unpredictable behavior of users in the network. Ten is chosen as an arbitrary number but it must be noted that the higher the simulation number, the finer the results that are obtained and a better representation of network behavior is obtained.

b) Run the simulation for the central system starting with two clients. Once the average PSNR for the network is obtained, run the simulation again with three clients. Continue to increase the number of clients in the network with each simulation until the PSNR value drops below a certain arbitrary value. This value was chosen to be 23 dB as the MOS suggests that video quality would be poor at this point.

c) Apply the same principle as described in Step (b) to the decentral system. For each simulation run, divide the video frames evenly across network peers. This ensures that division of responsibility is applied fairly and total decentral storage capacity does not exceed that of the central system. Run simulations from two peers to the same number of peers reached by the central system and record the average network PSNR.

d) Repeat Steps (b) and (c) at least twice to average out results and obtain the highest possible accuracy for measurement in both systems.

e) An arbitrary PSNR value of 28 dB is chosen as the MOS suggests this to be the threshold between good and fair quality. Using this PSNR value, determine the number of users served in each network. The difference between these numbers would be a good reflection of the difference in scalability between the central and decentral architectures.

RESULTS

The method described in Section 4.2 was followed and video quality results obtained were processed. Systems described in Section 3 were used in simulations performed and quality and scalability analyses were performed.

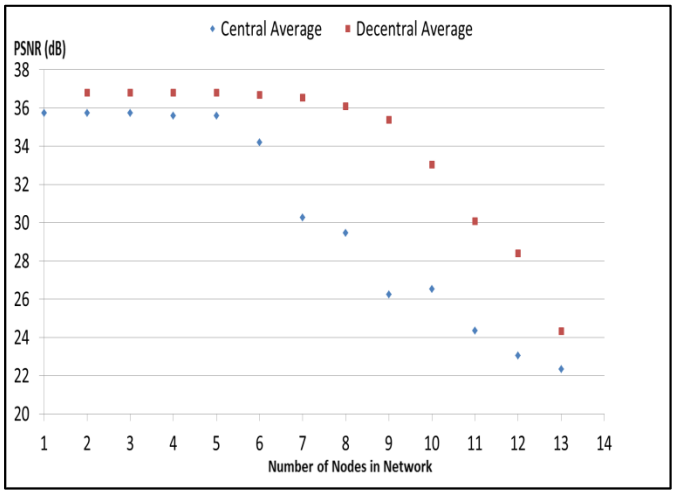

Fig. 4. Average PSNR of Central [blue marker] and Decentral [red marker] Architectures vs. Number of Network Nodes. Degradation of video quality through a decrease of PSNR is observed with an increasing user base in both architectures.

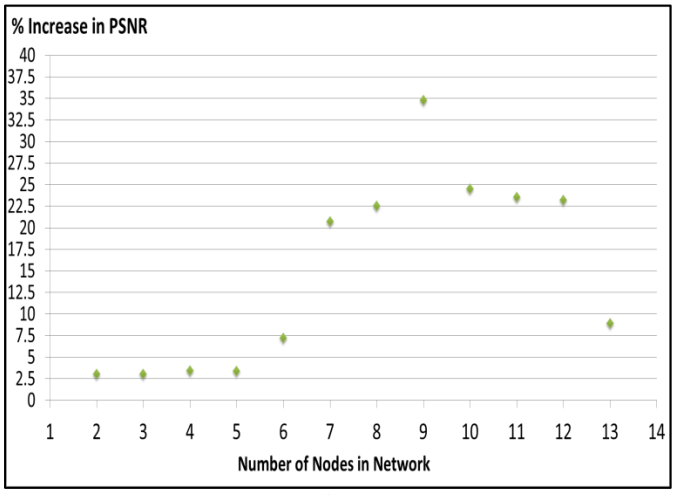

Fig. 5. Percentage Quality Increase of Decentral Architecture over Central Architecture. An observation of the advantage held by the decentral architecture for increasing loads.

Figures 4 and 5 show the improvement a decentral architecture has over its central counterpart. When the number of nodes in the system is low, there is almost no quality improvement as video reception in both systems is of high quality. However, as the load on the central server increases with an increase in number of users, quality of received videos starts to drop at a high rate. This is due to the insufficient upload bandwidth of the central server for the increased workload. The same cannot be said for the decentral architecture. Figure 4 shows that the decentral architecture maintains an excellent to good video quality (based on MOS) for the same increase in workload as the central system. Figure 5 shows that when serving 9 nodes, the average video quality is around 34 % higher than the central architecture. On average, a 15 % higher PSNR average is observed in the decentral network. This improvement can be attributed to the division of responsibility principle applied in the network. The increased workload is shared amongst the peers evenly. This means that with an increase in workload as a result of an increasing user base, network resources grow in size and are able to serve more peers.

However, it is clear from Figure 4 and 5 that the above mentioned principle does not hold true with a further increase in workload. As the user base grows beyond 12 users, the quality improvement observed earlier in the decentral architecture begins to drop off at a high rate. Video quality of both architectures drops off at very high loads, with a higher rate of decline associated with the central architecture. This is expected for the central architecture but unexpected for the decentral architecture as more users entail more network resources. Reduction in quality in the decentral architecture under very high loads can be attributed to the high demand placed on each peer to serve higher requests in the network. Peers do not have the capability of serving many other peers as they are not equipped with the resources of a central server. Thus, at very high workloads, as a result of an increased user base, a ‘central nature’ appears in the decentral architecture, where peers have to perform like resource intensive servers.

Implementation of Step (e), from the method described in Section IV-B, allows for a quantification of the difference in scalability between the architectures. An inspection of Figure 4 shows that, at 28 dB, representative of good quality video, the decentral architecture is capable of serving on average, 34 % more peers than its central counterpart.

PSNR Evaluation of a Serving Peer Protocol implementation in the Decentral Architecture

A serving peer protocol adapted from the helper protocol presented in Yifeng *et al* [12] is presented in this section. Implementation of this protocol is performed in NS2. As Figure 4 shows, the basic decentral system is not adapted to very high system loads. The serving peer protocol works in the following way. The protocol assumes that there are peers available with spare uplink bandwidth and the same content as the peers that are overloaded. When the system load exceeds one request per user during a simulation period, the protocol is deployed. It chooses a peer with spare uplink bandwidth to serve the requesting peer with the same content that was being uploaded by the overloaded peer.

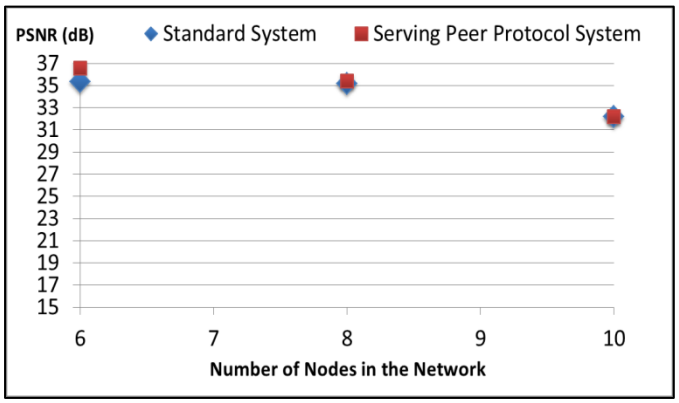

Fig. 6. Performance Evaluation of Serving Peer Protocol under Normal System Load for a Decentral Architecture. A performance comparison of the standard decentral architecture to decentral architecture with the serving peer protocol shows minimal difference.

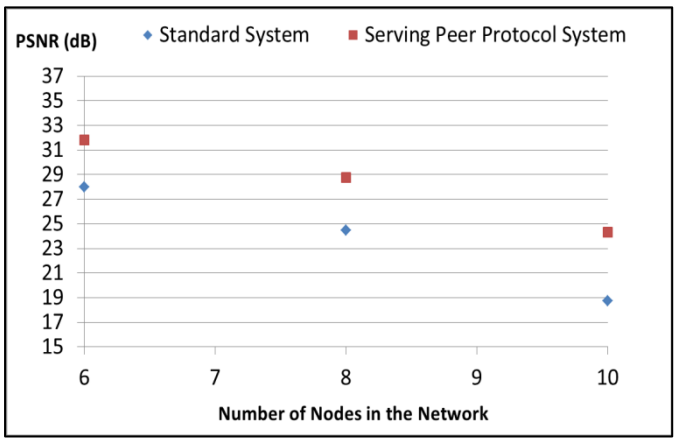

Fig. 7. Performance Evaluation of Serving Peer Protocol under High System Load for a Decentral Architecture. A performance comparison of the standard decentral architecture to decentral architecture with the serving peer protocol shows an average quality improvement of 30 %.

Results of the serving peer protocol performance, in increasing PSNR (video quality) in the system, is shown in Figures 6 and 7. At medium system loads, the effect of the serving peer protocol is less pronounced. Peers are able to serve contents when their uplink bandwidth is not saturated. Improvement in performance is seen in Figure 7 where system loads are very high due to high user requests. Peers in the standard architecture with over demanded contents are unable to effectively serve requests, dropping packets as a result of saturated uplink bandwidth. Standard system performance is shown to degrade to low values in Figure 7 as expected. Introduction of the serving peer protocol in the decentral system under high load shows an average quality improvement (PSNR increase) of about 30 % for a range of nodes in the system.

Effectiveness of the serving peer protocol is highlighted by the improvement observed in Figure 7 compared with that of Figure 6. However, many P2P protocols exist for the purpose of improving streaming capabilities and quality in P2P systems. The importance of evaluation of the serving peer protocol in this paper is the ability of the quality measurement system to quantifiably evaluate media streaming protocols.

CONCLUSION

It is widely accepted that a client server model suffers from poor scalability. Decentralized systems were acknowledged and extensively developed as an improvement to the scalability issues of central systems. It was proposed that it is possible to quantify the scalability of a decentral system over a central system. A Peak Signal to Noise Ratio quality measurement system that operates network wide was developed and was used to demonstrate quality and scalability benefits of a decentral architecture over a central architecture. Both architectures were developed by adhering to the same hardware criteria, in terms of network storage and bandwidth, thereby allowing a fair comparison. It was observed that on average, for medium to high workloads, the decentral architecture delivers 15 % higher quality video to peers than the central architecture. It was also inferred, by maintaining the same quality evaluation standards in both architectures, that the decentral architecture is around 34 % more scalable at high workloads. However, the decentral architecture starts to behave similarly to a central architecture as workloads become very high. This is attributed to the peers having to perform like centralized servers due to over demanded content, resulting in the saturation of their resources. Evaluation of a serving peer protocol implemented in the decentral architecture, showed a 30 % video quality increase under very high load, when compared to the standard decentral architecture. The system proposed was successful in determining the nature of central and decentral systems in terms of quality and scalability. Successful numerical evaluation of a P2P protocol by the quality evaluation system highlights the potential of the system in formalizing new experiential quality testing methods for media streaming systems.

FUTURE WORK

Focusing on the developments of other works in the field of media streaming, it is possible to envisage this quality measurement system being used as a performance indicator. Observations of the decentral architecture have revealed flaws at very high workloads. This work uses two rudimentary architectures to determine the basic nature of central and decentral systems. It is possible to use our system as a test bed to determine how new protocols improve on the simple architectures depicted here.

REFERENCES

[1] D. Kerr, Online Communication, May 2013. [Online]. Available: http://news.cnet.com/8301-1023_3-57584535-93/video-streaming-is-on-the-rise-with-netflix-dominating/. [Accessed 2 June 2013]

[2] C. Ke, C. Shieh, W. Hwang and A. Ziviani. (2008, Aug.) An Evaluation Framework for More Realistic Simulations of MPEG Video Transmission. *Journal of Information Science and Engineering, Vol. 24. 2, pp. 425-440*.

[3] Y. Guo, K. Suh, J. Kurose and D. Towsley. (2003). P2Cast: Peer-to-Peer Patching Scheme for VoD Service. Presented at WWW '03 Proceedings of the 12th international conference on World Wide Web, pp. 301-309.

[4] Y. Guo, K. Suh, J. Kurose and D. Towsley. (2003, July). A peer-to-peer on-demand streaming service and its performance evaluation. Presented at 2003 International Conference on Multimedia and Expo, vol. 2, pp. 649-652.

[5] M. Hefeeda, A. Habib, D. Xu, B. Bhargava and B. Botev (2005, November). CollectCast: A Peer-to-Peer service for media streaming. *Journal of Multimedia Syst., Vol. 11. 1, pp. 68-81.*

[6] H Deshpande, M Bawa, H Garcia-Molina. "Streaming Live Media over a Peer-to-Peer Network". Stanford InfoLab Publication Server, California, Tech. Rep. TR-501, Dec. 2008.

[7] K. Xiong and H. Perros. Service Performance and Analysis in Cloud Computing. Presented at 2009 World Conference on Services - I., pp 693-700.

[8] H. Khazaei, J. Misic and V.B. Misic. (2012, May). Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems. *IEEE Transactions on Parallel and Distributed Systems, Vol. 23. 5, pp. 936-943.*

[9] J.P. Munoz-Gea, S. Traverso and E. Leonardi. (2012, November). Modeling and Evaluation of Multisource Streaming Strategies in P2P VoD systems. *IEEE Transactions on Consumer Electronics, Vol. 58. 4, pp. 1202-1210.*

[10] The Network Simulator - ns-2. [Online] Available: http://www.isi.edu/nsnam/ns/. [Accessed 10 November 2012].

[11] Evalvid-RA. [Online] Available: http://www.item.ntnu.no/~arnelie/Evalvid-RA.htm. [Accessed 4 February 2013]

[12] H. Yifeng and L. Guan. (2010, Nov.) Solving Streaming Capacity Problems in P2P VoD Systems. *IEEE Transactions on Circuits and Systems for Video Technology, Vol. 20. 11, pp. 1638-1642.*

[13] YUV Video Sequences. *Video Trace Library.* [Online] Available: http://trace.eas.asu.edu/yuv/. Accessed [23 August 2013]