LIVE PEER-TO-PEER STREAMING WITH SCALABLE VIDEO CODING AND NETWORK CODING

by

Shabnam Mirshokraie
B.Sc., Ferdowsi University of Mashhad, Mashhad, Iran, 2006

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2010

APPROVAL

Name: Shabnam Mirshokraie

Degree: Master of Science

Title of Thesis: Live Peer-to-Peer Streaming with Scalable Video Coding and Network Coding

Examining Committee:

Dr. Thomas Shermer,
Professor of Computing Science
Chair

Dr. Mohamed Hefeeda
Senior Supervisor
Associate Professor of Computing Science

Dr. Joseph Peters
Supervisor
Professor of Computing Science

Dr. Jiangchuan Liu
Examiner
Associate Professor of Computing Science

Date Approved: November 30, 2010

Declaration of
Partial Copyright Licence

Abstract

Providing high-quality streaming over peer-to-peer (P2P) systems faces multiple challenges such as limited upload capacity of peers and high heterogeneity of receivers in terms of download bandwidth and screen resolutions. In this thesis, we present the design of a P2P live streaming system that uses scalable video coding (SVC) as well as network coding. The proposed design enables flexible customization of video streams to support heterogeneous receivers, highly utilizes upload bandwidth of peers, and quickly adapts to network and peer dynamics. The proposed design is simple and modular. Therefore, other P2P streaming systems could also benefit from various components of the proposed design to improve their performance. We conduct an extensive quantitative analysis to demonstrate the expected performance gain from the proposed design.

Keywords: peer-to-peer streaming; scalable video coding; network coding; live streaming;

Acknowledgments

I would like to express my deepest gratitude to my senior supervisor, Dr. Mohamed Hefeeda, for mentoring me in the past two years with his patience and knowledge. It was his guidance and encouragement that taught me how to do research, and without his support, completion of this thesis would not have been possible for me.

I would like to express my gratitude to Dr. Joseph Peters, my supervisor, and Dr. Jiangchuan Liu, my thesis examiner, for being on my committee and reviewing this thesis. I also would like to thank Dr. Thomas Shermer for taking the time to chair my thesis defense.

I am grateful to all the members at the Network Systems Lab for providing me a stimulating and fun environment. A big thank-you goes to Cheng-Hsin Hsu, who has helped me a lot for my academic work. The knowledge that I have gained through the insightful discussions with Cheng are invaluable.

Last but certainly not least, I owe my deepest gratitude to my parents for their love and endless support. I will never forget what I owe them, and my eternal gratitude to them cannot be expressed by words.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we briefly introduce peer-to-peer (P2P) streaming systems. Then, we describe the problem addressed in this thesis and summarize our contributions. The organization of this thesis is given at the end of this chapter.

1.1 Introduction

P2P live video streaming has been proposed to provide live multimedia contents at low cost for large number of participants [1–3]. Recently, it has seen wide deployment around the globe. Such attraction towards this technology lies in two key reasons. First, unlike Ip multicast systems, it does not require support from Internet routers and network infrastructure. IP multicast is a technique for real-time communication over an IP infrastructure. In Ip multicast, the nodes in the network (network switches and routers) take care of replicating the packet to reach multiple receivers. However, in P2P streaming system, peers make a portion of their resources, such as processing power, disk storage or network bandwidth directly available to other network participants, without the need for central coordination by servers or stable hosts. Peers are both suppliers and consumers of resources. Therefore, it is cost-effective and easy to deploy [4]. Second, a participant in a live streaming session is not only downloading a video stream, but also uploading it to other participants watching the same video [4]. Thus, the available system resources increase with more participants, which makes the P2P system scalable.

P2P streaming systems such as CoolStreaming [5], PPLive [6], UUSee [7], SopCast [8], and TVAnts [9] attract numerous users every day. As more users get used to viewing

multimedia content online, they will demand higher and better video quality than available on many of the current P2P streaming systems. As an indication of this demand and the response from industry, Huang et al. [10] show that the average bit rate of videos offered by the MSN Video Services has increased by more than 50% over a nine-month period, and it is the likely that the bit rate will continue to increase in the future. Providing high-quality streaming over P2P systems, however, faces multiple challenges, including: (i) limited upload capacity of peers, (ii) high heterogeneity of receivers in terms of download bandwidth, screen resolutions, and CPU capacity, and (iii) high churn rate as the peer population is constantly changing. Addressing these challenges requires not only increasing the capacity of peers and deploying additional seeding servers to make up the shortage in resources, but also employing novel methods for encoding and distributing multimedia content and developing algorithms and protocols to optimally utilize the available resources.

P2P streaming systems are often divided into two major categories: tree-based and mesh-based (also known as swarm-based and data driven). In tree-based systems, peers organize themselves into one or more multicast trees and data will be pushed along the tree structure [11,12]. Tree-based systems incur high costs for the management and maintenance of the tree structure, especially with high peer churn rates. Mesh-based systems allow peers to self-organize in mesh-shaped graphs [5,13,14]. These systems usually yield better performance in practice as they are more robust against high-level of peer and network dynamics [15]. Our work focuses on mesh-based P2P streaming systems.

1.2 Thesis Statement and Contributions

In this thesis, we propose a new design for P2P live streaming systems that significantly improves their performance. The new design strives to address many of the challenges impeding current systems by efficiently utilizing peers' resources, easily customizing multimedia content to support receivers with diverse resources and requirements, and quickly adapting to network and peer dynamics. The proposed design is simple and practical; we actually have implemented it. The proposed design employs scalable video coding to support heterogeneous receivers as well as network coding to maximize the streaming throughput and handle network dynamics. Although scalable video coding and network coding have been *individually* proposed for various systems in the literature, e.g., [16–21], their *integrated* use in P2P live streaming systems has not been fully explored in the literature, to the best

of our knowledge. The integration of these two technologies provides many performance benefits beyond those offered by each of them individually, as will be shown in this thesis.

In particular, the contributions of this thesis can be summarized as follows [22]:

- We present the design of a P2P streaming system that employs both scalable video coding and network coding. The proposed design is modular and can be used as an improvement plug-in in other P2P streaming systems. That is, we focus on the new components needed to handle multimedia content compressed in scalable manner and encoded using network coding. Thus, our work and software components are readily useful for other systems.

- We *quantitatively* show the expected performance gain from the proposed design using actual scalable video traces in realistic P2P streaming environments with high churn rates, heterogeneous peers, and flash crowd scenarios.

- We show that the proposed system can achieve: (i) significant improvement in the visual quality perceived by peers (several dBs are observed), (ii) smoother and more sustained streaming rates (up to 100% increase in the average streaming rate is obtained), (iii) higher streaming capacity by serving more requests from peers, and (iv) more robustness against high churn rates and flash crowd arrivals of peers compared to systems that use only scalable video coding (SVC), only single layer streams, or systems that employs both single layer streams and network coding.

1.3 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, we provide a brief background on network coding and scalable video coding, and we summarize the previous work in the literature. In Chapter 3, we describe the proposed system and its main components. In Chapter 4, we evaluate the proposed system using actual video traces, and we conclude the thesis in Chapter 5.

Chapter 2

Background and Related Work

In this chapter, we present an overview of P2P systems and in particular P2P streaming systems. We also describe network coding and scalable video coding, which are employed in the proposed P2P streaming systems.

More details about network coding can be found in [23–25] and the references therein. More information about scalable video coding can be found in [26] which describes the recent H.264/SVC standard.

2.1 Overview of P2P Systems

P2P systems are a class of distributed systems in which participants (peers) cooperate to achieve a desired service. The P2P paradigm brings several benefits including improved scalability, eliminating the need for having centralized servers, and cost effectiveness by utilizing peers resources. In P2P systems, peers form an overlay network among each other. Peers can join and leave frequently which makes the network topology highly dynamic. Therefore, such systems should be self-organizing.

File sharing and media streaming are among the most widely deployed applications for P2P systems. File sharing systems are used for storing and providing files to requesting peers. Many examples of file-sharing applications are currently running such as Bittorrent [27], and Gnutella [28]. In next section, we briefly explain P2P media streaming systems.

2.1.1 P2P Streaming Systems

In file-sharing P2P applications, a client download the entire file before start using it. However, in real-time streaming applications, a user starts playing out the requested movie after a short (e.g., order of seconds) waiting period. In this section, first we provide a brief introduction to two general P2P streaming types, live and on-demand streaming. Then, we focus on two main approaches towards streaming data dissemination, namely, tree-based and data driven.

2.1.2 On-Demand and Live Streaming Systems

P2P streaming systems are categorized into two types called live streaming and on-demand streaming. Live video streaming refers to sending video content in real time over the Internet. In a P2P live streaming system, a video is created and streamed to clients synchronously , that is all peers watch the same part of the video at the same time. Such systems include CoolStreaming [5] and PULSE [29]. In a P2P on-demand streaming system, media content is stored beforehand on media servers. Peers join a streaming session at any time and start watching from any part of the video. They can also do operations like pause and roll back. PPLive [6], PPStream [30], and UUsee [7] are examples of on-demand P2P streaming systems.

2.1.3 Tree-Based Approaches

In this approach, peers are organized into tree structures with well-defined parent-child relationships. Such approaches are typically push-based, meaning that when a node receives a data packet, it forwards (pushes) copies of the packet to each of its children. It is critical to maintain the structure as nodes join and leave the tree. If a node crashes or stops performing adequately, all of its offspring in the tree will stop receiving packets, and the tree must be repaired.

In tree-based approaches, the failure of nodes may disrupt the delivery of data to a large number of users. Moreover, a majority of nodes are leaves in the tree structure. Therefore, their outgoing bandwidth cannot be utilized. In response to these concerns, more resilient approaches have been proposed. In particular, data-driven approaches have gained popularity in the research community.

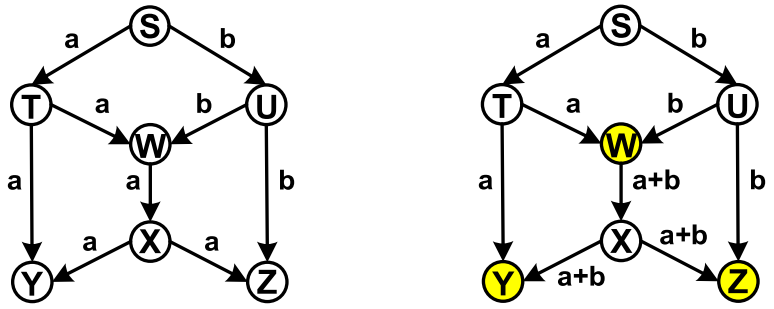
2.1.4 Data-Driven Approaches

Data-driven overlay designs do not maintain an explicit structure for delivering data. Unlike tree-based approaches, they use the availablity of data to guide the data flow. In data-driven approaches, using random push algorithms for forwarding packets may cause significant redundancy with the high-bandwidth video. To address this problem, approaches such as Cool-Streaming [5] and Chainsaw [31] adopt pull-based methods, that is, nodes maintain a set of neighbors, and periodically exchange data availability information with them. The overlay structure is resilient to failure and departure of any node since its neighbors use other neighbors to receive data packets. Furthermore, in contrast to tree-based approaches, the available bandwidth between peers can be fully utilized.

2.2 Network Coding

In traditional packet forwarding methods, each node simply repeats data packets destined to other nodes in the network. In contrast, with network coding, each node combines a number of packets it has received or created into several outgoing packets. The network coding concept enables source and intermediate nodes to perform simple operations on packets before forwarding them. These operations allow nodes to send partial information to the destination. After receiving all the necessary partial information, the receiver will be able to recover the original packet.

The principle of network coding is explained with a simple example [25] in Figure 2.1. In this example, the source S has access to bits a and b at a rate of one bit per unit time. It aims to communicate these bits to two sinks Y and Z so that both sinks can receive both bits per unit time. All links capacities are one bit per unit time. If we use the traditional store and forward method, the central link would be able to carry a or b. If a is sent through the central link, the left destination (Y) would receive a twice, without receiving b. Sending b results in a similar problem for the right destination (Z). Thus, as shown in Figure 2.1(a), no routing scheme can transmit both a and b simultaneously to both destinations. Using network coding, as shown in Figure 2.1(b), both a and b can be received by both destinations simultaneously by sending $a+b$ through the central link, where + denotes xor operation. At node Y, b is recovered from a and $a+b$. Similarly, a can be recovered at node Z. Therefore, both sinks are able to receive both bits per unit time by encoding the information at the intermediate node W. this simple example illustrates the potential of employing network



(a) Traditional method (b) Network Coding

Figure 2.1: A simple example to demonstrate the potential of using network coding to increase the throughput achieved by all nodes in the network.

coding to increase the total amount of data that can be exchanged in the network.

Linear network coding, in general, is similar to this example. The difference is that the xor operation is replaced by a linear combination of the data, interpreted as numbers over a finite field.

Packing information at intermediate or source nodes is called encoding and extracting real data from encoded ones is referred to as decoding. Encoding and decoding are linear operations over a Galois Field of size 2^l, which is denoted by GF(2^l). A GF(2^l), for some integer $l > 0$, is a finite field with 2^l elements. In GF(2^l), elements can be stored using l bits [32]. Addition, subtraction, multiplication and division are performed on strings of l bits [24]. Addition is the standard bitwise xor operation. For multiplication, we can represent the sequence $s_0, ..., s_{l-1}$ of l bits as the polynomial $s_0 + s_1X + ... + s_{l-1}X^{l-1}$. Then, we need to take a polynomial of degree l that is irreducible over GF(2). Multiplication is obtained by calculating the product of the two polynomials, and then computing the remainder modulo the chosen irreducible polynomial [32]. Division can be computed by the Euclidian algorithm [24, 32, 33].

Encoding is a linear combination of blocks, which is formulated as: $x = \sum_{n=1}^{N} c_n \cdot b_n$, where N is the total number of blocks, c_ns are coefficients of size l bits taken from GF (2^l) and b_ns are data blocks of size k bytes. As shown in Figure 2.2, each block consists of a vector of $(k \times 8)/l$ symbols. The symbol x represents one encoded block of size k. Each encoded block is a linear combination of the original blocks. Thus, it is uniquely identified by the set of coefficients included in the linear combination. Multiplications and additions are done over GF (2^l).

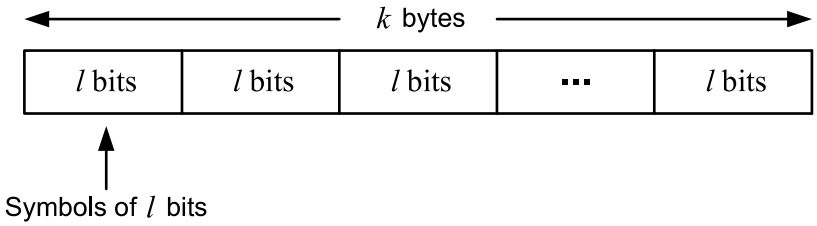


Figure 2.2: Data block over GF (2^l) is divided into l-bits symbols. Finite field operations are done on symbols.

Assuming a node receives a set of (C^1, x^1), (C^2, x^2), ..., (C^N, x^N) packets, where C^n is the vector of coefficients $(c_1^n, c_2^n, \ldots, c_N^n)$ for block x^n $(1 \leq n \leq N)$. The decoding process is performed by solving: $x^m = \sum_{n=1}^{N} c_n^m \cdot y^n$, $(1 \leq m \leq N)$, where y^ns are unknowns. This is a system of linear equations with N unknowns (the original data blocks) and N equations (created by receiving N packets and each packet n $(0 \leq n \leq N)$ carries the coefficient vector C^n and the encoded data block x^n). This system of equations can be solved by the Gaussian elimination method.

2.3 Scalable Video Coding

In today's multimedia systems, clients are getting quite heterogeneous in terms of connection bandwidth, processing power, and display resolution. Scalable video coding techniques can support this heterogeneity by encoding a video stream once and extracting it in several ways according to receiver's capabilities. While nonscalable streams provide very limited flexibility in supporting heterogeneous receivers, multimedia streams created using scalable video coding techniques can support a wide range of diverse receivers and network conditions. This ease of adaptation comes at a cost of reduced coding efficiency. However, recent scalable video coding techniques have improved the coding efficiency. For example, the scalable video coding (SVC) extension of the H.264/AVC video coding technique [34], known as the H.264/SVC standard [26], generates highly flexible video streams at a bitrate close to the bitrate of the corresponding nonscalable video stream.

The H.264/SVC video coding standard adds scalability to the widely used H.264/AVC video coding technique. The H.264/SVC standard supports temporal, spatial, and quality scalability at the same time. Temporal scalability is achieved by employing a hierarchical prediction structure among video frames belonging to the same group of pictures (GoP), in

which frames of higher temporal layers can only be predicted from lower temporal layers. In the spatial scalability, a spatial layer s of a frame can be predicted from the s-th spatial layer of some other frames (in lower temporal layers), as well as lower spatial layers in its own frame.

For providing quality scalability, there are two possibilities. The first one follows the spatial scalability structure, but assigns the same resolution and different quantization parameters to layers. This produces a coarse-grained scalable (CGS) video with limited number of quality layers. A finer granularity can be provided by the second possibility, which uses medium-grained scalability (MGS) coding to divide a single CGS quality layer into multiple sub-layers, which are referred to as MGS layers. This is done by partitioning the residual discrete cosine transform (DCT) coefficients of a CGS layer into multiple MGS layers. A stream can be truncated at any CGS or MGS layer. In addition, some packets of an MGS layer can be discarded, while the remaining ones can still be decoded to improve quality. Packet discarding can be done in arbitrary ways, depending on the bitstream extraction process [35]. H.264/SVC allows up to 7 temporal, 8 spatial, and 16 quality layers [26].

Using scalable video coding, users with high link capacities experience better video quality by receiving more layers, while others with lower bandwidth get quality proportional to the number of layers they can receive.

2.4 Related Work

Most of the currently deployed P2P streaming systems, e.g., [5–9], use nonscalable video streams. Thus, they serve a single-version of the video stream to all peers, and they have limited support for heterogeneous peers. To address these issues, a number of works have proposed P2P streaming systems with scalable video streams, e.g., [13, 21, 36–39]. Cui and Nahrstedt [36] present an algorithm to decide for each peer how to request video layers from a given set of senders. They assume layers have equal bitrate and provide equal video quality. Liu et al. [40] propose another approach to the layered P2P streaming problem with the goal of minimizing the resource consumption of the media server and maximizing the streaming qualities of the receiving nodes. They formulate an optimization problem and propose an approximation algorithm, called FABALAM, to simplify the problem. However, FABALAM relies on static mapping of layer-to-sender and a layer is provided by only one sender. Hefeeda and Hsu [38] study this problem for Fine-Grained Scalable (FGS) videos,

taking into account the rate-distortion model of the video for maximizing the perceived quality.

Rejaie and Ortega [37] present a framework for layered P2P streaming, where a receiver coordinates the transmission of video packets from multiple senders using a TCP-friendly congestion control mechanism. Lan et al. [39] propose a scheduling algorithm for peers to request data from senders. The allocation of seed server resources in P2P streaming systems with scalable videos has also been considered in [21]. Magharei and Rejaie [13] introduce PRIME to reduce bandwidth bottlenecks by choosing an efficient pattern of delivery and proper peer connectivity. However, PRIME uses multiple description coding (MDC) while we use SVC in our work. While the above works enable serving streams with different qualities to peers with diverse resources, none of them employs network coding to further enhance the utilization of peer resources.

Network coding has been shown to maximize the throughput and bring various performance benefits in different environments [25, 41, 42]. For example, in wireless networks, network coding improves the message delivery probability for ad-hoc multicast protocols [43], and it overcomes broadcast storm problems [44]. Network coding has also been proposed for P2P file-sharing systems. For example, in the Avalanche system [19, 45], the authors use randomized network coding to efficiently distribute files and to decrease the download time. The authors provide a method to ensure that any piece uploaded by a peer can be useful to other peers. However, these techniques are not applicable to P2P streaming systems, which have strict timing constraints and packet sequence requirements.

Several works have proposed using network coding for P2P streaming applications. Annapureddy et al. [46] show that network coding improves the video streaming quality in video-on-demand (VoD) services. Network coding is applied on a segment of a single-layer stream. Feng and Li [47] develop analytical models to show the benefits of using network coding in live P2P streaming systems. Wang and Li [48, 49], address practical aspects of using network coding in P2P streaming systems such as benefits and trade offs of applying network coding in P2P live streaming as well as the architectural design challenges in implementing network coding. All of these works confirm the viability of network coding in different applications. However, none of them has considered integrating network coding with scalable video coding to support wider ranges of clients. They basically improve the performance of single-layer P2P streaming systems.

Recently, a few works have considered both network coding and scalable video streams

[50–52]. For example, Zhao et al. [50] try to provide each end user in a multicast session with the maximum number of layers through solving an optimization problem using a greedy algorithm. While in [51], Chenguang et al. formulate an integer linear programming problem to solve the same problem. Unlike our work, the above works target tree-based P2P streaming systems, and they assume that peers know the global tree structure and this structure is fairly static. These assumptions typically do not hold in practice. In contrast, we target the highly dynamic mesh-based P2P streaming systems with no assumptions/constraints on the topology. In [52], Nguyen et al. explore the feasibility of combining network coding and SVC. They design and evaluate Chameleon, an adaptive P2P streaming protocol designed to incorporate network coding with scalable video coding. In this work, they propose a quality based neighbor selection method. In this algorithm, peers are classified based on their streaming quality level. Each peer selects a neighbor whose average streaming quality level is closest to its class. However, they do not provide extensive performance evaluation to validate their proposed protocol. They evaluate their system based on two performance metrics: (i) the percentage of segments skipped during playback, and (ii) the average quality satisfaction. While our work is independant from neighbor selection algorithms, we conduct an extensive quantitative analysis to demonstrate the expected performance gain from the proposed design.

Finally, Nguyen et al. [53] propose hierarchical network coding (HNC) to be used with scalable video coding. HNC performs network coding across all layers of the same video stream to provide higher error protection to lower video layers. HNC is designed to reduce the impact of packet losses. However, it assumes that most users are capable of or willing to receive all video layers. For example, a limited bandwidth receiver that is interested in only a 2-layer version of the stream may end up receiving data blocks from higher layers, although the data cannot be used. Thus, the bandwidth of peers can be wasted. This implies that HNC will not efficiently support heterogeneous clients. In addition, performing network coding on all layers will increase the size of the coefficient matrix needed for network coding operations. Since the time and space complexities of the encoding and decoding processes depend on the size of the coefficient matrix, HNC will impose a significant overhead on peers, which have limited-resources in the first place. Furthermore, the work in [53] did not provide a rigorous quantitative evaluation of HNC in real dynamic P2P environments, as we do. Our proposed system performs intra-layer network coding and is fairly efficient.

Chapter 3

Proposed P2P Streaming System

In this chapter, we describe the proposed P2P live streaming system that employs network coding and scalable video coding. We start with a high-level overview, followed by more details.

3.1 Introduction

We target mesh-based P2P streaming systems which have been widely used in practice [5, 13, 14]. In our system model, there are three entities: tracker, source, and peer. The tracker is used for coordination between all peers. It keeps a list of all currently active peers and it performs peer matching whenever a peer asks for a list of neighbors. We use a random peer matching algorithm. That is, the tracker randomly selects among the list of active peers who are viewing the same video stream and returns them to the requesting peer as potential senders. This matching results in multiple dynamic swarms in the system.

There is at least one source node in the system to introduce new streams to peers. The source node (sometimes called seed server) also provides additional capacity in case that peers do not have enough resources, especially in the beginning of new streaming sessions where very few peers exist. Source nodes perform network coding operations on the scalable video streams in order to prepare them for distribution in the system. Peers act as receiving clients as well as share some of their upload bandwidth to serve other peers. As receivers, peers decode network-coded data received from others and process this data to create proper scalable video streams to ensure smooth video quality. As senders, peers encode video data using network coding with parameters based on their own upload capacity as well as the

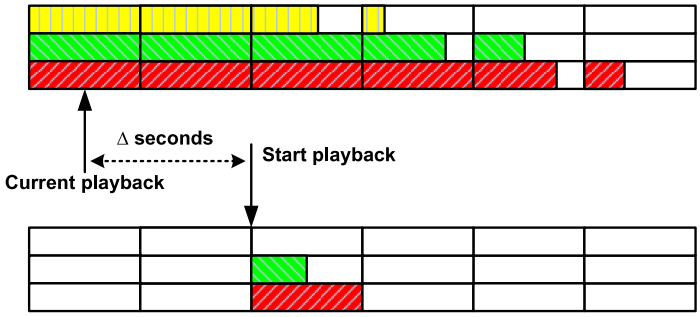


Figure 3.1: Initial buffering delay allows the receiver to skips few segments. The receiver buffers encoded blocks from segments that are Δ seconds after the current playback point.

characteristics of the receiving peers.

In addition to the above components, we use buffer maps in order to exchange availability information among peers. We use two bits to represent the status of a layer. One bit, called availability bit, is used to determine the decodability of a packet. That is, if all necessary blocks for fully decoding a packet are successfully received, the corresponding availability bit is set to 1 and otherwise is set to 0. Another bit, called serving bit is used to identify whether a network-coded packet can be served to other peers or not. Meaning that, immediately after receiving one coded block for a given packet, the serving bit for this packet is set to 1.

When a peer joins a session, it is informed about the availability information through receiving buffer maps from its initial neighbors (assigned by tracker). It also receives the current segment being played in the current live streaming session. Each peer maintains segments played in near future in its playback buffer. As shown in Figure 3.1, the newly joined peer does not start receiving encoded blocks from exactly the same segment that is currently playing. The receiver requests to receive segments that are Δ seconds after the current playback point. Δ is called initial buffering delay. This allows the peer to receive some segments in its playback buffer before the playback starts. Therefore, it has enough time to receive some of the encoded blocks from future segments and decode them ahead of time.

A simplified model for the software architecture of a peer in our system is shown in Figure. 3.3. A similar model is used for source nodes, but with some differences as elaborated later. We do not address the design or optimization of trackers; the function of the tracker is orthogonal to the work presented in this thesis. We also do not address other problems in mesh-based P2P streaming systems, including neighbor selection, gossip protocols (for

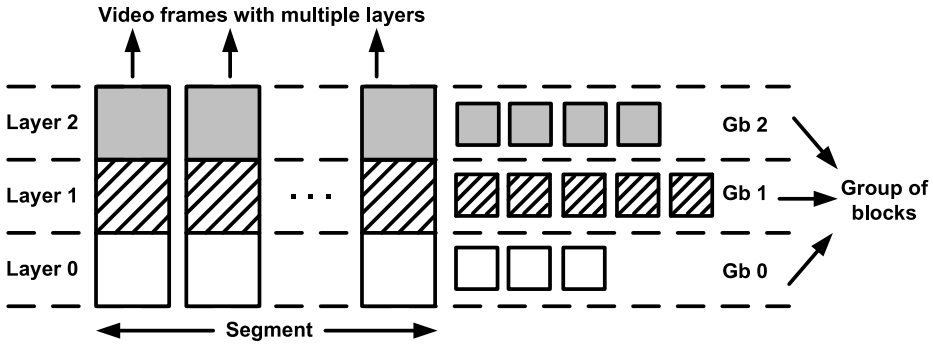


Figure 3.2: An example of the segmentation method used in combination of network coding and SVC.

exchanging data availability), incentive schemes, and overlay optimization—which all have been heavily researched in the literature. All of the above issues are abstracted in the Connection Manager component in Fig. 3.3, while our work is focused on the components in the shaded box in that figure. The separation and abstraction of functions enable us to support different P2P streaming systems with minimal changes in our design and code. Therefore, our work is fairly general.

For quick reference, we list all symbols used in this chapter in Table 3.1.

3.2 Source Node Software Architecture and Functions

The source node prepares video streams before introducing them into the system. A video stream is encoded into multiple layers in a scalable manner. The video stream is divided into equal-length segments. Each segment consists of a fixed number of frames. The number of frames is chosen to make an integer number of group of pictures (GOP), e.g., 2 GOPs. GOP structure specifies the order in which frames are arranged. GOP is the set of pictures between two successive pictures of the temporal base layer together with the succeeding base layer. Each GOP contains a fixed number of video frames. For example, a segment can contain 30 frames in a video encoded into GOPs with 15 frames each. Since we consider scalable streams, each video frame is composed of multiple layers. We apply network coding operations on the data contained in individual layers as follows. We divide each layer in a segment into fixed-size blocks. Then these blocks are encoded. Notice that different layers may contain different number of blocks, depending on the visual complexity of the video frames contained in each segment. Figure 3.2 shows an example of the segmentation method

Table 3.1: List of symbols used in this chapter.

Symbol	Description
N_l	The number of blocks in layer l.
$b_{i,l}$	The ith block in layer l ($1 \leq i \leq N_l$).
$c_{i,l}$	The ith coefficient corresponding to the ith block in layer l ($1 \leq i \leq N_l$).
x	One generated encoded block.
$(C^{i,l}x^{i,l})$	The ith vector of coefficients corresponding to the ith encoded block in layer l ($1 \leq i \leq N_l$).
$(ReCoef^{i,l}, ReBl^{i,l})$	The ith vector of coefficients corresponding to the ith re-encoded block in layer l.
$c_j^{i,l}$	The jth element of the ith coefficient vector in layer l ($1 \leq i, j \leq N_l$).
B	The block size.
L	The number of layers.
P	The number of potential senders.
Ps	The list of potential senders.
Ps_p	The pth sender from the list of potential senders.
S	List of segments.
$Size_s$	Size of segment $s \in S$.
$Max_{s,p}$	Maximum number of encoded blocks from segment $s \in S$ assigned to sender $p \in Ps$.
E_s	The number of encoded blocks generated for segment $s \in S$.
Up_p	The upload bandwidth of sender $p \in Ps$.
$Down_I$	The download bandwidth of receiver I.
d_s	The deadline of segment $s \in S$.
$BufMapMsg$	The buffer map message.
α	The communication overhead ratio.

used in combination of network coding and scalable video coding. As shown in the figure, network coding operations are applied on different group of blocks for different layers.

The encoding process is applied at the source node by using random network coding. On intermediate nodes, i.e., uploading peers, the blocks are re-encoded with different coefficients. In both cases, the coefficients of each block are attached to the block itself during transmission.

3.3 Peer Software Architecture and Functions

The main functions of a peer in our system are summarized in Figure 3.3. These functions are divided into two parts: (i) receiving, and (ii) uploading. The interaction among these two parts are coordinated through the sharing buffer. We describe the receiving and uploading parts of the peer model in the following two subsections.

3.3.1 Receiver Part

At the receiver side, a peer interested in receiving a specific part of the video stream, determines and requests a proper number of encoded blocks through the Download Scheduler. The Download Scheduler computes the number of required encoded blocks based on the currently available bandwidth and the number of video layers that the receiver is interested in. It then assigns each of the active senders in the session a number of blocks proportional to its upload bandwidth.

Immediately after receiving any encoded block through the network, the block is forwarded to the Progressive NC Decoder component and its coefficients are accumulated in the coefficient matrix. The Progressive NC Decoder rearranges the coefficients and encoded block matrices. This is done through one round of the Gauss-Jordan elimination method [48]. The Gauss-Jordan elimination method is a version of the Gaussian elimination method which inserts zeros above and below the pivot elements in the matrix as it goes from the top row to the bottom one. In other words, the Gauss-Jordan elimination method converts a matrix to its reduced row echelon form (RREF) where every leading coefficient is 1 and it is the only nonzero element in its column.

We illustrate progressive decoding with Gauss-Jordan elimination through an example in Figure 3.4. In this example, we consider 3 blocks with one byte of data in each of them.

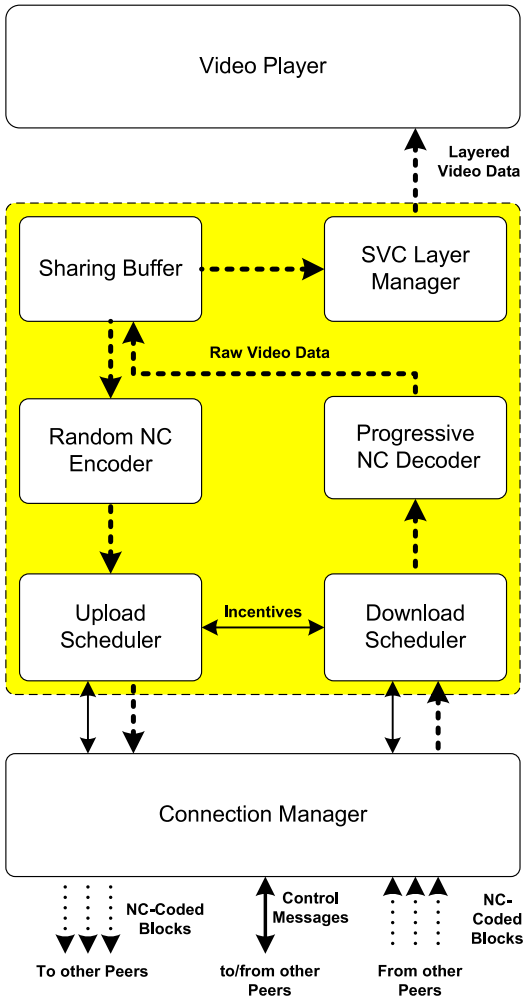


Figure 3.3: Peer software architecture. Dashed arrows denote video data, and solid arrows denote control messages.

Operations are over Galois Field of size (2^8). By receiving the second encoded block, Gauss-Jordan elimination is applied on the first encoded block as well as its coding coefficients. The result is shown in Figure 3.4(b). Similarly, in Figure 3.4(c), the receiver receives the third encoded block, while at the same time the Gauss-Jordan elimination is applied on the first two encoded blocks as well as their coding coefficients. As it is shown in the figure, each iteration of Gauss-Jordan elimination partially decodes the encoded blocks by reducing the coding coefficient matrix to the RREF form. After receiving the last encoded block, in Figure 3.4(d), a final iteration of Gauss-Jordan elimination recovers the original data.

Using the progressive decoding allows the process of decoding to start before receiving all linearly independent encoded blocks [48]. That is, the time of decoding process overlaps the time needed for receiving the original block. Therefore, it is hidden from the overhead caused by decoding time [48]. Since the encoding processes among all peers are done independently, it is difficult to ensure that encoded blocks received from different peers are linearly dependent. Other factors such as circles in the network topology can also impact the linear dependency. Progressive decoding enables the Progressive NC Decoder to immediately remove any linearly dependent equations, as it converts all the coefficients of a received linearly dependent row in the coefficient matrix into zeros. This signals the Progressive NC Decoder to eliminate this row from the coefficient matrix immediately after it is received which eliminates explicit checks of linear dependencies after all required blocks are received. The Progressive NC Decoder will then investigate the coefficients matrix. If it is reduced to an identity matrix, the resulted encoded blocks are equal to the original blocks without any further decoding process.

If the original data is obtained, it will be passed to the SVC Layer Manager, which prepares the video data for the video player. After a block is successfully decoded, it will be stored in the Sharing Buffer for potential upload to other peers.

We show the algorithm used at the receiver side in Figure 3.5. The receiver starts requesting from the lowest layer in the current segment scheduling window. The receiver aims at receiving the required number of encoded blocks before the deadline. Therefore, each sender is assigned by the maximum number of encoded blocks according to its upload capacity.

We show the algorithm used to allocate the maximum number of encoded blocks that can be sent from each sender. The stream consists of a series of video frames at frame rate F fps (frames per second). Each frame has a playout duration of $1/F$. The deadline of the nth

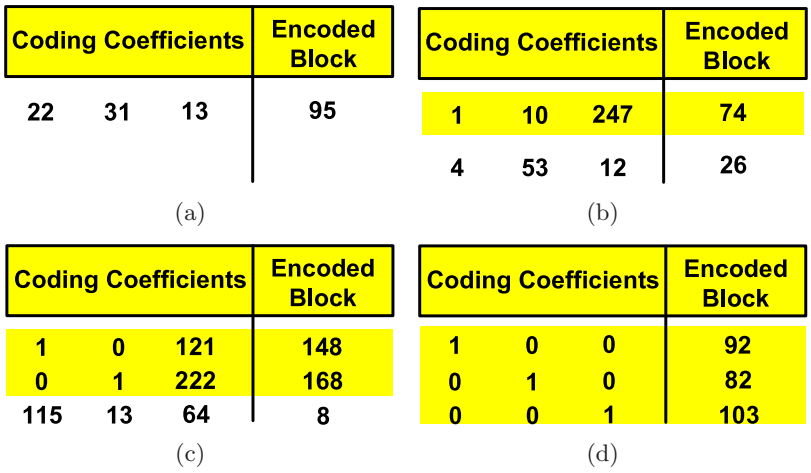


Figure 3.4: Example of progressive decoding with Gauss-Jordan elimination.

frame from the video stream is $d = (n-1)/F$. Then, the decoding deadline of segment s is defined by d_s which is the decoding deadline of the first video frame in segment s. In every iteration, the algorithm assigns to sender p ($p = 1, 2, ..., P$) the maximum number of encoded blocks which sender p can transmit within the deadline period d_s. The maximum number of encoded blocks are allocated based on two constraints: (i) the sender's remaining upload bandwidth Up_p and size of segment s defined by $Size_s$, (ii) the receiver incoming bandwidth $Down_I$. The allocation stops when the entire download bandwidth of the receiver is used.

The receiver sends a request to each sender to upload a number of encoded blocks from the desired layer. After receiving each encoded block, it goes through one pass of Gauss-Jordan elimination, and it checks if the received encoded block is linearly independent from previously received ones. After receiving all required encoded blocks, stop messages are sent to senders notifying them that there is no need for sending more encoded blocks from this layer.

3.3.2 Uploading Part

Next, we describe the uploading part of the peer. Network coding enables senders to provide receivers with partial information without needing a huge buffer map to keep availability of each partial data. The mechanism for producing such kind of partial data is as follows. Upon receiving a request at the sender side, random network coding is performed on the

Receiver Side

1. // Ps: The list of potential senders provided by tracker
2. // N_l: The number of blocks in layer l
3. // $Max_{s,p}$: Maximum number of encoded blocks from segment $s \in S$ assigned to sender $p \in Ps$
4. // Up_p: Upload bandwidth of sender p
5. // $Down_I$: Download bandwidth of the receiver
6. // s: Segment $s \in S$
7. // d_s: Deadline of segment $s \in S$
8. // $Size_s$: Size of segment $s \in S$
9. // B: Block size
10. // P: The number of potential senders
11. // L: The number of layers
12. $rec \longleftarrow 0$ // set the number of received encoded blocks to zero
13. $r_{total} \longleftarrow 0$
14. **for** $p = 1$ **to** P **do**
15. $\Delta_p = \min(Up_p d_s, Size_s)$ // Maximum number of bits allocated to sender p
16. $r_p = (\Delta_p / d_s)$ // Maximum streaming rate of sender p
17. **if** $r_{total} + r_i < Down_I$ **then**
18. $r_{total} = r_{total} + r_p$
19. $Max_{s,p} = \Delta_p / B$ // Maximum number of encoded blocks allocated to sender p
20. **else**
21. $r_p = Down_I - r_{total}$
22. $Max_{s,p} = \frac{d_s \times r_p}{B}$
23. break
24. **for** $l = 1$ **to** L **do**
25. **for** $p = 1$ **to** P **do**
26. request(Ps_p, l, s, $Max_{s,p}$) // Request for sending a number of encoded blocks
27. **while** $rec < N_l$ **do**
28. receive($ArrivedEnBlock$) // Receive one encoded block as well as its coefficients
29. decode($ArrivedEnBlock$) // Gauss-Jordan elimination
30. **if** linearlyIndependent() **then** // Check for the independent linearity
31. rec++
32. sendStopMsg(Ps) // Signal senders to stop sending further encoded blocks

Figure 3.5: The receiver side algorithm for SVC+NC system.

blocks of the requested layer. Random network coding is used because it provides robustness against frequent network topology changes, and it eliminates the need for having a centralized knowledge about the network topology [23]. Random network coding has been shown by Ho et al. [54] to be feasible in practical settings, without deterministic code assignment algorithms. Deterministic code assignment requires determining node's behaviour in a distributed manner and knowledge of the entire network topology [23, 54]. In random network coding, a peer independently and uniformly at random chooses coefficients from the Galois Field. Suppose the original data in layer l on the source node is divided into N_l blocks $[b_{1,l}, b_{2,l}, ..., b_{N_l,l}]$, where each block has a fixed size of k bytes. In random network coding, the source node chooses randomly and independently a set of M_l coding coefficients $[c_{1,l}, c_{2,l}, ..., c_{M_l,l}]$ $(M_l \leq N_l)$ over some Galois Field. It also chooses M_l blocks out of all original blocks. Then, it produces one encoded block x of size k where $x = \sum_{m=1}^{M_l} c_{m,l} \cdot b_{m,l}$. A similar process is done on intermediate nodes through the re-encoding process. In order to upload data to a receiving peer, the uploading peer chooses randomly M_l set of $(C^{m,l}, x^{m,l})$, where $C^{m,l}$ is a vector of coding coefficients corresponding to an encoded block x^m which is produced from layer l. It then generates new coding coefficient vectors $ReCoef^{m,l}$ and encoded blocks $ReBl^{m,l}$ by the following equations, where $(1 \leq m \leq M_l)$ and $(1 \leq n \leq N_l)$:

$$ReCoef^{m,l} = \sum_{m=1}^{M_l} g_m \cdot c_n^{m,l} \tag{3.1a}$$

$$ReBl^m = \sum_{m=1}^{M_l} g_m \cdot x^m \tag{3.1b}$$

Here, g_ms are chosen over some Galois Field by the uploading peer. In order to minimize linear dependency among the encoded blocks, the uploading peers select g_ms randomly and independently for each of their downloading peers. The re-encoding process at intermediate nodes is done through sparce linear coding strategy [48, 55]. This means we choose not to perform re-encoding among all received encoded blocks, but to apply on m blocks out of all received encoded blocks. Unlike complete linear coding, sparce linear coding doesn not require the re-encoding process to be done on the whole coefficient matrix. Therefore it increases the encoding rate compared to complete linear coding. It has been shown in [55,56] that sparse linear coding results in smaller computational overhead for encoding.

It has been proved that by using random network coding even with a small Galois Field

Sender Side

1. // $BufMapMsg$: Buffer map message
2. // l: Layer requested by the receiver
3. // s: segment $s \in S$
4. // $Max_{s,i}$: Maximum number of encoded blocks from segment $s \in S$ assigned to sender $i \in Ps$
5. // E_s: The number of encoded blocks generated so far for segment $s \in S$.
6. **if** peerIsActive **do**
7. **while** $E_s < Max_{s,i}$ **do**
8. **if** isStopMsg($BufMapMsg$) **then**
9. break
10. **else**
11. $EncodedBlock \longleftarrow$ encode(l) // Produce one encoded block
12. E_s++
13. sendToReceiver($EncodedBlock$)

Figure 3.6: The uploading part algorithm for SVC+NC system.

size, such as 8, the probability of selecting linearly dependent combinations is negligible [57]. In order to reduce the network coding complexity, we need to reduce the number of blocks [58]. For this purpose, in our network coding scheme, we apply network coding operations on blocks of each video layer separately. Furthermore, we use a Galois Field of size 8, as larger sizes increase the complexity while improving the results only marginally.

In Figure 3.6, we show the algorithm used at the sender side. After receiving a request, sender starts sending encoded blocks. These encoded blocks are produced by the Random NC Encoder module continuously with regards to the meximum number of encoded blocks requested by the receiver. It stops sending further encoded blocks upon receiving a stop message from the receiver.

3.4 Overhead Analysis

There are two kinds of overhead imposed by the proposed streaming system: computation and communication.

The computation overhead is imposed by the encoding and decoding processes of the network coding scheme. These processes require quadratic number of operations in terms of the number of blocks in a packet. These operations are on finite fields and thus are performed

as xor operations, which can be done efficiently by the processor. Therefore, most of the current commodity PCs can handle the encoding and decoding operations. Shojania et al. [59] show how to exploit single instruction, multiple data (SIMD) and graphics processing units (GPUs) to accelerate network coding operations. In addition, recent works [60, 61] show the potential of using GPUs and multicore CPUs to efficiently perform network coding operations.

The communication overhead is due to attaching the encoding coefficients to the encoded data blocks.

The communication overhead ratio α is given by:

$$\alpha = \frac{AdditionalBytes}{DataBytes} = \frac{NoCoefs \times NoBlocks \times CoefSize}{NoBlocks \times BlockSize} \tag{3.2}$$

In Equation 3.2, $AdditionaBytes$ is the additional bytes sent to the receiver, $DataBytes$ is the payload size, $NoCoefs$ is the number of coefficients per block, $NoBlocks$ is the number of blocks, $CoefSize$ is the size of each coefficient, and $BlockSize$ is the block size. In the proposed system, we use a Galois Field of size 256 (2^8). Therefore, the size of each coefficient is one byte.

Since each encoded block is a linear combination of the original blocks, it can be identified by its encoding coefficients. Therefore, every time a peer sends an encoded block to its downstream peer, it embeds its encoding coefficients in the header of the encoded block. Thus, the number of coefficients per block equals to the total number of blocks. Therefore, the communication overhead ratio becomes:

$$\alpha = \frac{NoBlocks}{BlockSize} \tag{3.3}$$

To assess the expected communication overhead in real systems, we need to estimate the appropriate values for the number of blocks and the size of each block. We performed three experiments to investigate how the number of blocks as well as their size affect the encoding and decoding performance. We show the results in Tables 3.2, 3.3 and 3.4 with 1 KB, 2 KB and 4 KB data in each block, respectively.

In each experiment, we vary the number of blocks from 32 to 1024 for a certain block size. The encoding time is the time required to generate a set of coding coefficients and to compute one encoded block. The decoding time is the time required to recover the original data using Gauss-Jordan elimination. We observe that by segmenting into more than 128

Table 3.2: Encoding and decoding times with 1 KB in each block.

No. of Blocks	32	64	128	256	512	1024
Encoding Time (msec)	0.5	1	2	3	7	14
Decoding Time (msec)	20	50	200	1100	5020	25100

Table 3.3: Encoding and decoding times with 2 KB in each block.

No. of Blocks	32	64	128	256	512	1024
Encoding Time (msec)	0.7	1.5	3	7	14	20
Decoding Time (msec)	30	110	500	2010	9010	40000

blocks, the decoding time grows rapidly. However, the growth of encoding times is not as considerable as decoding time. The reason is that, larger number of blocks result in longer waiting times for receiving enough encoded blocks as well as longer decoding times caused by the Gauss-Jordan elimination process. Furthermore, we observe that the number of blocks has more significant role on the decoding time compared to the block size. These results show that it is better to keep the number of blocks less than 128 to avoid introducing latencies in recovering the original data. Based on these results, we recommend using a block size of 1 KB and a total number of blocks equals to 64. In this case, the communication overhead ratio α is $64/1024 \cong 0.06$, which is a small overhead.

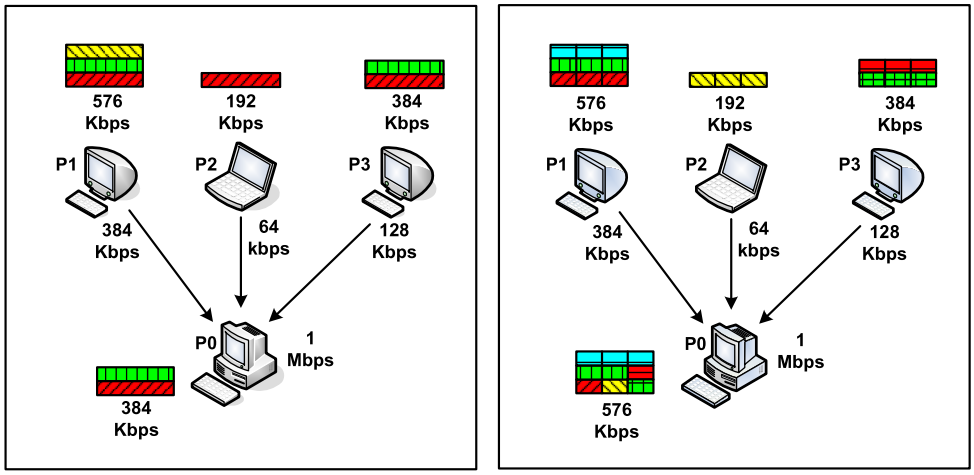
3.5 Illustrative Example

In order to show how SVC+NC streaming system is different from a traditional layered scalable peer-to-peer streaming system, we provide an illustrative comparison in Figure 3.7. In this figure, we compare the quality of the received stream in each model. There are three senders P1, P2, P3 and one receiver P0. The upload capacity of P1, P2 and P3 is 384, 64 and 128 Kbps respectively. The download capacity of P0 is 1 Mbps. Senders are assumed to have received different portions of the stream in the past. In this example, we show a layered scalable scheme, where the stream is divided into three layers. Each layer has a bit rate of 192 Kbps. In Figure 3.7(a), the receiver can get up to two layers from P1. P2 and P3 can not contribute their bandwidth due to their limited upload capacity.

Table 3.4: Encoding and decoding times with 4 KB in each block.

No. of Blocks	32	64	128	256	512	1024
Encoding Time (msec)	1	3	7	15	30	70
Decoding Time (msec)	61	210	860	4100	16020	56040

Therefore, the received video stream has a rate of 384 Kbps. As we show in Figure 3.7(a), peers cannot collaborate with each other to serve different portions of each layer. In layered scalable scheme, the coded video data is organized into network abstraction layer (NAL) units. NAL units contain an integer number of bytes representing the type of the contained data and the payload data. If one of these NAL unit packets is lost, it introduces error and impacts the decoding of the received video stream. Therefore, peers receiving incomplete layers cannot use them to enhance quality. Moreover, avoiding redundancy in transmitting data and too many allocation possibilities of different portions of the stream to different senders complicate the coordination of senders. As shown in Figure 3.7(b), in SVC+NC model, each layer is further divided into three blocks of 64 Kb. Thus, the receiver should receive three encoded blocks to recover the original ones. Each encoding process results in one encoded block of 64 Kb. Furthermore, any encoded blocks produced by any sender can be considered new and useful. The first layer is served by all three senders. The second layer is recovered by receiving two encoded blocks from P1 and one encoded block from P3. Finally, by leveraging the remaining 192 Kbps upload bandwidth of P1, all three encoded blocks of the third layer are obtained from sender P1. Therefore, each layer is served by multiple senders, results in receiving three layers of the video stream with a rate of 576 Kbps. The main reason behind such coordination between peers without exchanging any protocol messages is exploiting random network coding. By using random linear codes, each coded block is as useful as other coded blocks regardless of the uploading peer producing it [48]. Therefore, if a block is lost during transmission, it is replaced by receiving from any other neighbors. Moreover, random linear codes eliminate the problem of transmitting redundant data as well as allocating different portion of data to senders. In this figure, we do not show the nonscalable streaming scheme, which assumes that the stream has to be entirely downloaded, or it is not decodable. Thus, under the nonscalable streaming scheme, the receiver cannot get any version of the stream.



(a) SVC enabled P2P streaming (b) SVC+NC P2P streaming

Figure 3.7: An illustrative comparison between SVC enabled P2P streaming protocol and SVC+NC. In SVC+NC each layer is served by multiple peers.

Chapter 4

Experimental Evaluation

In this chapter, we first explain the setup of our experiment and define several performance metrics used in the evaluation. We then present our evaluation results of the proposed SVC+NC system, and compare them with the results of current systems that use single-layer streams and proposed systems that use either network coding alone or scalable video coding alone.

4.1 Experimental Setup

We have implemented the proposed P2P streaming system in Java. Our implementation performs all functions described in Chapter 3 and summarized in Figure 3.3. Our implementation was validated by using actual video streams.

To conduct rigorous quantitative analysis of the proposed system under wide range of working conditions, we implemented a testing application to emulate the characteristics of realistic P2P streaming systems. This testing application enables us to conduct controllable and repeatable experiments with different parameters and large number of peers. We considered deploying our system on the PlanetLab testbed and on our own local area testbed. However, these testbeds would not allow us to control important parameters such as peer upload/download bandwidth, neither would they enable us to test under high churn rates, flash crowd scenarios, and large number of *heterogeneous* peers.

The setup of our experiments is as follows. We use multiple scalable video traces obtained from the Arizona State University video library [62]. In particular, the results in this thesis are based on three video streams: a demonstration video from Sony, a clip from the Tokyo

Table 4.1: Characteristics of videos used in the experiments.

	Sony Demo	**Tokyo Olympics**	**NBC News**
Frame Rate (fps)	30	30	30
Average Frame Size (bytes)	3320	1507	1090
Average PSNR (dB)	47.6	42.7	35.5
Average Bit Rate (Kbps)	850	500	325

Olympics, and a clip from NBC News. These videos are chosen because they have diverse characteristics in their quality and bit rates. This diversity is important to assess the performance of our system in real settings. Each video is encoded in 5 scalable layers and has a frame rate of 30 fps. The frame size is CIF (352x288) and each group-of pictures (GoP) has 16 frames. We use 10 minutes of each clip in our experiments. Table 4.1 summarizes the characteristics of these video streams.

We divided each video stream into segments, where the segment size is varied. Each layer of a segment is then encoded using network coding in a number of blocks. We use different block sizes for evaluating the performance of network coding. But in any given experiment, the block size is fixed for all layers in a segment and all segments in the video. This is done to reduce the computation complexity of the network coding process, as network coding with variable block sizes is expensive. Since the video visual contents change with time, the number of blocks in a segment varies with the size of the video frames in that segment.

We create a highly-dynamic P2P streaming system with more than 1,000 heterogeneous peers that are continually changing. The upload bandwidth values of peers are chosen according to the distribution given in Table 4.2. This distribution is recommended based on actual measurement studies performed in [63]. The contributed bandwidth of each class of peers is also given as recommended in [63]. We note that peers do not contribute all their bandwidth to P2P streaming, because doing so slows down other Internet applications such as email and Web. Peers in our system can randomly fail, and they join/leave the system following different probability distributions. Each probability distribution is chosen to create a specific testing scenario such as flash crowd arrivals and high peer churn rates. More details will be given in the corresponding experiments later.

We compare the proposed system (denoted by SVC+NC in the figures) against three different systems: (i) a system that uses scalable video coding only (denoted by SVC), (ii)

Table 4.2: Upload bandwidth distribution of peers.

Fraction of Peers (%)	10.0	14.3	8.6	12.5	2.2	1.4	6.6	28.1	16.3
Total Bandwidth (Kbps)	256	320	384	448	512	640	768	1024	> 1500
Contributed Bandwidth (Kbps)	150	250	300	350	400	500	600	800	1000

a system that uses single layer video streams with network coding (denoted by SL+NC), and (iii) current systems that use single-layer, nonscalable, streams (denoted by SL).

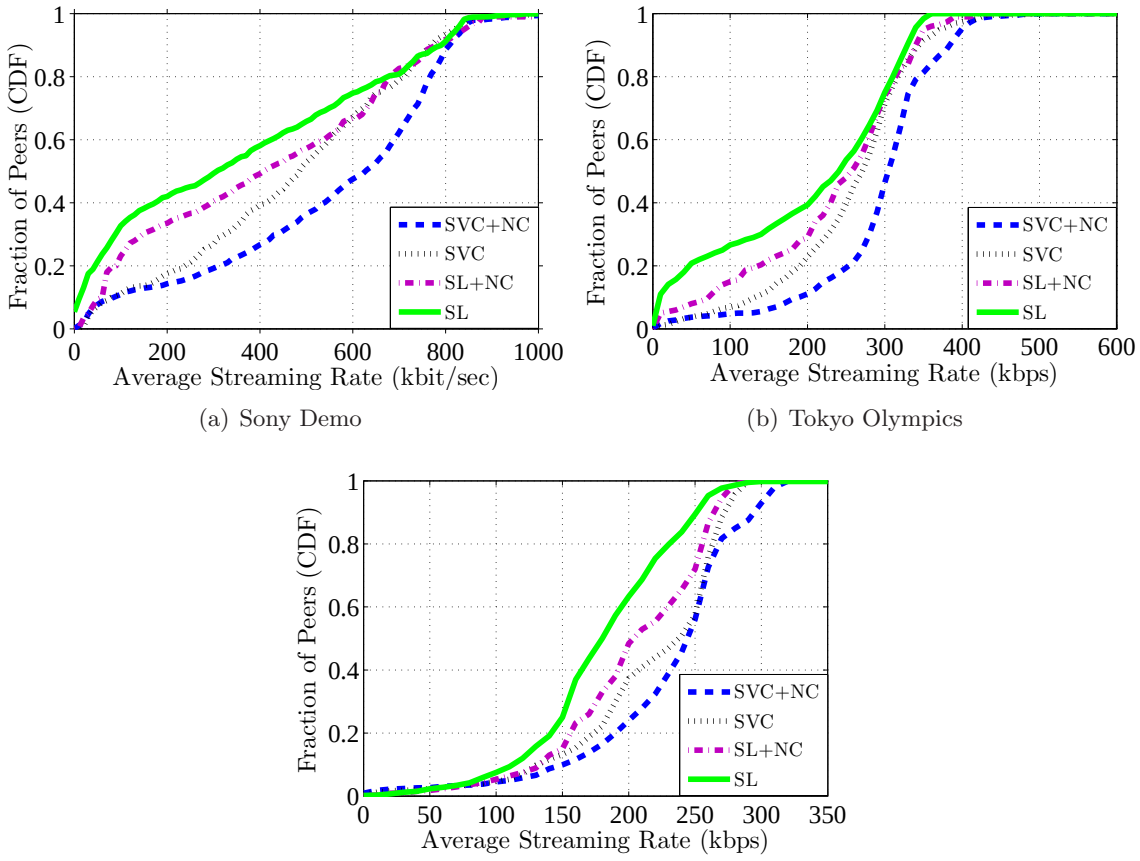
We consider several performance metrics, including: (i) average streaming rate, (ii) average streaming quality, (iii) number of streaming requests served, and (iv) fraction of late frames. These performance metrics are computed across all peers in the system, for diverse video streams, under various network conditions, and different probabilistic distributions for peer behavior. Moreover, most of these metrics are computed on a frame by frame basis and consider each layer in every frame. Furthermore, we study the impact of different system parameters on the performance and robustness of the proposed system. Therefore, our experiments are comprehensive and the results are representative of real systems.

We present the results of our extensive evaluation in the following subsections. We first present the results for the performance metrics mentioned above. Then, we present the results of analyzing the impact of several system parameters on the performance and robustness of the proposed system, especially in presence of high peer churn rates, flash crowd arrivals, and when different segment and block sizes are used.

4.1.1 Average Streaming Rate

We measure the average streaming rate during live streaming sessions. We define the streaming rate as the total number of bits of received video data per second. The average streaming rate is computed across all active peers and represents a basic performance metric.

We schedule 1,000 peers to uniformly at random join the P2P streaming system during the 10-min simulation time. We also schedule a fraction of the peers to fail or depart the system uniformly at random during the simulation time. On overage, 10% of the peers leave the system at random times. For each streaming session, a receiver is randomly chosen. Then, a group of five senders that already have the requested stream are randomly chosen



(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

Figure 4.1: Average streaming rate achieved using different systems.

to serve the receiver.

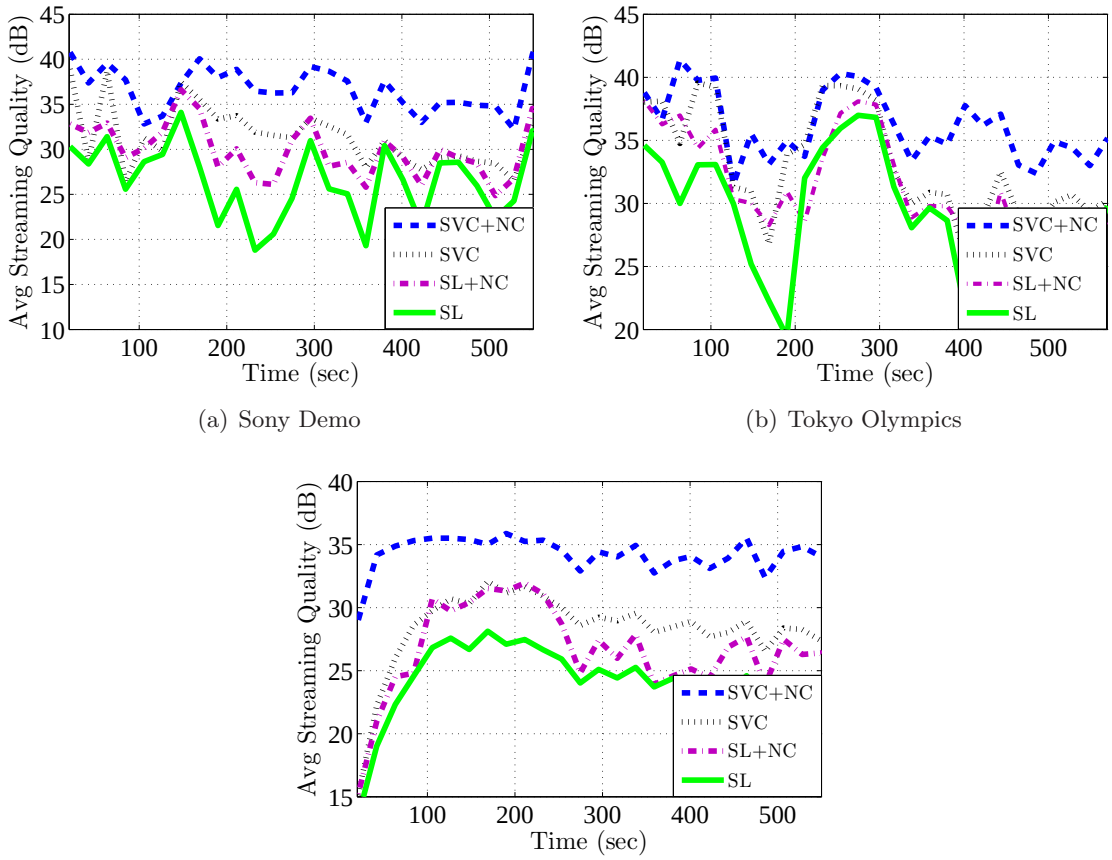
We plot the results for three different video clips in Figure 4.1 as a CDF (cumulative distribution function). The figure clearly shows that the proposed SVC+NC system outperforms the other three systems by a wide margin. For example, in the Sony Demo video, less than 18% of the peers in our SVC+NC system obtain a streaming rate of 200 kbps or less, while more than 40% of the peers in the current single-layer streaming systems receive at that low rate. On the other hand, almost 50% of the peers in our SVC+NC system receive a high streaming rate of at least 600 kbps, while this percentage is only about 30% in SVC and SL+NC, and 22% in SL systems.

4.1.2 Average Streaming Quality

Next, we consider the video quality for each active peer. Unlike the average streaming rate, which is a raw performance metric, the video quality depends on the characteristics of the video streams being served in the system and it is closer to the actual quality perceived by users. There are several methods for computing the video quality. We choose the Y-PSNR (peak signal to noise ratio of the intensity component of the video) as our video quality metric, because it is simpler to compute and interpret by readers. We acknowledge that the Y-PSNR may not always be the most accurate quality metric for all types of videos, but it is sufficient for the comparative study in this thesis.

We compute the quality as the Y-PSNR of the frames received on time divided by the total number of frames. Then we take the average among all peers and plot the results in Figure 4.2. The figure shows that the average quality in the SVC+NC system is consistently higher than that in the SVC, SL+NC, and SL systems. For example, for the NBC News video in Figure 4.2(c), the SVC+NC system yields up to 10 dB improvement in quality compared to the current single-layer streaming systems. This is a substantial gain that will definitely be felt by users and will increase their satisfaction from the P2P streaming system. Figure 4.2(c) also shows that the proposed SVC+NC outperforms the SVC and SL+NC systems by up to 5 dB, which is also a significant gain. The results for other videos indicate similar gains. In addition, the results in Figure 4.2 indicate that the video quality achieved by the SVC+NC system is more stable and smoother over time. For example, in Figure 4.2(b), there is a dramatic drop in quality for the SL system around 200 sec of the video because of the increased visual complexity of the video in this period. In contrast, the SVC+NC system did not suffer a large drop in quality.

In addition to the average video quality, we measure the fluctuations in the video quality. We show the results in Figures 4.3, 4.4 and 4.5. We quantify the fluctuations by measuring the standard deviation of the observed quality with time. Lower quality fluctuations means smoother streaming quality. For each peer, we divide the streaming time into several 20 seconds time period. For each time period, we measure the streaming quality 20 times at a specific peer, and we define *quality fluctuation* as their standard deviation. We compute the average standard deviation among all peers. For computing mean values, we take the average value of the streaming quality during 20 seconds intervals among all peers. Based on different streaming quality per 20 seconds, standard deviations are obtained. We then

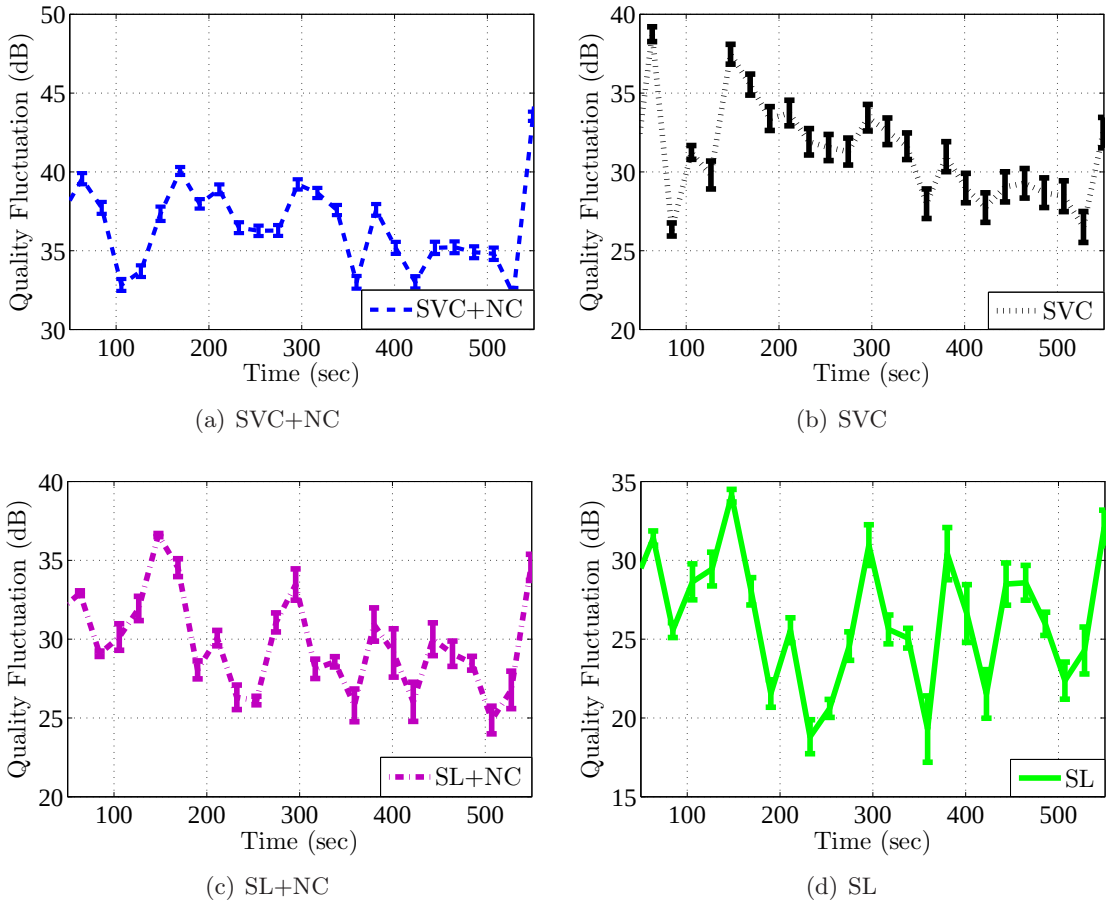


(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

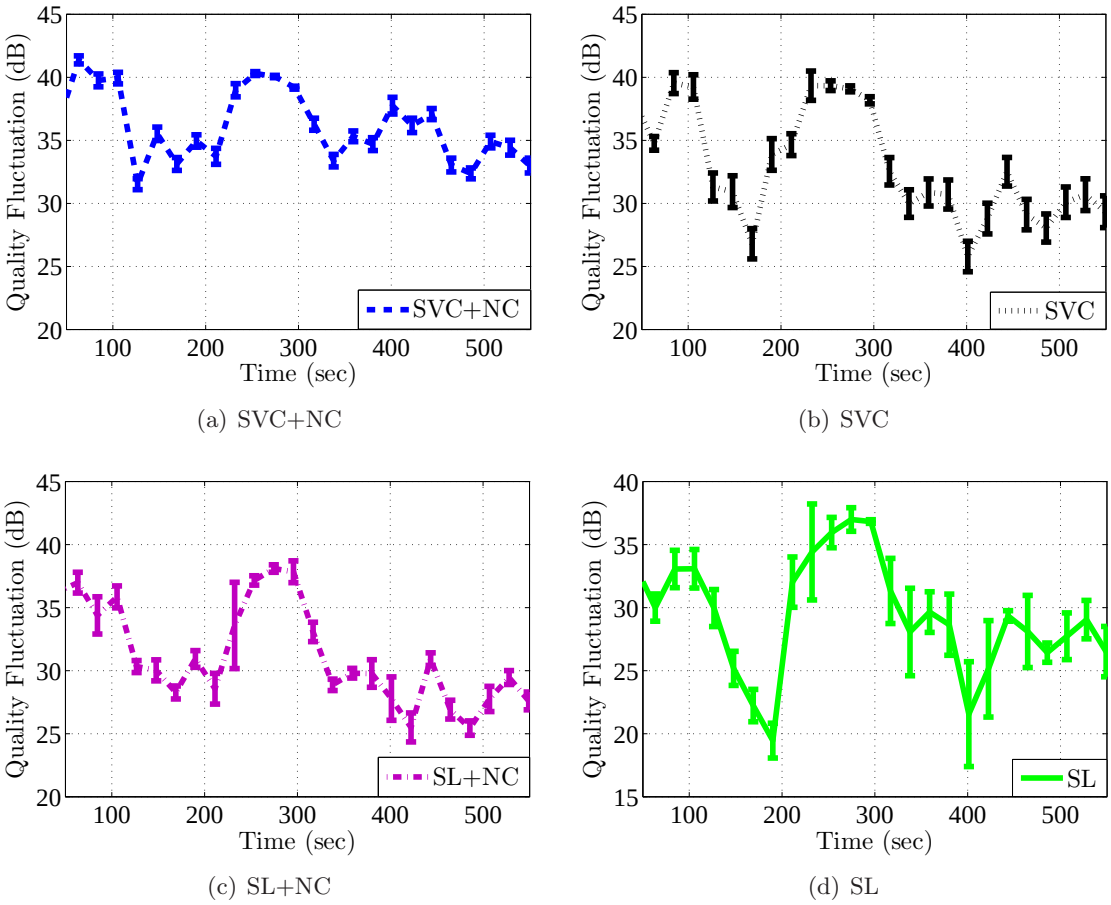
Figure 4.2: Average streaming quality achieved using different systems.



(a) SVC+NC (b) SVC

(c) SL+NC (d) SL

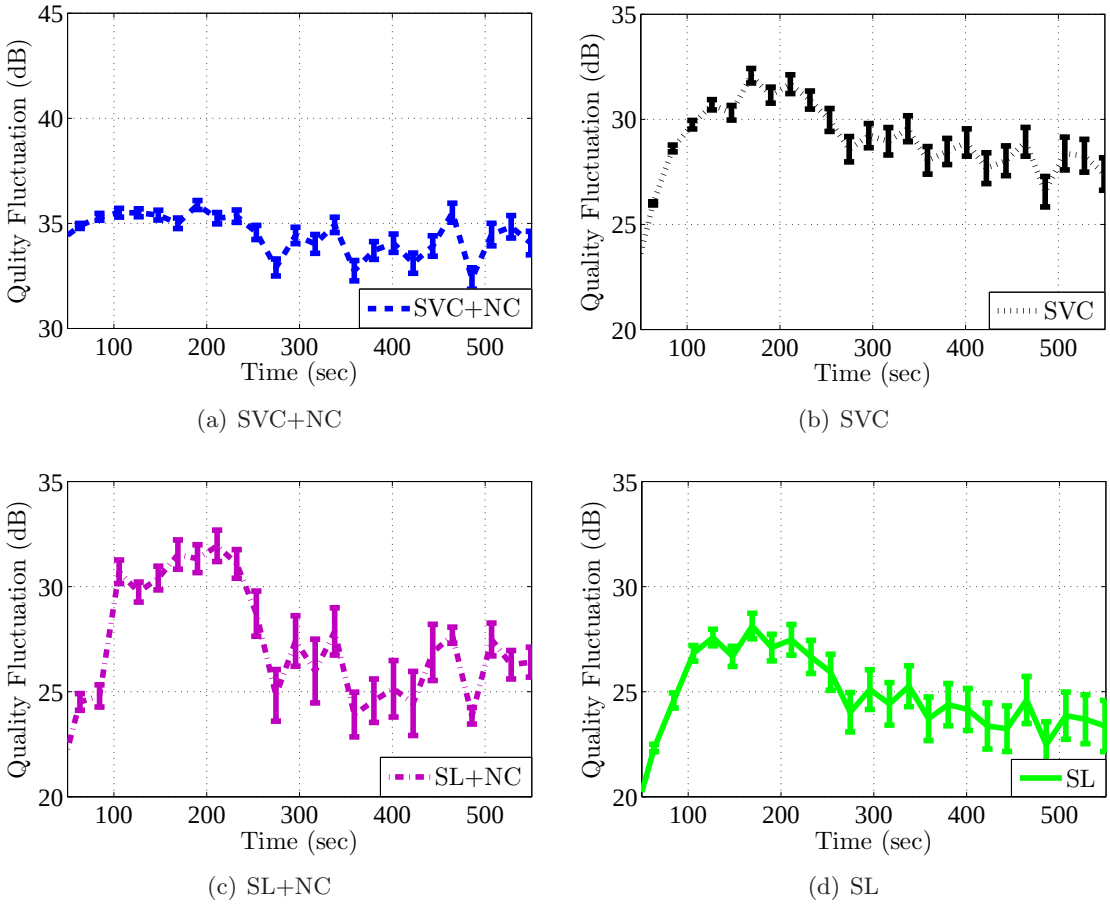
Figure 4.3: Fluctuations in video quality for Sony Demo video using different systems.



(a) SVC+NC (b) SVC

(c) SL+NC (d) SL

Figure 4.4: Fluctuations in video quality for Tokyo Olympics video using different systems.



(a) SVC+NC (b) SVC

(c) SL+NC (d) SL

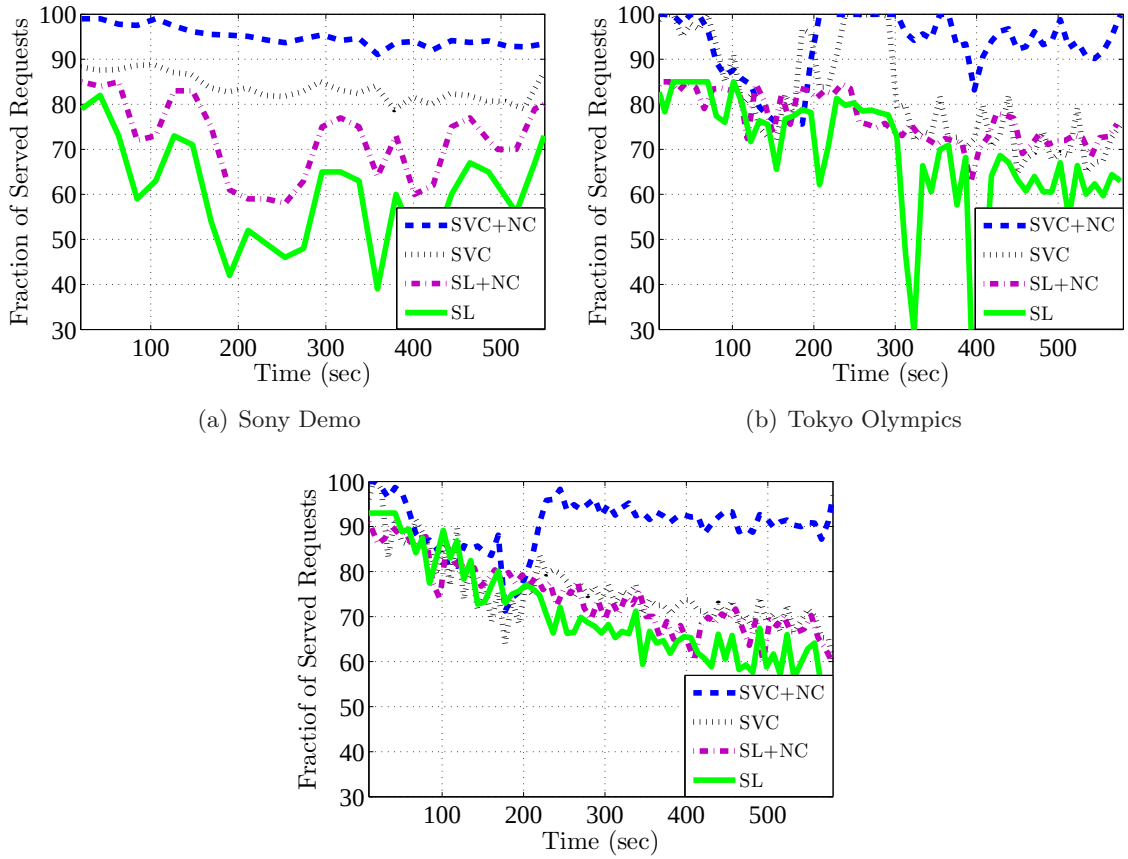
Figure 4.5: Fluctuations in video quality for NBC News video using different systems.

compute error bars per 20 seconds defined by three points: average quality minus one average standard deviation, average quality, and average quality plus one average standard deviation. Our results show that the proposed system provides much smoother quality for peers than other systems. In particular, in the proposed SVC+NC system, the average quality fluctuations per 20 seconds are about 100% less than the fluctuations observed in the current single-layer streaming systems, and about 50% less than the fluctuations in the other SL+NC and SVC systems for all three video types.

In summary, the results for the above two metrics (average streaming rate and quality) demonstrate that the proposed SVC+NC system outperforms the other systems in both raw as well as user-perceived performance metrics. The reasons for this better performance can be summarized as follows. Single-layer streaming systems are not flexible in terms of adapting the quality to the current network and peer conditions. They also do not provide optimal throughput. Therefore, they yield the worst performance. Streaming systems that use network coding with single-layer videos increase the system throughput, but do not improve the flexibility of the single layer video streams. Thus, SL+NC systems improve the performance beyond what is achievable by SL systems. On the other hand, P2P streaming systems that use scalable video streams adapt well to network and peer dynamics, but they may not fully utilize peer resources. Therefore, SVC systems also improve the performance compared to SL systems. Combining scalable video streams with network coding achieves both flexibility and increased throughput. Thus, SVC+NC systems consistently provide superior performance compared to other systems.

4.1.3 Number of Streaming Requests Served

In Figure 4.6, we plot the fraction of served requests in different streaming systems. We refer to a request as served when it is responded by neighbors and received on time by the requesting peer. We obtain the fraction of served requests by computing the number of served requests made by active peers divided by the total number of requests. We compute this fraction every 20 seconds. The results in Figure 4.6 show that the proposed SVC+NC system serves more requests than the other systems. For example, for the NBC News video in Figure 4.6(c), after 200 seconds from the beginning of the streaming session, up to 30% more requests can be served using the SVC+NC system than the SL system. Therefore, the proposed system not only provides better video quality, but also serves more requests from peers.



(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

Figure 4.6: Number of served requests for different systems.

4.1.4 Fraction of Late Frames

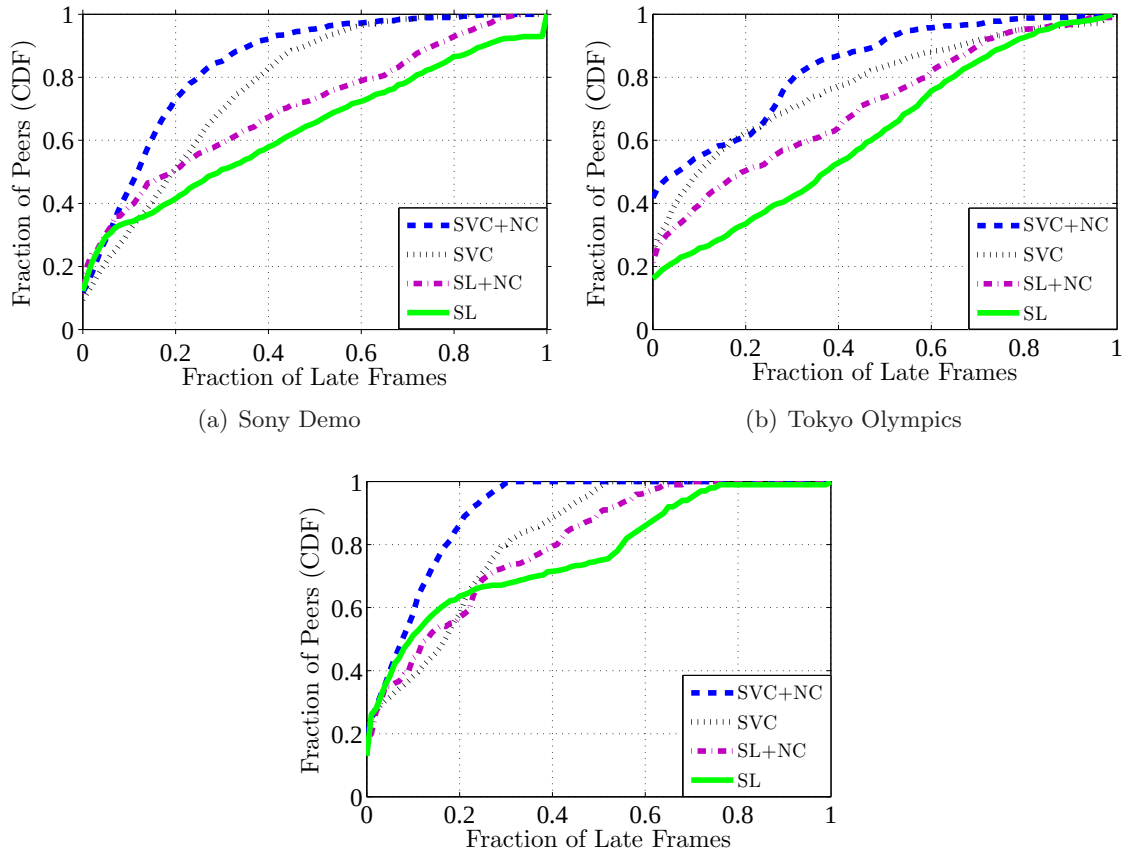
Next, we analyze the fraction of late frames for all considered streaming systems. The fraction of late frames is obtained by dividing the number of frames that arrive after their deadlines to the total number of requested frames. When a peer first joins the network, it waits for a few segments of the video according to its initial buffering delay. The initial buffering delay helps peers to receive some of the encoded blocks from future segments before the playback starts. In all experiments, we let peers wait for two segments when they join the system as recommended by [47].

We plot the CDF of the fraction of late frames in Figure 4.7. The figure shows that in the SVC+NC system, more frames meet their deadlines than in the other systems. For example, in Figure 4.7(a), in the single-layer streaming system, about 16% of the peers received more than 80% of the frames after their deadlines. While in the proposed SVC+NC system, almost no peer observed that high fraction of late frames. As another example, Figure 4.7(b) shows that the SVC+NC system achieves nearly 100% improvements over other systems in terms of the fraction of peers that observed no late frames: from about 20% of the peers in the SL, SL+NC, and SVC systems to about 40% in the SVC+NC system. Finally, Figure 4.7(c) shows that there is no peer with more than 30% of late frames, while this fraction is almost 18%, 25% and 35% in SVC, SL+NC and SL systems, respectively.

4.1.5 Impact of Churn Rate on Video Quality

We next study the impact of the churn rate on the streaming quality. In this scenario, we consider a highly dynamic peer-to-peer network with frequent arrivals and departures of peers. Maintaining a reasonable video quality in dynamic systems shows their robustness to frequent changes in network topology. In this experiment we will show that SVC+NC is more resilient and provides a more reliable peer-to-peer streaming system than the other systems.

In highly dynamic peer-to-peer systems, some peers join the system, start streaming and also contribute their resources to others. At the same time, other peers may be leaving the system, which will result in loss of upload resources and perhaps disruption of some on-going streaming sessions. We refer to the ratio of the total number of peers that join the streaming system during the simulation time to the total number of peers that leave the system as the churn rate. All arrivals and departures are scheduled according to a Poisson distribution



(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

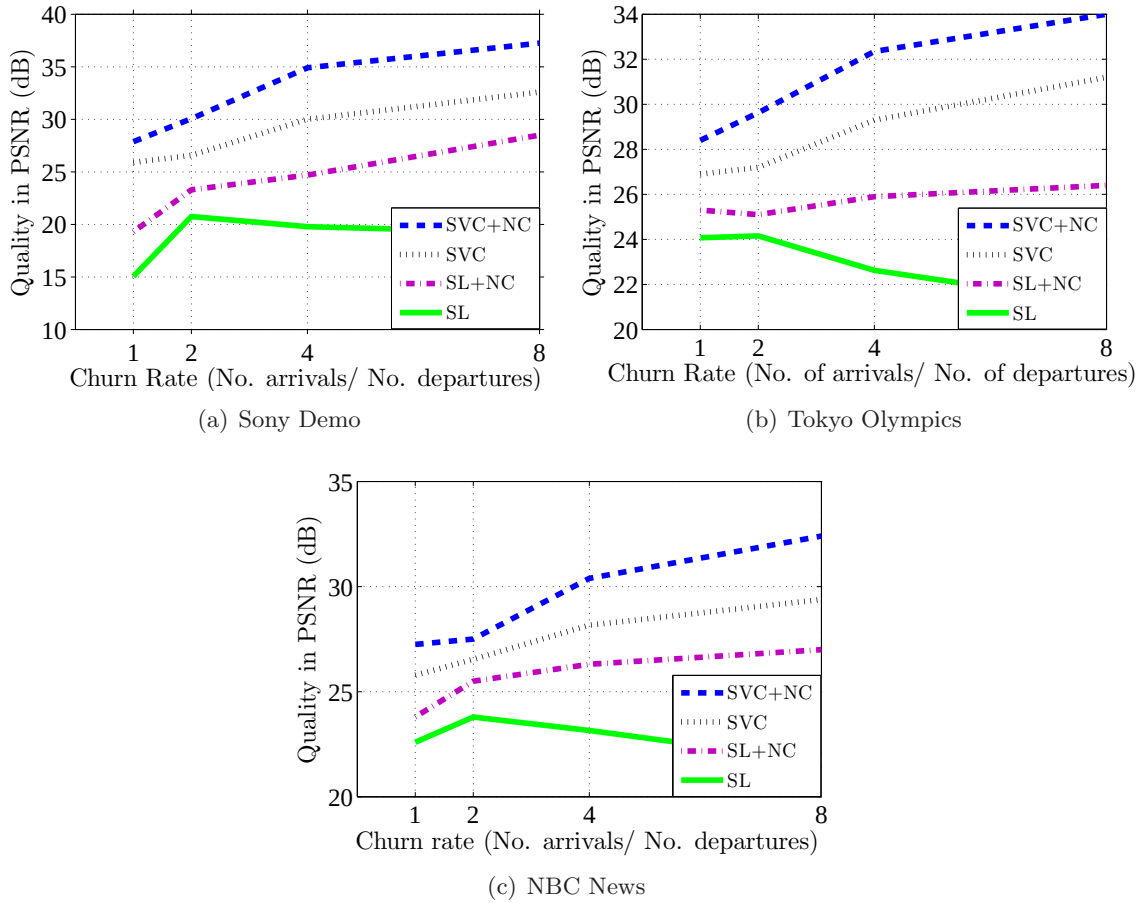
Figure 4.7: Fraction of late frames for different systems.

during the simulation time. We vary the churn rate between 1 and 8. For example, a churn rate of 2 means that if x number of peers leave the system during the simulation time, $2x$ new peers will arrive during that period. A robust P2P streaming system should utilize the resources brought by the new peers as well as provide them with good quality.

We run the experiments for the three video streams in Table 4.1 and for each streaming system: SL, SL+NC, SVC, and SVC+NC. We repeat the experiment for each considered churn rate. We measure the average quality perceived across all peers for each churn rate. We plot the results in Figure 4.8. The results confirm the superior and stable performance of the proposed SVC+NC streaming system as several dBs in quality gain are observed in all cases. The figure also shows that as more peers join, the quality for all peers improve, which is shown for high churn rates. This is because: the proposed SVC+NC system can: (i) increase the average throughput in the system since it uses network coding to harvest the resources of the new peers, and (ii) improve the quality by providing more video layers, which is enabled by the scalability nature of the video streams. We note that single-layer streaming systems may actually suffer in presence of high churn rates, as shown in Figure 4.8. This is because these systems take time to start effectively utilizing the resources of the new peers and they only provide a single version of the video streams. As the figure also shows, adding network coding to single-layer streaming systems mitigates the first problem, but not the second: the average quality provided by SL+NC systems slightly improves as more peers join the system.

4.1.6 Impact of Flash Crowd Arrivals

In flash crowd arrivals, peers join the network in a short period of time. In this case, the demand for receiving the video data may become more than the available resources. Flash crowds scenarios put a substantial stress on the P2P streaming systems that strive to provide a reasonable and sustained video quality to peers. Addressing flash crowd arrivals is important for practical systems as they often happen after the release of popular video clips. We change the average number of peer arrivals per minute from 10 to 60 with an increment of 10. Peers arrive uniformly at random during the simulation period. We allow up to 10% of the active peers to leave during the streaming sessions, which also happen at uniform random times. We measure the average quality in dB for all considered systems for each peer arrival rate. The results, shown in Figure 4.9, demonstrate that while under very high peer arrival rates the quality rendered by all systems decreases because of the

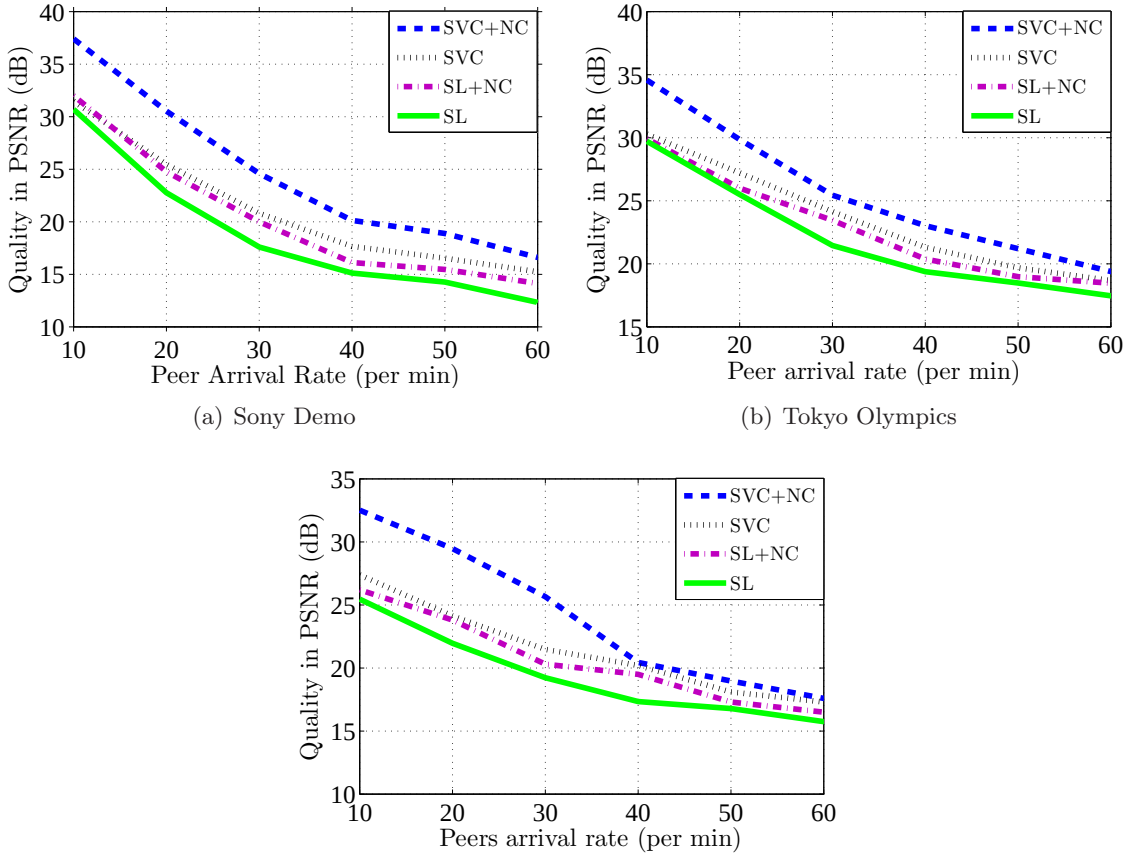


(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

Figure 4.8: Impact of churn rate on the average video streaming quality.



(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

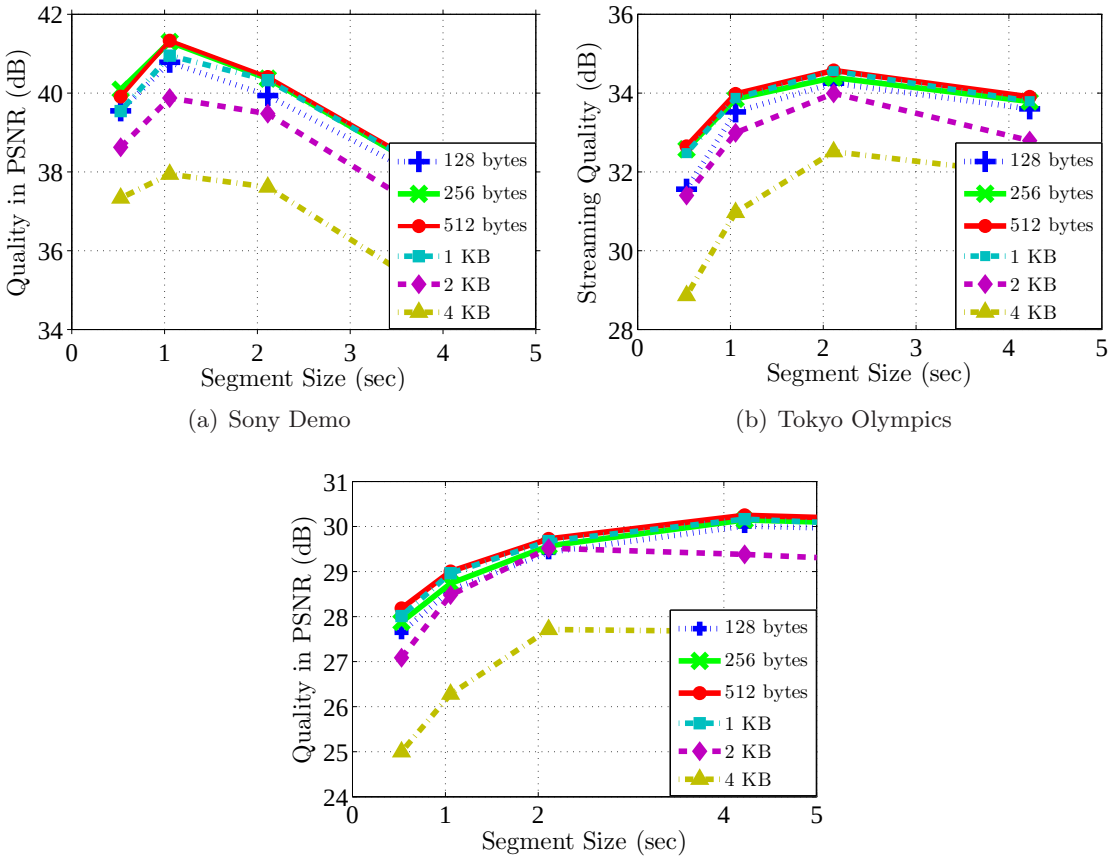
Figure 4.9: Impact of flash crowd on video streaming quality.

limited upload capacity, the SVC+NC system provides relatively better quality in flash crowd scenarios than other systems. The figure shows that there is at least 2 dB quality difference by the SVC+NC and SL systems (right lower corner of Figure 4.9(b)) and up to 7 dB (left top corner of Figure 4.9(c)).

4.1.7 Impact of Segment and Block Sizes

Finally, we investigate the effect of the segment and block sizes on the streaming quality of the proposed SVC+NC system. We vary the segment size from 0.5 to 5 sec. For each segment size, we vary the block size from 128 bytes to 4 kilobytes and we run the experiments for each considered streaming system. We measure the average streaming quality and plot the

results in Figure 4.10. A few observations can be drawn from this figure. First, decreasing the block size for network coding (up to 512 bytes) generally yields better video quality. This is because when blocks are small, a single segment will have many blocks. This allows multiple sending peers to cooperate and send different (non-redundant) encoded blocks. On the other hand, decreasing the block size below 512 bytes yields marginal or no additional benefits. The second observation is that, the ideal segment size (in sec) varies for different video streams. This is because the videos used in the experiments have diverse average bit rates of: 850, 500, 325 Kbps for the Sony Demo, Tokyo Olympics, and NBC News videos, respectively. From our experiments and the results shown in Figure 4.10, we have found that a segment should contain about 100 to 200 KB of video data. Thus, the actual segment size (in sec) will depend on the bit rate of the video stream distributed to peers. For example, a segment size of 1 sec yields the best performance for the Sony Demo video according to Figure 4.10(a). Given that the Sony Demo has an average bit rate 850 Kbps, the amount of video data in a segment is about 106 KB. Whereas a segment size of 4 sec provides the best performance for the NBC News video according to Figure 4.10(c), which means that the segment will contain about $4 \times 325/8 = 162$ KB.



(a) Sony Demo

(b) Tokyo Olympics

(c) NBC News

Figure 4.10: Impact of Segment and Block Sizes on video streaming quality.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Most of the current P2P streaming systems use nonscalable video streams and thus they provide a single version for all receivers despite their diverse resources. These systems may also suffer from suboptimal utilization of peer upload bandwidth. In this thesis, we showed that designing P2P streaming systems with scalable video coding and network coding can solve both of the above problems. We showed that the integration of the network coding *and* scalable video coding techniques improves the system performance beyond what is possible in current systems that use single-layer streams and proposed systems that use either network coding alone or scalable video coding alone. We implemented the proposed system and conducted extensive evaluation study in realistic settings and with actual scalable video traces. The evaluation study confirms the significant potential performance gain. It shows that the proposed system supports receiver heterogeneity, better utilizes the peer upload bandwidth and is robust against network and peer dynamics.

In summary, we show that the combination of network coding and scalable video coding in live P2P streaming systems is beneficial and it can achieve (i) significant improvement in the visual quality perceived by peers (several dBs are observed), (ii) smoother and more sustained streaming rates (up to 100% increase in the average streaming rate is obtained), (iii) higher streaming capacity by serving more requests from peers, and (iv) more robustness against high churn rates and flash crowd arrivals of peers.

5.2 Future Work

The work in this thesis can be extended in multiple directions. For example, we can develop an analytical model to analyze the performance of the proposed P2P live streaming system. We can also implement the proposed system as a plug-in library that can be used in other streaming systems in order to enable them to benefit from scalable video streams and network coding methods. Another possible extension is to investigate the interaction between the network coding module and other parts of the P2P streaming system. For instance, we can explore how different peer matching algorithms can affect the performace gain while we are using network coding and scalable video coding. Finally, it is interesting to study the feasibility and potential gain of using scalable video coding and network coding on mobile platforms such as iPhone.

Bibliography

[1] P. Rodriguez, S. Tan, and C. Gkantsidis. On the feasibility of commercial, legal P2P content distribution. *ACM SIGCOMM Computer Communication Review (CCR'06)*, 36(1):75–78, January 2006.

[2] Y. Tu, J. Sun, M. Hefeeda, and S. Prabhakar. An analytical study of peer-to-peer media streaming systems. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 1(4):354–376, November 2005.

[3] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai. Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution. *ACM/Springer Multimedia Systems Journal*, 11(4):383–399, April 2006.

[4] J. Liu, S. Rao, B. Li, and H.i Zhang. Opportunities and challenges of peer-to-peer Internet video broadcast. *Proceedings of the IEEE Special Issue on Recent Advances in Distributed Multimedia Communications*, 96(1):11–24, January 2008.

[5] X. Zhang, J. Liu, B. Li, and T. Yum. DONet/CoolStreaming: a data-driven overlay network for live media streaming. In *Proc. of IEEE INFOCOM'05*, pages 2102–2111, Miami, FL, March 2005.

[6] PPLive. `http://www.pplive.com/`.

[7] UUSee. `http://www.uusee.com/`.

[8] SopCast. `http://www.sopcast.com/`.

[9] TVAnts. `http://www.tvants.com/`.

[10] C. Huang, J. Li, and K. Ross. Can Internet video-on-demand be profitable? In *Proc. of ACM SIGCOMM'07*, pages 133–144, Kyoto, Japan, August 2007.

[11] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 282–297, Bolton Landing, NY, October 2003.

[12] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proc. of ACM Symposium on Operating Systems Principles (SOSP'03)*, pages 298–313, Bolton Landing, NY, October 2003.

[13] N. Magharei and R. Rejaie. Prime: peer-to-peer receiver driven mesh-based streaming. In *Proc. of IEEE INFOCOM'07*, pages 1415–1423, Anchorage, AK, May 2007.

[14] M. Zhang, L. Zhao, J. Tang, and S. Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *Proc. of ACM Multimedia'05*, pages 287–290, Singapore, November 2005.

[15] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: a comparative study of live P2P streaming approaches. In *Proc. of IEEE INFOCOM'07*, pages 1424–1432, Anchorage, AK, May 2007.

[16] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *Proc. of IEEE INFOCOM'06*, pages 1–13, Barcelona, Spain, April 2006.

[17] P. Larsson. Multicast multi-user ARQ. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC'08)*, pages 1985–1990, Las Vegas, NV, April 2008.

[18] D. Petrovic, K. Ramchandran, and J. Rabaey. Overcoming untuned radios in wireless networks with network coding. *IEEE Transactions on Information Theory*, 52(6):2649–2657, June 2006.

[19] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. of IEEE INFOCOM'05*, pages 2235–2245, Miami, FL, March 2005.

[20] O. Hillestad, A. Perkis, V. Genc, S. Murphy, and J. Murphy. Adaptive H.264/MPEG-4 SVC video over IEEE 802.16 broadband wireless networks. In *Proc. of IEEE Packet Video Workshop (PV'07)*, pages 26–35, Lausanne, Switzerland, November 2007.

[21] K. Mokhtarian and M. Hefeeda. Efficient allocation of seed servers in peer-to-peer streaming systems with scalable videos. In *Proc. of IEEE International Workshop on Quality of Service (IWQoS'09)*, pages 1–9, Charleston, SC, July 2009.

[22] S. Mirshokraie and M. Hefeeda. Live peer-to-peer streaming with scalable video coding and network coding. In *Proc. of ACM on Multimedia Systems conference (MMSys'10)*, pages 123–132, Phoenix, AZ, February 2010.

[23] P. Chou, Y. Wu, and K. Jain. Practical network coding. In *Proc. of Allerton Conference on Communication, Control, and Computing (Allerton'03)*, Monticello, IL, October 2003.

[24] C. Fragouli, J. Le Boudec, and J. Widmer. Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68, January 2006.

[25] R. Ahlswede, N. Cai, S. Li, and R. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, July 2000.

[26] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[27] BitTorrent. `http://www.bittorrent.com/`.

[28] Gnutella. `http://www.gnutella.com/`.

[29] F. Pianese, J. Keller, and E. Biersack. Pulse, a flexible P2P live streaming system. In *Proc. of IEEE INFOCOM'06*, pages 1–6, Barcelona, Spain, April 2006.

[30] PPStream. `http://www.ppstream.com/`.

[31] V. Pai, K. Tamilmani, V. Sambamurthy, K. Kumar, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS'05)*, pages 127–140, Ithaca, NY, February 2005.

[32] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 2nd edition, 1994.

[33] C. Lim and P. Lee. More flexible exponentiation with precomputation. In *Proc. of Advances in Cryptology (CRYPTO'94)*, pages 95–107, Santa Barbara, CA, August 1994.

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

[35] I. Amonou, N. Cammas, S. Kervadec, and S. Pateux. Optimized rate-distortion extraction with quality layers in the scalable extension of H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1186–1193, September 2007.

[36] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *Proc. of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, pages 162–171, Monterey, CA, June 2003.

[37] R. Rejaie and A. Ortega. PALS: peer-to-peer adaptive layered streaming. In *Proc. of ACM International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, pages 153–161, Monterey, CA, June 2003.

[38] M. Hefeeda and C. Hsu. Rate-distortion optimized streaming of fine-grained scalable video sequences. *ACM Transactions on Multimedia Computing, Communications and Applications*, 4(1):1–28, January 2008.

[39] X. Lan, N. Zheng, J. Xue, X. Wu, and B. Gao. A peer-to-peer architecture for efficient live scalable media streaming on Internet. In *Proc. of ACM Multimedia'07*, pages 783–786, Augsburg, Germany, September 2007.

[40] Y. Liu, W. Dou, and Z. Liu. Layer allocation algorithms in layered peer-to-peer streaming. In *Proc. of IFIP International Conference on Network and Parallel Computing (NPC'04)*, pages 167–174, Wuhan, china, October 2004.

[41] S. Li, R. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371, 2003.

[42] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE Transactions on Information Theory*, 11(5):782–795, October 2003.

[43] J. Park, M. Gerla, D. Lun, Y. Yi, and M. Medard. Codecast: A network-coding-based ad hoc multicast protocol. *IEEE Wireless Communications*, 13(5):76–81, October 2006.

[44] D. Nguyen, T. Nguyen, and B. Bose. Wireless broadcasting using network coding. In *Proc. of Workshop on Network Coding, Theory, and Applications (NetCod'07)*, pages 914–925, San Diego, CA, January 2007.

[45] C. Gkantsidis, J. Miller, and P. Rodriguez. Anatomy of a P2P content distribution system with network coding. In *Proc. of International Workshop on Peer-to-Peer Systems (IPTPS'06)*, Santa Barbara, CA, February 2006.

[46] S. Annapureddy, S. Guha, and C. Gkantsidis. Is high-qality VoD feasible using P2P swarming? In *Proc. of International Conference on World Wide Web (WWW'07)*, pages 903–912, Banff, Canada, May 2007.

[47] C. Feng and B. Li. On large-scale peer-to-peer streaming systems with network coding. In *Proc. of ACM Multimedia'08*, pages 269–278, Vancouver, Canada, October 2008.

[48] M. Wang and B. Li. R2: Random push with random network coding in live peer-to-peer streaming. *IEEE Journal on Selected Areas in Communications*, 25(9):1655–1666, December 2007.

[49] M. Wang and B. Li. Lava: A reality check of network coding in peer-to-peer live streaming. In *Proc. of IEEE INFOCOM'07*, pages 1082–1090, Anchorage, AK, May 2007.

[50] J. Zhao, F. Yang, Q. Zhang, Z. Zhang, and F. Zhang. Lion: Layered overlay multicast with network coding. *IEEE Transactions on Multimedia*, 8(5):1021–1032, October 2006.

[51] X. Chenguang, X. Yinlong, Z. Cheng, W. Ruizhe, and W. Qingshan. On network coding based multirate video streaming in directed networks. In *Proc. of IEEE International Conference on Performance, Computing and Communications (IPCCC'07)*, pages 332–339, New Orleans, LA, April 2007.

[52] A. Nguyen, B. Li, and F. Eliassen. Chameleon: Adaptive peer-to-peer streaming with network coding. In *Proc. of IEEE INFOCOM'10*, pages 1–9, San Diego, CA, March 2010.

[53] K. Nguyen, T. Nguyen, and S. Cheung. Peer-to-peer streaming with hierarchical network coding. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME'07)*, pages 396–399, Beijing, China, July 2007.

[54] T. Ho, M. Medard, J. Shi, and M. Effros. On randomized network coding. In *Proc. of Allerton Conference on Communication, Control, and Computing (Allerton'03)*, Monticello, IL, October 2003.

[55] G. Ma, Y. Xu, M. Lin, and Y. Xuan. A content distribution system based on sparse linear network coding. In *Proc. of Third Workshop on Network Coding (Netcod'07)*, San Diego, CA, January 2007.

[56] M. Wang and B. Li. How practical is network coding? In *Proc. of IEEE International Workshop on Quality of Service (IWQoS'06)*, pages 274–278, New Haven, CT, Jun 2006.

[57] Y. Wu, P. A. Chou, and K. Jain. A comparison of network coding and tree packing. In *Proc. of IEEE International Symposium on Information Theory (ISIT'04)*, page 143, Chicago, IL, July 2004.

[58] P. Maymounkov, N. Harvey, and D. Lun. Methods for efficient network coding. In *Proc. of Allerton Conference on Communication, Control, and Computing (Allerton'06)*, pages 482–491, Urbana, IL, September 2006.

[59] H. Shojania and B. Li. Parallelized progressive network coding with hardware acceleration. In *Proc. of IEEE International Workshop on Quality of Service (IWQoS'07)*, pages 47–55, Evanston, IL, June 2007.

[60] H. Shojania and B. Li. Pushing the envelope: Extreme network coding on the GPU. In *Proc. of International Conference on Distributed Computing Systems (ICDCS'09)*, pages 490–499, Montreal, Canada, June 2009.

[61] H. Shojania, B. Li, and X. Wang. Nuclei: GPU-accelerated many-core network coding. In *Proc. of IEEE INFOCOM'09*, pages 459–467, Rio de Janeiro, Brazil, April 2009.

[62] Video Traces Research Group, 2009. `http://trace.eas.asu.edu/h264svc/`.

[63] Z. Liu, Y. Shen, K. Ross, J. Panwar, , and Y. Wang. Substream trading: Towards an open P2P live streaming system. In *Proc. of IEEE Conference on Network Protocols (ICNP'08)*, pages 94–103, Orlando, FL, October 2008.