AN ABSTRACT OF THE DISSERTATION OF

Kevin Gatimu for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on December 13, 2019.

Title: Adaptive Streaming Methods using Dual Protocols and Reinforcement Learning

Abstract approved: _____
Ben Lee

The ubiquity of high quality video and proliferation of mobile devices has contributed to an unprecedented rise in video consumption. HTTP, in conjunction with adaptive streaming, has become the *de facto* mechanism for delivering the vast majority of video as it readily caters to heterogeneous networks and devices. This dissertation presents adaptive streaming methods using the Flexible Dual Streaming TCP-UDP Protocol (FDSP) as well as queueing theory within an Markov Decision Process (qMDP).

FDSP is an improved alternative to traditional HTTP/TCP-based streaming that combines TCP's reliability with the low latency provided by UDP, resulting in an improved quality of experience (QoE) for the viewer. On the other hand, qMDP builds on HTTP/TCP-based streaming via a queueing-theory-based state space for applying reinforcement learning towards improved QoE.

Adaptive Streaming Methods using Dual Protocols and Reinforcement Learning

by
Kevin Gatimu

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented December 13, 2019
Commencement June 2020

Doctor of Philosophy dissertation of Kevin Gatimu presented on December 13, 2019.

APPROVED:

Major Professor, representing Electrical and Computer Engineering

---

Head of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Kevin Gatimu, Author

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Ben Lee, for his tireless partnership and guidance throughout my graduate school career. I am also thankful to all my academic collaborators and colleagues for being valuable peers. Finally, special thanks to my friends and family, for propping me up over the years.

CONTRIBUTION OF AUTHORS

Research presented in this dissertation was supported in part by LG Display Corporation. In particular, Hyoung-Sik Kim, Chang-gone Kim, and Jong-Sang Baek helped administer the grant that funded the research presented in Chapter 2.

Taylor Johnson, Mohammed Sinky, and Jing Zhao contributed to the experimental study portion of Chapter 2. Along with Arul Dhamodaran, they also made significant contributions to the development of the FDSP framework that was used as the basis for several successive research papers (including the manuscripts presented in Chapters 5 and 6, and Appendices A and B).

Arul Dhamodaran and Taylor Johnson assisted with maturing FDSP from a long-held simulation environment to a physical testbed as detailed in the manuscripts in Chapters 5 and 6.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDIX FIGURES

LIST OF APPENDIX FIGURES (Continued)

LIST OF APPENDIX TABLES

Chapter 1: General Introduction

Online video consumption is rising rapidly, and is currently at almost 90 minutes per day for the average viewer [1]. The Internet landscape is shifting accordingly, with video and content delivery networks projected to account for 82% and 71% of all Internet traffic by 2021, respectively [2].

Meeting today's high video demands is made possible not only by advances in networking speeds, but also lossy compression. The H.264 codec, which is the most common video encoding standard, drastically reduces the volume of transmissible video data while offering features such as error resiliency in poor network conditions. Chapter 2 of this dissertation evaluates wireless H.264 video streaming between devices in various challenging wireless network conditions. Since wireless networks are highly prone to packet losses, our findings demonstrate the importance of performing error concealment in order to reduce video quality degradation and improve quality of experience (QoE). A common error concealment technique is spatial motion vector recovery. Chapters 3 and 4 describe *Weighted Nearest Valid Motion Vector Averaging* (WNVMVA), which is used to estimate lost motion vectors based solely on available ones.

On a broader scale, HTTP, due to its ubiquity, has become the *de facto* mechanism for video delivery across the Internet. This has led to the state-of-the-art in video streaming, i.e., *HTTP Adaptive Streaming* (HAS). The Transmission Control Protocol (TCP), HTTP's typical underlying protocol, offers several advantages, most notably reliability. However, TCP often results in high latency, which leads to rebuffering and poor QoE. On the other hand the User Datagram Protocol (UDP) is better suited towards low-latency applications, but its lack of reliability makes it highly prone to packet loss, which leads to degraded video quality. We proposed a hybrid approach called the *Flexible Dual TCP-UDP Streaming Protocol* (FDSP). It combines the reliability of TCP with the low latency characteristics of UDP. FDSP delivers the more critical parts of the video data via TCP and the rest via UDP. In addition, *Bitstream Prioritization* (BP) provides a sliding scale that is used to throttle the proportion of TCP data based on the level of network congestion. Chapters 5 shows that FDSP achieves improved latency, lower rebuffering time and less rebuffering instances

compared to TCP-based streaming, as well lower packet loss than UDP-based streaming. This is shown on a physical testbed that mimics a client-server model in a video-on-demand (VoD) service. Chapter 6 then shows FDSP's improvements to adaptive streaming compared to HAS in congested networks through higher average bitrate and a more stable client buffer.

The manuscript in chapter 7 succeeds the work in chapters 5 and 6 through an alternative view of adaptive streaming. Reinforcement learning is used to develop an adaptive streaming algorithm called *qMDP* that observes streaming metrics and learns the optimal policy for maximizing QoE. Unlike other adaptive streaming RL algorithms, qMDP analyzes client buffer characteristics via queueuing theory in order to develop a more robust reward signal.

The appendices contain manuscripts that show further improvements to FDSP. Since FDSP operates on one substream at a time, the substream lengths ultimately affect the quality of the received video. The manuscript in appendix A shows that TCP rebuffering time decreases as substream lengths increase. However, this also leads to increased rebuffering instances. , which necessitates finding the optimal substream length for the highest possible QoE. Meanwhile, appendix B shows that the extent to which UDP and TCP data streams are overlapped directly affects rebuffering and packet loss. An adaptive scheme is shown to optimize QoE.

Evaluation of Wireless High Definition Video Transmission using H.264 over WLANs

Kevin Gatimu, Taylor Johnson, Mohammed Sinky, Jing Zhao, Ben Lee, Myungchul Kim, Hyoung-Sik Kim, Chang-gone Kim, and Jong-Sang Baek

Proceedings of the 9th Annual IEEE Consumer Communications and Networking Conference (CCNC), January 14-17, 2012
https://doi.org/10.1109/CCNC.2012.6181087

Chapter 2: Evaluation of Wireless High Definition Video Transmission using H.264 over WLANs

Many challenges and limitations stand in the way of streaming high resolution video content over a wireless network. A shift towards the 60-GHz band is taking place in order to accommodate for current High Definition streaming demands. However, today's Wi-Fi networks are able to provide the necessary bandwidth with the help of compression. In this paper we evaluate the effectiveness of streaming video over wireless LANs using the H.264 codec. Our study shows that streaming HD content wirelessly over 802.11n is a viable option. However, perceptual quality of video is affected by the amount of background traffic and the presence of interfering nodes, i.e., hidden nodes.

2.1 Introduction

Wireless video transmission is an important technology for consumer electronics, such as digital television (DTV), mobile multimedia devices (e.g., smartphones, mobile video terminals, and pad/tablet devices), and video telephony. However, streaming video, especially high definition (HD) video, over bandwidth limited wireless media poses a significant challenge. Recently, technologies such as *Wireless Home Digital Interface* (WHDI) [1] and *Wireless High Definition* (WirelessHD) [2] have emerged to allow transmission of uncompressed HD video. WHDI operates at 5 GHz at a data rate of 3 Gbps with a Non-line-of-sight (NLOS) range of about 100 feet. On the other hand, WirelessHD operates in the 60 GHz spectrum at a maximum data rate of 28 Gbps, but requires Line-of-sight (LOS) that limits its range to about 30 feet. These technologies, referred to as *wireless HDMI*, are attractive replacements for HDMI cables.

Alternatively, existing 802.11 networks also make wireless transmission of video possible with the help of compression, and the ubiquity of such networks allows for a wide range of consumer applications. However, the question still remains as to how 802.11 fares with enormous demands of HD video transmission. Full HD video (1080p @60fps) encoded with H.264 using Main Profile and Level 4.2 can require a data rate of up to 50 Mbps, which may

stress a typical 802.11g network. Moreover, the crowded WiFi spectrum needs to be shared with multiple devices within carrier sense range of each other and is prone to interference from other on-going transmissions due to the hidden node effect. However, the future of 802.11 networks is bright for wireless video transmission as new strides are being made to increase their bandwidth with efforts such as 802.11ac [3] and the Wireless Gigabit Alliance (WiGig) [4].

Numerous studies have been performed to analyze video transmission of H.264 over 802.11b [5, 6, 7, 8, 9], 802.11g [10] and 802.11n [11]. Error resilient features of H.264 have been observed by way of corrupting videos with synthetic packet loss [12, 13, 14, 15]. Studies have also been performed on the proposed 802.11e standard for QoS [16, 17, 18]. However, with the exception of [9], which studied Scalable Video Coding (SVC), none of the prior efforts are based on experimentation of real testbeds. Typically, Network Simulator 2 (NS2) is the main platform used. In addition, most of these studies (with the exception of [11, 10, 18]) used videos no larger than CIF resolution (352×288), which are not consistent with current home entertainment demands. There are also 802.11n-based commercial solutions such Apple's AirPlay™[19], Intel's WiDi[20], and Cavium's WiVu™[21]. However, as with any commercial products, their measured quality is unknown.

Therefore, this paper presents our evaluation of wireless transmission of HD video over WLANs using H.264. The evaluation was performed using both a real testbed and simulation. The testbed consisted of three laptops that serve as receivers and a laptop, iPad2, and iPod Touch as a set of senders. The simulation was carried out using the *Open Evaluation Framework for Multimedia Over Networks* (OEFMON) [22], which was developed at Korea Advanced Institute of Science and Technology (KAIST) and integrates a multimedia module and a network simulator. The OEFMON tool allows us to not only study networks that are difficult to create with testbeds, but also facilitates evaluation of perceptual video quality as well as network performance to provide additional insight into the issues that take place within the network.

Our study shows that streaming HD content wirelessly over 802.11n is a viable option. However, perceptual quality of video is affected by the amount of background traffic and the presence of interfering nodes, (i.e., hidden nodes). The rest of the paper is organized as follows: Sec. 7.5 presents the experimental study using a testbed. Sec. 2.3 discusses the simulation study using OEFMON. Finally, Sec. 2.4 concludes the paper and discusses our future work.

Table 2.1: Device Specification.

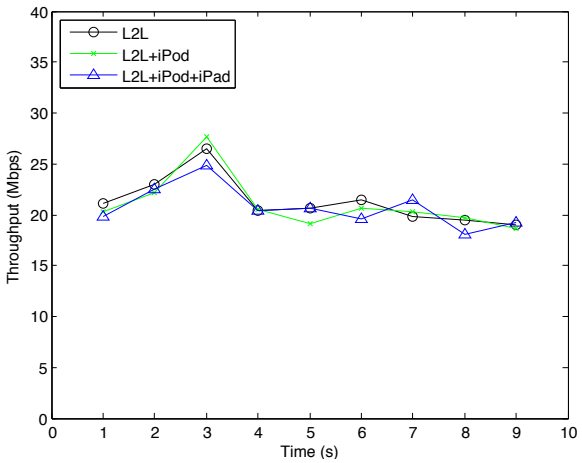
Device	Specification
Laptop1∼3	2.4 GHz Intel Core 2 Duo processor 4GB 1067 MHz DDR3 memory
Laptop4	2.5 GHz Intel Core 2 Duo processor 4 GB 667 MHz DDR2 memory
iPad2	1 GHz dual-core A5 Application processor 512 MB memory
iPod	1 GHz A4 Application processor 256 MB memory

2.2 Experimental Study

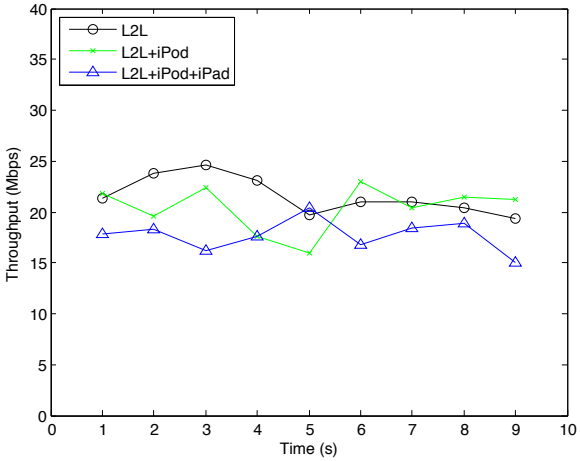
Our testbed consists of HD video streamed among six mobile devices (four laptops, an iPad2, and an iPod Touch) connected as an ad hoc network using both 802.11n and 802.11g radios on channel 9. The specifications for the devices are listed in Table 2.1. The open source *VLC Media Player* is run on all devices for streaming and playback [23]. The RTP protocol is used for streaming and network statistics are gathered using *Wireshark* [24]. The test video is a 10-second clip from *Battlefield 3* encoded with H.264 (Main Profile, L4.2) at 1080p @60fps generating an average bitrate of 20 Mbps. This video is representative of current trends in high-end video entertainment. The trailer is in fact a new video game title that will soon hit the market.

Our experiments were based on the following three configurations: (1) laptop-to-laptop stream (L2L), which serves as the primary video stream for the performance study, (2) L2L with an iPod-to-laptop interference stream (L2L+iPod), and (3) L2L+iPod with a second iPad-to-laptop interference stream (L2L+iPod+iPad). Note that all video streams are the same Battlefield clips.

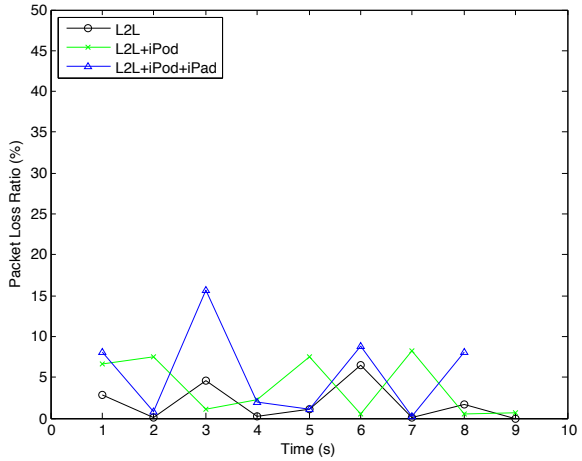
Fig. 2.1 shows the throughput, packet loss ratio (PLR), and luminance peak signal-to-noise-ratio (Y-PSNR) for both 802.11n and 802.1g from the perspective of the primary video stream (L2L) for all three configurations. Fig. 2.1a shows throughput for 802.11n, which closely matches the encoded bitrate of the corresponding video clip for all three configurations. Fig. 2.1b shows throughput for the same configurations in an 802.11g network. These results show that throughput for the primary video stream suffers with reduced bandwidth of 802.11g and becomes worse as additional interference streams are added. This can be explained by the higher maximum transmission rate available in 802.11n, which was con-



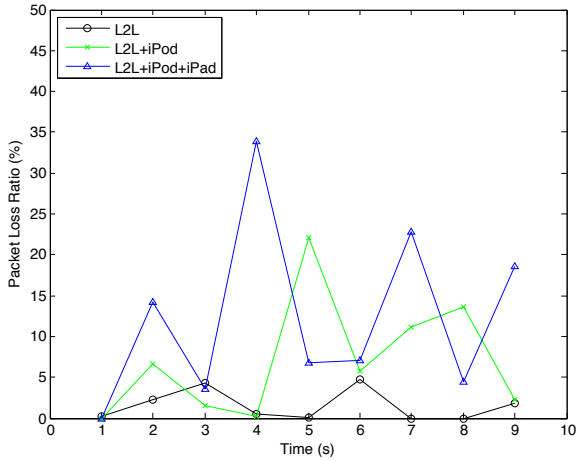
(a) Throughput for 802.11n.



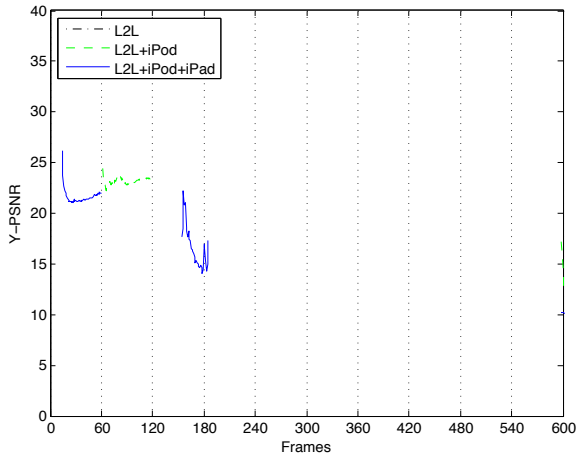
(b) Throughput for 802.11g.



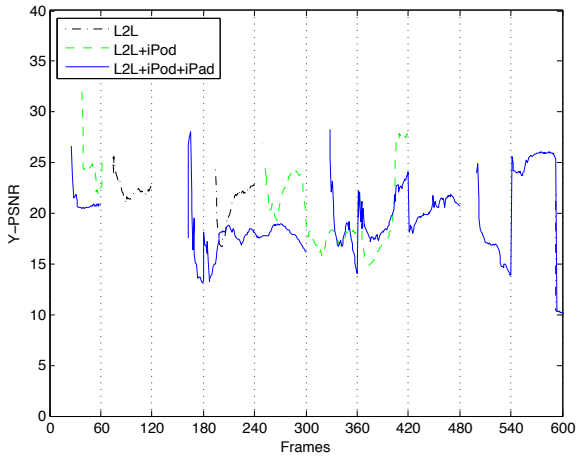
(c) Packet Loss Ratio for 802.11n.



(d) Packet Loss Ratio for 802.11g.



(e) Y-PSNR for 802.11n.



(f) Y-PSNR for 802.11g.

Figure 2.1: Experimental Results

sistently at 145 Mbps versus 54 Mbps for 802.11g. This in turn leads to higher packet loss as shown in Figs. 2.1c and 2.1d, particularly when throughput of the primary video stream reaches its peak (16% for 802.11n and 35% for 802.11g).

Figs. 2.1e and 2.1f show the quality of the received videos. Both the sent and received videos are decoded to raw YUV sequences. These graphs clearly indicate the advantage of using 802.11n for streaming full HD video. Perfect PSNR (the blank segments of the graph) is observed for large portions of the streamed video with background traffic. No distortion is observed throughout the entire video for the L2L configuration. In contrast, major degradation is experienced by the primary video stream for 802.11g, especially when both the iPod and iPad2 generate additional background traffic. The portions of degraded video come to an average Y-PSNR of 19.56 dB for the latter case. Note that I-frames occur at intervals of every 60^{th} frame, as depicted by the dotted lines in the figures, and do have a role in restoring quality to the primary stream. However, when packet losses are severe as in the case of 802.11g, this is short-lived and the quality of the video quickly degrades. For the L2L+iPod+iPad configuration, 69 frames are dropped as opposed to 15 for L2L+iPod and 9 for L2L alone. Lost frames are replaced by duplication of the previous frame, which is a typical decoder behavior.

For the most part, PLR shown in Figs. 2.1c and 2.1d coincide well with the reduction in video quality depicted in Figs. 2.1e and 2.1f, respectively. Increases in PLR cause noticeable degradation in video quality, particularly for the L2L+iPod+iPad case in 802.11g. However, for the L2L configuration in 802.11n, there is some packet loss but no degradation in Y-PSNR. This is most likely due to the nature of the video, types of packet losses, missing packets (i.e. not captured), the error concealment features used in VLC, and as with any testbed, limitations that prevent precise control over the various parameters. We are currently investigating these relationships and their effect on video quality.

2.3 Simulation Study

The simulation portion of our evaluation was performed using OEFMON [22]. OEFMON integrates the *DirectShow* multimedia framework and the *QualNet* [25] network simulator, resulting in a versatile, modular framework for evaluating video quality with respect to network performance. OEFMON requires three primary inputs. The first input is a *YUV 4:2:0 video source*. The primary video selected for this study was 300 frames (10 seconds of

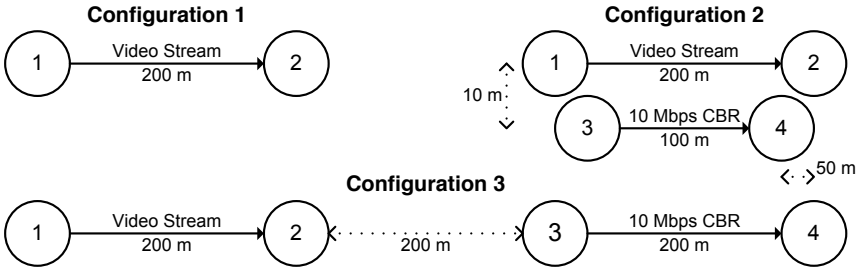


Figure 2.2: Simulated Network Scenarios

1920×1080 @30fps) from the *African Cats* trailer. The second input is a *DirectShow filter graph*, which is used to specify the encoding/transmission/decoding process that the YUV input file will go through. For this evaluation, the raw input file was encoded using the MONOGRAM H.264 encoder [26] and then passed to QualNet via the OEFMON QualNet Connector filter to undergo simulated wireless network transmission. The received packets are then passed to a decoder (for this evaluation, the CoreAVC decoder [27] was used) and then saved as a decoded YUV file. The third input is a *QualNet network configuration file*, which details node placements, types of communications between nodes, and other various network settings such as network type and link speed.

Three network configurations were constructed for the purposes of this simulation study. Configuration 1 represents a typical single source, single destination scenario where one device streams the primary video to another device via an ad-hoc network. Configuration 2 represents a dual source, dual destination scenario where two videos are being streamed simultaneously. The primary video is streamed from Device 1 to Device 2, as before, but a secondary video is streamed from Device 3 to Device 4. The secondary video is a 10 Mbps of CBR data representing a video encoded at H.264 Level 3.1 (1280×720 @30fps). Configuration 3 repeats the network traffic of Configuration 2, but the nodes are now positioned in a classical hidden-node arrangement.

Fig. 2.2 shows the placement of devices for all three configurations implemented as ad-hoc 802.11g (QualNet does not currently support 802.11n) networks with a link speed of 18 Mbps.

Fig. 2.3 shows the performance from the perspective of the primary video stream in terms of end-to-end delay, throughput, PLR, and Y-PSNR for all three configurations. Fig. 2.3a shows that there is an increase in delay from Configuration 1 to Configuration 2, and a

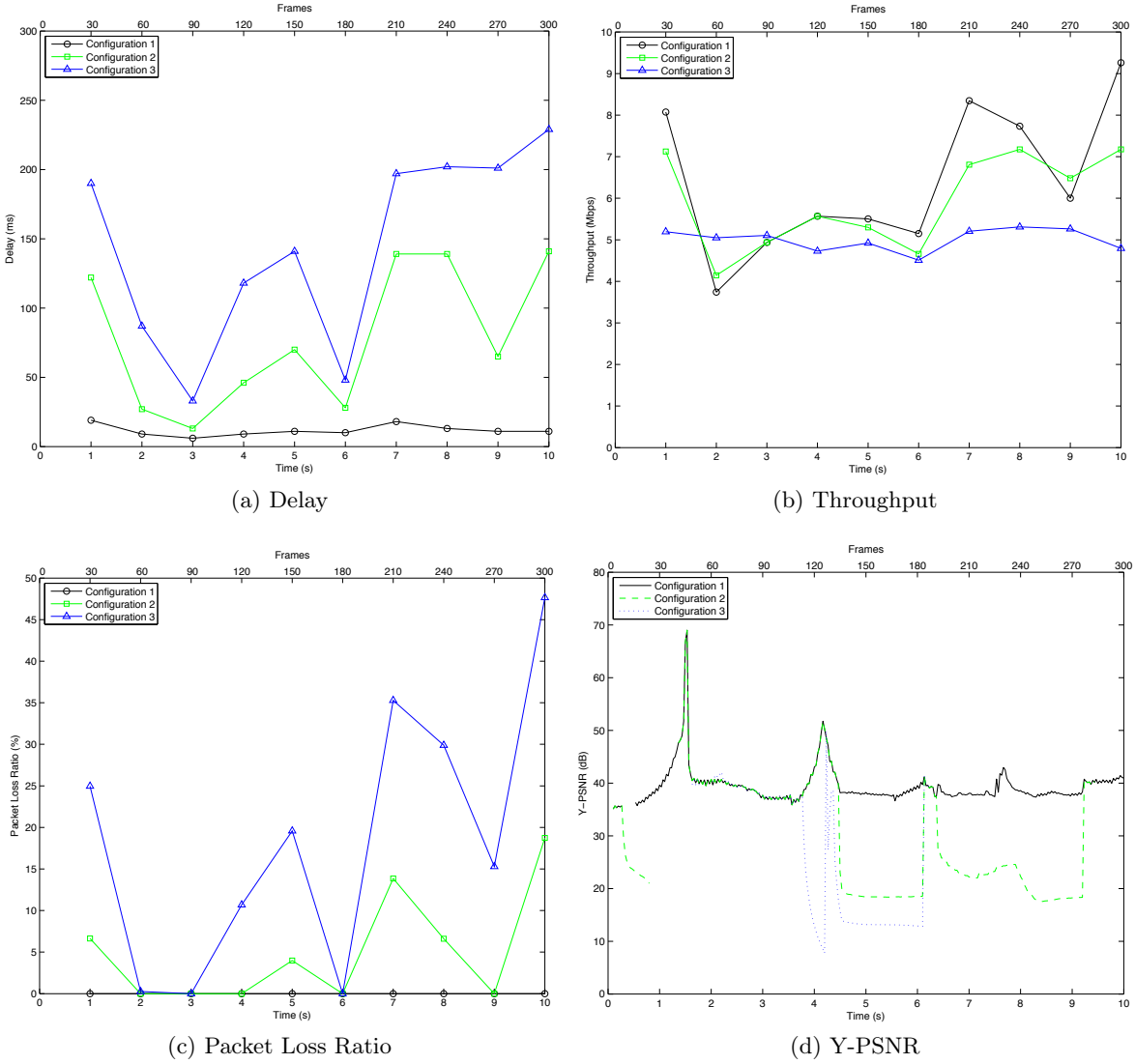
further increase in delay from Configuration 2 to Configuration 3. The former can be attributed to a significant increase in average time spent by packets in the outbound queue. This is a direct result of competition to seize the wireless medium between the primary video stream and the CBR stream. The latter is also due to increased average time spent by packets in the outbound queue. However, in this case the extra delay is due to increased packet retransmissions due to ACK timeouts at the MAC layer, caused by packet collisions resulting from the hidden-node effect.

Fig. 2.3b shows that the throughput for Configuration 1 is the highest among the three configurations and serves as an indication of the amount of data the video transmission generates when unconstrained. Fig. 2.3b also shows that in Configurations 2 and 3, the network is unable to meet the throughput demands of the video source due to severe network congestion and hidden-node-induced collisions, respectively. Although there are a couple data points where Configurations 2 and/or 3 appear to achieve a higher throughput than Configuration 1, this is simply a side effect of the whole-second averaging process used by OEFMON to generate performance results. When averaged over the entire 10 seconds, the throughput of Configuration 1 is clearly higher than Configuration 2, which in turn achieves a higher throughput than Configuration 3.

Fig. 2.3c shows that Configuration 1 exhibited no packet loss while Configuration 2 exhibited minor packet loss during parts of the transmission, sometimes even leading to lost frames. On the other hand, Configuration 3 resulted in major packet loss. These packet loss ratio results also relate directly to the throughput results. Figs. 2.3c and 2.3b together clearly show that the larger the difference between throughput demand and achieved throughput, the larger the percentage of packets lost. Additional packet losses in Configuration 3 occur due to hidden-node collisions.

Occasionally, there are no packet losses in the congestion scenario, or even in the hidden-node scenario. This is a result of the bursty characteristics of CBR traffic generation used in QualNet. In our simulations, CBR data is generated as a 2500 byte item being sent every 2 ms, which yields an effective bitrate of 10 Mbps. If a 2500 byte item is successfully sent relatively early within its 2 ms interval, then the primary video stream will have uncontested use of the wireless medium for the rest of the 2 ms. This is a limitation of using CBR to represent a second, background video stream.

Fig. 2.3d shows the Y-PSNR results for all three configurations. These results are best interpreted in terms of packet loss. When Figs. 2.3c and 2.3d are considered together,



(a) Delay

(b) Throughput

(c) Packet Loss Ratio

(d) Y-PSNR

Figure 2.3: Simulation results

there is a direct correlation between packet loss and degradation in user-perceived quality. For packet loss of 0∼15%, there is a significant decrease in user-perceived quality. Fig. 2.4 shows the received frame 24 for Configurations 1 and 2 (frame 24 for Configuration 3 was lost entirely and thus not shown). For Configuration 1, this received frame has a Y-PSNR of

(a) Configuration 1 (b) Configuration 2

Figure 2.4: Received frame number 24.

37.8 dB, with a corresponding high user-perceived quality. For Configuration 2, this received frame has a Y-PSNR of 21 dB, with a corresponding low user-perceived quality. The low quality of this Configuration 2 frame is due to the loss of AC component and motion vector data for many macro blocks from the previous several frames. The exact PLR values that lead to entire frames being lost is mostly a function of a specific decoder's implementation. For the CoreAVC decoder, our results show that when packet loss is more than 15%, entire frames are lost. As an example of this relationship, Fig. 2.3d shows that frames 188∼300 are lost for Configuration 3, while in Fig. 2.3c, frames 188∼300 correspond to PLRs that are consistently above 15%.

2.4 Conclusion and Future Work

The results of our experimental and simulation studies show that, while 802.11g has fundamental bandwidth limitations that prevent successful wireless transmission of multiple HD videos, 802.11n with H.264 continues to be a viable method for wireless streaming of HD video. This conclusion is supported by the recent increase in popularity of consumer devices that utilize 802.11n to facilitate wireless video transmission. However, there are still some situations which can have a diminishing effect on 802.11n's ability to provide wireless video content, specifically severe network congestion and hidden-node scenarios.

Our future plan is to continue developing and expanding our wireless video transmission evaluation toolset, with the goal of researching and evaluating new techniques in error resiliency, error concealment, and MAC-layer optimizations, all in order to make H.264 over

802.11n more robust and efficient.

2.5 Acknowledgements

This research was supported in part by LG Display Co., Korea.

Weighted Nearest Valid Motion Vector Averaging for Spatial Motion Vector Recovery in Wireless HD Video Transmission using H.264 over WLANs

Kevin Gatimu, Ben Lee, Tae-Wook Lee, Chang-Gone Kim, and Jone-Keun Shin

Proceedings of the Fourth International Conference on the Network of the Future (NoF),
Pohang, South Korea, October 23-25, 2013
https://doi.org/10.1109/NOF.2013.6724519

Chapter 3: Weighted Nearest Valid Motion Vector Averaging for Spatial Motion Vector Recovery in Wireless HD Video Transmission using H.264 over WLANs

High definition (HD) video is the standard of choice for today's video demands. HD video is characterized by high data rates, but it can be compressed using H.264 and transmitted over 802.11 wireless networks. However, such networks are prone to packet losses, which result in degraded perceptual video quality. It is thus important to perform error concealment. Spatial motion vector recovery is a key error concealment technique. This paper proposes a new spatial motion vector recovery technique called Weighted Nearest Valid Motion Vector Averaging (WNVMVA), which uses properly decoded motion vectors to estimate lost ones.

3.1 Introduction

Wireless HD video transmission (WHDVT) is an important technology for mobile devices and setting up highly customizable systems, such as home entertainment. However, WHDVT presents many challenges because it is bandwidth-intensive. A full HD video (1080p at 60 fps) compressed using H.264 Main Profile @Level 4.2 would require a data rate of up to 50 Mbps [1]. Currently, 802.11 wireless technology provides a ubiquitous and relatively inexpensive solution for WHDVT. For example, 802.11n transmits up to 150 Mbps per stream and up to 600 Mbps for four MIMO streams [2]. However, WHDVT still poses challenges for 802.11n, especially with multiple streams on the same channel. Moreover, 802.11n transmission is prone to interference from other devices operating at 2.4 GHz or 5 GHz. In spite of these limitations, 802.11 is still very promising and there are on-going efforts to increase its throughput, i.e. 802.11ac [3]. Also, *WiGig* [4] is backward compatible with 802.11, and promises to deliver a maximum data rate of 7 Gbps.

Although lost network packets lead to distorted video, *error concealment* (EC) can be used to minimize the perceived visual distortion of received video. EC involves reconstructing lost video information using received data by utilizing redundancy in the spatial and temporal domains.

This paper proposes a new spatial EC technique called *Weighted Nearest Valid Motion Vector Averaging* (WNVMVA), which performs motion vector (MV) recovery by exploiting spatial relationships among *valid* (i.e., properly received and decoded) MVs. This is different from existing EC techniques, such as zero MV, mean MV, median MV and last MV, which utilize estimated MVs in addition to valid ones. This leads to approximation errors and error propagation due to estimating lost MVs based on previously estimated MVs. The proposed method reduces these effects by estimating lost MVs based exclusively on valid MVs. WNVMVA is combined with pre-existing techniques into an aggregate method, which will be referred to as *Competitive MV Recovery* (CMVR), to provide multiple candidates for estimating each lost MV. Our experimental results show that CMVR in many cases outperforms CMVR without WNVMVA, which is the default EC method in our decoder of choice FFmpeg [5]. Furthermore, WNVMVA outperforms frame copy, which is very basic yet is commonly used to conceal slice-based losses in popular media players such as *QuickTime* and *Windows Media Player*.

3.2 Background

This section provides the necessary background to better understand the motivation behind the proposed method.

3.2.1 H.264

H.264 compression is done via *intra-prediction* and *inter-prediction* [1]. Intra-prediction uses samples (i.e., contiguous groups of pixels) from the current frame as references and results in intra-macroblocks (I-MBs), which make up I-frames. Inter-prediction uses samples from past and/or future frames as references and results in P-MBs and B-MBs, which make up P- and B-frames, respectively. For every block that is to be inter-predicted, *motion estimation* is used to search for the best-match block in the previous frame(s). An offset between the current block and the best-match block is measured. This offset is called a *motion vector* (MV). The difference between the actual and inter-predicted block is *prediction error*. Both the MV and the prediction error are encoded, which is more efficient than encoding a complete image block. Sometimes the prediction error is too large for efficient coding, e.g., complex movement (stretching, contortion, etc.) and when a frame's contents at the

(a) Original frame.

(b) Corrupted frame (slices 2 and 4 are lost).

(c) EC via frame copy.

(d) Error propagation 5 frames later.

Figure 3.1: Portion of original frame vs. corrupted frame (slices 2 and 4 are lost). Poor EC via frame copy results in error propagation five frames later.

edges lack suitable references in past frames. In these situations, an H.264 encoder performs intra-coding rather than inter-coding resulting in some I-MBs within both P-frames and B-frames.

When an MV is lost, its corresponding block cannot be properly decoded. Furthermore, this usually results in the absence of an accurate reference for a future inter-predicted block. This degradation process continues until an intra-coded block is encountered. This phenomenon is called *error propagation*. Therefore, in order to conceal inter-prediction errors within a particular frame and reduce error propagation, lost MVs need to be estimated. *Spatial MV recovery* is a type of EC that estimates lost MVs by inferring from data within the current frame.

3.2.2 Packet Loss and Its Effect on Video

H.264 data is coded as packets called *Network Abstraction Layer (NAL) Units*, which readily adapt well to different networks. Lost packets directly correspond to lost NAL Units and result in missing slices or parts of slices. A loss of an entire slice may occur due to severe packet loss or loss of packets containing slice header NAL Units. Figs. 4.3a and 4.3b show a comparison between part of an original frame and the same frame with slices 2 and 4 missing due to packet losses. A simple form of EC can be performed via *frame copy* as shown in Fig. 4.3c, which involves simply copying video data from a previous frame by using a zero MV for motion compensation. As can be seen, this is not visually sufficient and also

results in significant error propagation as shown in Fig. 4.3d. Therefore, more accurate EC techniques are needed, for instance, spatial MV recovery using spatial relationships that exist among MVs.

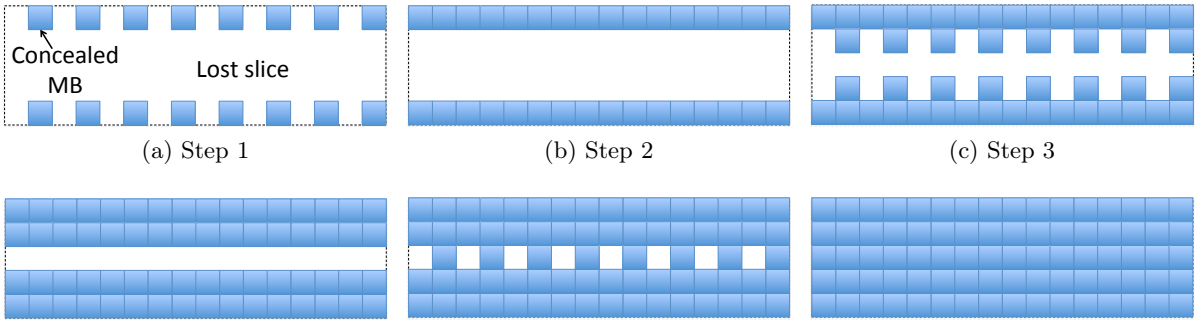
3.2.3 Error Concealment Techniques

This section discusses the default H.264 EC techniques found in *FFmpeg*, the underlying software of *VLC Media Player*, which was the media player of choice for our previous work [6]. The FFmpeg H.264 decoder applies EC through spatial mean MV, spatial median MV, last MV, and zero MV [5]. The *Mean MV* averages the surrounding MVs to recover a lost MV, while *Median MV* uses the median instead. *Last MV* copies the MV from the same spatial location of the previous inter-predicted reference frame, while *Zero MV*, also known as frame copy, uses a zero-valued MV.

Each EC technique generates an estimated MV and resultant MB. Each estimated MB is compared against neighboring MBs via a *Boundary Matching Algorithm* (BMA) that calculates the total absolute pixel value difference between the edge pixels of the estimated MB and the adjacent pixels of surrounding MBs. The estimated MB that yields the least absolute difference is picked as the final estimated MB.

FFmpeg EC is based on available surrounding MVs, which are either properly decoded or previously lost and estimated. These surrounding MVs will be referred to as *MV estimators*. Each lost MV is surrounded by a maximum of four orthogonal MV estimators. If a neighboring MB is intra-coded, then its corresponding MV estimator is set to zero. The MV estimators are then used to generate mean MV and median MV.

Lost MBs are concealed in a convergent manner as shown in Fig. 4.4, which illustrates a simplified lost slice scenario. The outer lost MBs are concealed first as they are initially the only ones with neighboring MBs. This is followed by the next inner row of MBs and the process is repeated until the whole slice is completely concealed. Also, for each row of lost MBs, the even-entry MBs are concealed first, followed by the odd-entry MBs. The even-entry MBs are then re-concealed considering the new MV estimators provided by the recently concealed odd-entry MBs. Similarly, the odd-entry MBs are then re-concealed. This is repeated for each row until the estimated MVs do not change through the iterations or for up to 10 iterations, whichever comes first.



(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5 (f) Step 6

Figure 3.2: FFmpeg error concealment MB sequence.

3.3 Related Work

Our EC method is spatial in nature and thus orthogonal to temporal methods [7, 8, 9, 10]. Spatial techniques utilize the redundancy provided by surrounding MVs or pixel values for EC. In the case of using MVs, this may include estimating the lost MV via the median or mean of surrounding MVs, interpolation of missing MVs using surrounding ones, or some variation of these. The FFmpeg decoder [5] utilizes mean and median MVs. However, its EC algorithm tends to default to frame copy or *weighted pixel averaging* (WPA) [11], depending on the decoder settings. In WPA, each lost pixel is concealed by using a distance-weighted average of the four orthogonally closest pixels. WPA is based on the assumption of spatial continuity and produces blurry distortion.

Although WPA is a common spatial EC technique, it does not utilize MVs. For this reason it is more suitable for I-frames. In fact, spatial EC methods are commonly applied towards I-frames while our work is applied towards inter-frames. For instance, Kim *et al.* proposed directional interpolation (DI) for damaged MBs based on the prediction modes of neighboring intra-MBs [12]. Similarly, Nemethova *et al.* used spatial interpolation with edge preservation and smoothing based on two or four neighboring blocks relative to the missing one [13]. Jin-wang *et al.* combined WPA and DI into an algorithm that adaptively switches between the two [14]. This not only preserves existing edges but also avoids introduction of significantly erroneous ones. WPA is also improved in [15], where multi-directional interpolation is used rather than just bilinear interpolation.

Most of the existing work in MV recovery, and EC in general, is limited to sub-HD

videos, which do not present the challenges associated with HD videos such as high data rates. Furthermore, packet losses in sub-HD H.264 video often result in whole frame losses due to low bitrates. However, packet losses in WHDVT often result in partially lost frames, making it necessary to utilize incomplete frames for EC.

3.4 The Proposed Method

The goal of WNVMVA is to exploit the spatial relationships among MVs by using only valid MVs, i.e., only properly decoded MVs that are spatially closest to the lost MVs. Note that MVs could be invalid due to intra-coding or packet loss. The spatially closest valid MVs are then used to estimate lost MVs. This is in contrast to the default methods, which use both estimated and valid MVs. By analyzing only valid MVs, a more accurate spatial relationship among MVs can be exposed.

Fig. 4.5 illustrates WNVMVA, where for each lost MV, one valid MV above (MV_1) and another below (MV_2) are initially selected based on the availability of properly decoded MVs. The distance between $MV_{1/2}$ and the lost MV is $d_{1/2}$. Note that the MVs between the lost MV and $MV_{1/2}$ are assumed to be invalid because either they are intra-MBs or lost.

The corresponding valid MVs in the horizontal direction are usually unavailable because lost H.264 visual information stretches across the whole frame as illustrated in Fig. 4.3b. However, some of the top and/or bottom rows of lost slice regions could start or end in the middle of a frame, which would result in a limited number of horizontally available valid MVs. For example, a typical frame in a HD video contains 8 slices and each slice consists of 8 or 9 rows of MBs. If one of these slices is lost and begins or ends in mid-frame, then only about half a row of lost MVs (less than 10% of the total lost MVs in a slice) would have horizontally neighboring valid MVs. This attribute is not considered in WNVMVA, but will be considered as part of our future work. Therefore, WNVMVA assumes that in general there are no available valid MVs in the horizontal direction from the perspective of a lost MV.

As shown in Fig. 4.5, once MV_1 and MV_2 are established, their respective orthogonal neighboring valid MVs, if available, are then determined. The nearest *valid* neighbors for $MV_{1/2}$ are the left ($MV_{1L/2L}$), right ($MV_{1R/2R}$) and the top/bottom ($MV_{1T/2B}$) neighbors. Again, any MVs that may exist between MV_1, or MV_2, and its nearest valid neighbor, are assumed to be invalid due to intra-coding or packet loss.

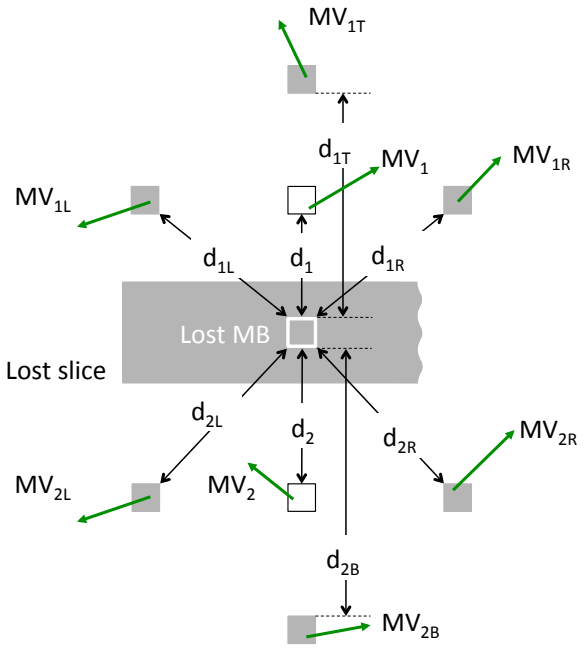


Figure 3.3: MV_1 and MV_2 and their neighbors are used to estimate the value of the lost MV via inverse distance weighting.

The orthogonal neighboring valid MVs of MV_1 and MV_2 provide complimentary spatial dependence information that is useful in more accurately estimating the lost MV. Furthermore, the left and right neighbors, in particular, account for horizontal spatial dependence in the absence of valid MVs that horizontally neighbor the lost MV.

An estimate for the lost MV is then computed using MV_1 and MV_2 and their valid orthogonal neighbors via *inverse distance weighting* (IDW), which is an interpolation technique that estimates unknown values using a weighted average of known values. The weight of each valid MV is assumed to be inversely proportional to its distance from the lost MV. The formula for estimating a lost MV based on WNVMVA, MV_{est}, is given as

$$MV_{est} = \frac{\sum_i w_i MV_i}{\sum_i w_i},\ i \in \{1, 1L, 1R, 1T, 2, 2L, 2R, 2B\},$$

where $w_i = \frac{1}{d_i}$ and d_i represents the distance of i^{th} valid MV from the lost MV.

The effectiveness of valid MVs chosen for WNVMVA can be analyzed by *spatial auto-*

correlation, which measures the spatial dependence of valid MVs based on their values and locations. Positive dependence among the valid MVs would justify not only their choice, but also the IDW formula. This can be calculated using *Moran's I* [16], which is an index ranging from -1 (perfect dispersion) to +1 (perfect correlation). Zero values indicate a random spatial relationship. Moran's I is defined as

$$I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(X_i - \bar{X})(X_j - \bar{X})}{\sum_i (X_i - \bar{X})^2},$$

where N is the number of valid MV values indexed by i and j according to MB (x, y) coordinates in a frame, $X_{i/j}$ represents valid MV values (i.e., $X_{i/j} \in \{MV_1, MV_{1L}, \cdots, MV_{2B}\}$), $\bar{X}$ is the mean of X, and w_{ij} is an element of an $N \times N$ *spatial weights matrix (SWM).*

In order to perform Moran's I analysis, consider the case when all 8 valid MVs are available. In addition, the supposedly lost MV is also considered as the 9^{th} MV by obtaining its value from an uncorrupted version of the lossy video. The initial 8 valid MVs will be tested for spatial dependence relative to the actual value and position of the supposedly lost MV.

A total of 9 MVs implies that $N = 9$. Subsequently, the SWM is a 9×9 matrix that expresses the potential for spatial dependence between two MV values at each i, j location [17]. Each of these interactions is represented in the form of a *weight.* Below is a general 9×9 SWM. By definition, diagonal elements (w_{ii}) are set to zero while the rest (w_{ij}) are distance-based weights. For example, the spatial weight relationship between MV_1 and MV_{1L} is represented by element w_{12} in the SWM. If these MVs are, for example, five MBs apart, then $w_{12} = \frac{1}{5}$.

$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{19} \\ w_{21} & w_{22} & \cdots & w_{29} \\ \vdots & \vdots & \ddots & \vdots \\ w_{91} & w_{92} & \cdots & w_{99} \end{pmatrix}$$

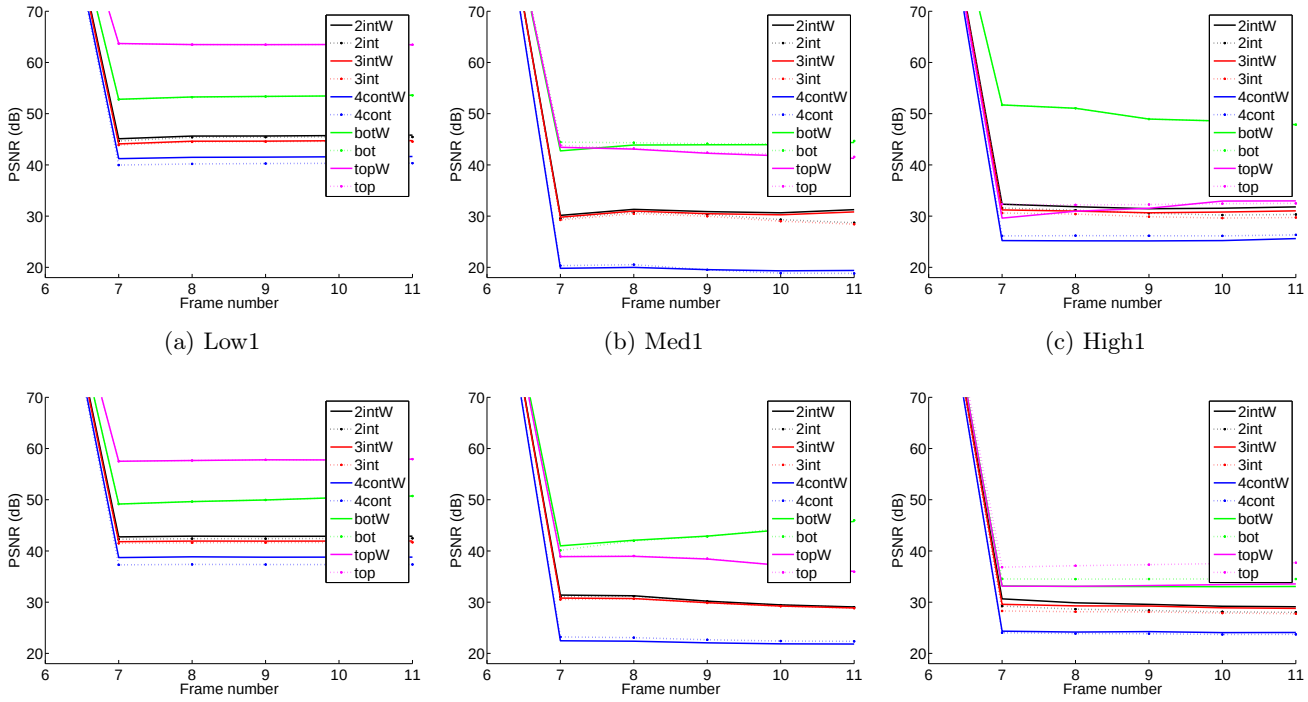
Using the SWM, Moran's I is calculated for each MV x- and y-component with respect to the nearest valid MVs. This is repeated for the next nearest valid MVs, and so on, for up to 12 iterations. The same procedure is repeated for default EC (i.e., CMVR without WNVMVA) using up to four orthogonal neighboring MVs. For each iteration, the average

Figure 3.4: A comparison between average Moran's I values for the default method vs. WNVMVA.

Moran's I values are plotted against the average distance of neighboring MVs as shown in Fig. 3.4. The graph shows an analysis of MV x-component values for frame 30 of the `X-Games (gap)` test video (see Table 4.1). Similar behavior applies not only for the corresponding y-component values, but also for the other frames as well as the other test videos.

In the graph, the vertical axes representing Moran's I values for WNVMVA and the default method are equally scaled. Thus, it can be seen that WNVMVA exhibits higher spatial correlation than the default method, even as its values decay. WNVMVA also shows that the closer the nearest valid MVs, the more spatially correlated they are relative to a lost MV. In addition, the exponential decay of Moran's I values for WNVMVA shows that the closer the nearest valid MVs relative to the lost MV, the more sensitive they are to spatial correlation. In the future, this could be useful for determining a suitable distance threshold, e.g., using WNVMVA only if the average distance of valid MV neighbors is 9 MB units or less. The horizontal axes are scaled differently since MV neighbors corresponding to WNVMVA and the default method occur in different regions. Valid neighboring MVs for WNVMVA are assumed to be in adjacent slices or further as shown in Fig. 4.5, whereas neighboring MVs for the default method are much closer and orthogonally adjacent to the lost MB.

WNVMVA ignores properly decoded intra-MBs, since they lack MVs, when searching for valid MVs. In order to investigate the validity of this procedure, additional experiments



(a) Low1 (b) Med1 (c) High1

(d) Low2 (e) Med2 (f) High2

Figure 3.5: PSNR measurements for 6 test videos with 5 loss scenarios concealed using CMVR. 2int = 2 intermittent slices, 3int = 3 intermittent slices, 4cont = 4 continuous slices, top = top slice and bot = bottom slice. The legends appended with W indicate CMVR with WNVMVA.

were performed by recursively applying WNVMVA. This was done by using WNVMVA to first estimate MVs for properly decoded intra-MBs encountered while searching for valid MVs. These estimated MVs were then used as valid MVs for finally applying WNVMVA towards EC. The corresponding PSNR and visual results showed that recursively applying WNVMVA to estimate MVs based on estimated ones, as is also done in the default method, yields worse results that applying WNVMVA just once.

3.5 Evaluation

3.5.1 Experiment Environment

Table 4.1 summarizes the test videos used in our experiment. There are two videos per speed level in overall motion – low, medium and high. Each video is about 60 frames long with five different lossy versions as follows: two intermittent slices (2int), three intermittent slices (3int), four continuous slices (4cont), top slice (top), and bottom slice (bot). These simulated losses resemble the effects of actual network losses. The lossy videos are then decoded and concealed using CMVR, which includes WNVMVA. The decoded results are compared against their corresponding original uncorrupted versions and their PSNR measurements are recorded.

3.5.2 Results and Analysis

Fig. 4.7 shows the PSNR comparisons for the six different videos, each with the five different loss scenarios. For each video, the corrupted frame corresponds to where the PSNR results decline sharply. These results compare CMVR against the default method (i.e., CMVR without WNVMVA) for all five loss scenarios. The amount of video degradation is proportional to the number of lost slices as the average PSNR drops in the following order: top/bot, 2int, 3int, and 4cont. CMVR offers PSNR improvement over the default method for most cases by 0.5 to 2 dB. Note that PSNR differences are most visually significant when below 25 dB.

Table 3.2 shows the percentage distribution of the five EC techniques within CMVR and the four EC techniques within the default method. WNVMVA offers decent improvement by contributing to approximately 50% of CMVR. This shows that there is a considerable amount of spatial dependence between lost MVs and surrounding valid MVs.

The visual significance of WNVMVA is shown in Fig. 4.11, which is compared against

	Low1	Low2	Med1	Med2	High1	High2
Title	Beautiful Nature	Planet Earth	African Cats	X-Games (skate)	X-Games (gap)	Motorcross
Bitrate (Mbps)	9.6	9.3	24.5	12.7	10	22.1
fps	29.97	29.97	23.98	23.98	23.98	23.98

Table 3.1: 1080p HD test videos.

the default EC techniques by concealing a single corrupted frame (see Fig. 4.3b). Fig. 4.12 shows the visual improvement in error propagation after 10 frames by employing CMVR over the default method for the 4cont case of Med1. Only a portion of the frame is considered in order to highlight the differences.

The major exceptions to WNVMVA improvements are some of the 4cont cases (High2, Med1, and Med2), where CMVR degrades video by up to 2 dB. Note that the top and bot loss scenarios exhibit minimal degradation and thus do not need extensive EC. The decoder may choose erroneous WNVMVA-based candidates due to the following two related reasons. First, the larger the lost slice region, the further apart and less spatially dependent the valid MVs are, resulting in poor WNVMVA. Second is the limited efficiency of BMA. WNVMVA-based MVs can end up generating MBs that have better BMA performance within their immediate surroundings but end up degrading overall video quality.

3.6 Conclusion and Future Work

The future of H.264 HD video over 802.11 wireless networks is bright. However, network losses necessitate EC, which can be performed through spatial MV recovery by estimation based on MVs available within the same frame. This paper proposed a new method called WNVMVA, which captures the spatial dependence among MVs by exclusively analyzing valid MVs. Our experiment showed that among the multiple candidates in CMVR for

Videos	Methods	Contribution of Individual EC Technique (%)				
		Zero MV	Mean MV	Median MV	Last MV	WNVMVA
Low1	Default	13.6	10.4	14.9	61.1	-
	CMVR	2.9	11.9	5.0	34.6	45.6
Low2	Default	13.6	8.3	22.3	55.6	-
	CMVR	1.8	6.9	7.8	28.7	54.9
Med1	Default	22.8	8.8	39.0	29.4	-
	CMVR	0.9	5.8	18.6	21.9	52.7
Med2	Default	18.2	7.6	29.6	44.5	-
	CMVR	3.7	7.5	18.2	23.6	47.0
High1	Default	25.4	11.0	40.6	23.0	-
	CMVR	6.0	5.7	25.7	17.6	45.0
High2	Default	23.4	10.8	39.1	26.7	-
	CMVR	4.3	5.1	18.6	17.0	54.9

Table 3.2: Percentage distribution of estimated MVs for the default method (Default) and CMVR (Default + WNVMVA) for six test videos.

(a) Zero MV (b) Last MV (c) Mean MV

(d) Median MV (e) WNVMVA

Figure 3.6: WNVMVA compared against pre-existing EC techniques (zero MV, last MV, mean MV and median MV)

estimating lost MVs, WNVMVA was chosen around 50% of the time and provided up to 2 dB in PSNR improvement.

There are a number ways WNVMVA can be improved. For example, since intra-prediction modes in H.264 are directional gradients potentially related to the direction of lost MVs, neighboring intra-MBs can be considered by predicting their MVs. Also, even though the limited number of available horizontal neighboring valid MVs are ignored as explained in Section 4.4, taking them into account could reveal extra spatial characteristics for MVs and improve WNVMVA. We also plan to investigate a temporal technique for MV recovery where estimated MVs can be forecasted using time series analysis. Temporal MV recovery can in turn supplement spatial MV recovery in a decision algorithm that chooses between spatial and temporal MV estimates. Alternatively, a unified spatial-temporal realization can be used.

(a) Original frame

(b) Default EC

(c) CMVR EC

Figure 3.7: Comparison between the original frame and error propagation 10 frames after the corrupted frame for the 4cont case of Med1 concealed with both the default method and CMVR.

Acknowledgements

This research was supported in part by LG Display Co., Korea and Ministry of Education Science and Technology (MEST) and the Korean Federation of Science and Technology Societies (KOFST).

Spatial Motion Vector Recovery for Wireless HD Video Transmission over WLANs using Weighted Nearest Valid Motion Vector Averaging

Kevin Gatimu, Ben Lee, Tae-Wook Lee, Chang-Gone Kim, Jone-Keun Shin

Under review

Chapter 4: Spatial Motion Vector Recovery for Wireless HD Video Transmission over WLANs using Weighted Nearest Valid Motion Vector Averaging

High definition (HD) and, more recently, 4K video are the standards of choice for today's video demands. These video standards are characterized by high data rates, but they can be compressed using H.264 and transmitted over 802.11 wireless networks. However, such networks are prone to packet losses, which result in degraded perceptual video quality. The problem is exacerbated by the ever-growing number of wireless mobile devices. It is thus important to perform error concealment. Spatial motion vector recovery is a key error concealment technique. This paper proposes a new spatial motion vector recovery technique called Weighted Nearest Valid Motion Vector Averaging (WNVMVA), which uses properly decoded motion vectors to estimate lost ones.

4.1 Introduction

Wireless HD video transmission (WHDVT) over 802.11 is an important technology for direct, peer-to-peer streaming applications, such as screen mirroring for home entertainment [1] and N-screen services [2, 3]. Recently, a number of technologies have become available for WHDVT. For example, Google's *Chromecast* uses remote control from a mobile device to pull multimedia content from the Internet [4] and can also perform mirroring between Android devices [5]. Similarly, Intel's Miracast-based *Wireless Display* (WiDi) [6] and Apple's *AirPlay* [7] enable multimedia streaming and mirroring between compatible devices.

Despite the availability of these technologies, WHDVT still presents many challenges because each video stream is bandwidth-intensive and multiple peer-to-peer streams will have to be simultaneously supported. For example, a full HD video (1080p at 60 fps) compressed using H.264 Main Profile @Level 4.2 would require a data rate of up to 50 Mbps [8]. For most general-purpose 1080p videos, YouTube recommends at least 8 Mbps [9]. Thus, supporting multiple streams (video or other data) on the same channel would quickly saturate the network. Moreover, 802.11 transmission is prone to interference from other

devices operating at 2.4 GHz or 5 GHz. In spite of these limitations, 802.11 is still very promising. For example, 802.11ac has recently become available and boasts a throughput of 1.3 Gbps [10]. Also, WiGig promises to deliver a maximum data rate of 7 Gbps and is backward compatible with 802.11 [11].

Due to the challenges of WHDVT via 802.11, network packet losses are common leading to distorted video. Nevertheless, *error concealment* (EC) can be used to minimize the perceived visual distortion of received video. EC involves reconstructing lost video information by utilizing redundancy and correlation of the received data in the spatial and temporal domains.

In [12], we presented a new spatial EC technique called *Weighted Nearest Valid Motion Vector Averaging* (WNVMVA), which performs motion vector (MV) recovery by exploiting spatial relationships among *valid* (i.e., properly received and decoded) MVs. This is different from existing EC techniques, such as zero MV, mean MV, median MV and last MV, which utilize estimated MVs in addition to valid ones. This leads to approximation errors and error propagation due to estimating lost MVs based on previously estimated MVs. WNVMVA was combined with pre-existing techniques to provide a set of candidates for estimating lost MVs. Our experimental results show that including the proposed WNVMVA in the EC candidate system in many cases improves performance. Furthermore, WNVMVA outperforms frame copy, which is very basic yet is commonly used to conceal slice-based losses in popular media players such as *QuickTime* and *Windows Media Player*. WNVMVA also outperforms other common spatial EC methods within the MV candidate system.

This paper extends our earlier work on WNVMVA by making the following contributions:

- Two additional methods, directional interpolation [13], which is a pixel-based method, and polynomial interpolation [14], which a MV-based method, are implemented and compared against WNVMVA.

- Performance evaluation is significantly expanded to include additional spatial analysis, three additional test videos, and accompanying PSNR calculation and statistical evaluation.

4.2 Background

This section provides the necessary background to better understand the motivation behind the proposed method.

4.2.1 H.264

H.264 compression is done via *intra-prediction* and *inter-prediction* [8]. Intra-prediction uses samples from the current frame as references and results in intra-macroblocks (I-MBs), which make up I-frames. Inter-prediction uses samples from past frames as well as both past and future frames as references and results in P-MBs and B-MBs, which make up P- and B-frames, respectively. For every block that is to be inter-predicted, *motion estimation* is used to search for the best-match block in the previous frame(s). This generates a motion vector (MV) representing a linear offset between the current block and the best-match block. The pixel difference between the actual (current) and inter-predicted (best-match) block is the *prediction error*. Both the MV and the prediction error are encoded, which is more efficient than encoding a complete image blocks. Sometimes the prediction error is too large for efficient coding, e.g., complex movement (stretching, contortion, etc.) and when a frame's contents at the edges lack suitable references in the past frames. In these situations, an H.264 encoder performs intra-coding rather than inter-coding resulting in some I-MBs within both P-frames and B-frames.

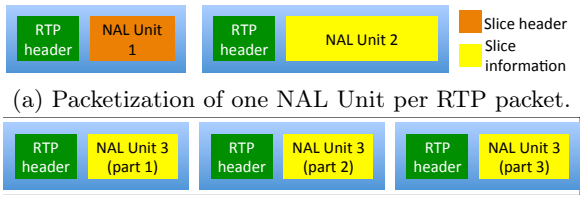
When an MV is lost during transmission, its corresponding block cannot be properly decoded. Furthermore, this usually results in the absence of an accurate reference for a future inter-predicted block. This degradation process continues until an intra-coded block is encountered. This phenomenon is called *error propagation*. Therefore, in order to conceal inter-prediction errors within a particular frame and reduce error propagation, lost MVs need to be properly estimated. *Spatial MV recovery* is a type of EC that estimates lost MVs by inferring from data within the current frame.

4.2.2 Packet Loss and Its Effect on Video

H.264 data is coded as packets called *Network Abstraction Layer (NAL) Units*, which structure the H.264 bitstream into a format compatible with different types of transport networks. For example, NAL Units are typically packetized via the Real-time Protocol (RTP) Payload

Format [15]. Fig. 4.1 shows the three different packetization schemes for NAL Units using this specification.

The RTP Payload Format for H.264 video is robust because more important NAL Units, such as those containing slice headers, are small enough to fit within individual RTP packets, e.g., NAL Unit 1 in Fig. 4.1a. Large NAL Units that contain slice data can be fragmented, such as NAL Unit 3 in Fig. 4.1b. Aggregation of multiple NAL Units, as in NAL Units 4, 5 and 6 in Fig. 4.1c, is also possible. This differs from Transport Stream (TS) packetization, which is used in legacy MPEG streaming. TS packetization disregards the H.264 NAL Unit structure and uses fixed-size RTP packets – seven 188-byte TS packets per RTP packet as shown in Fig. 4.2. A possible, and detrimental, consequence of TS packetization is fragmentation of a slice header NAL Unit. If a packet containing even just part of a slice header is lost, then the whole slice is considered lost regardless of how much of the rest of



(a) Packetization of one NAL Unit per RTP packet.

(b) Packetization of one NAL Unit across multiple RTP packets.

(c) Packetization of multiple NAL Units within one RTP packet.

Figure 4.1: NAL Unit packetization payload structures using the RTP Payload Format for H.264 Video.

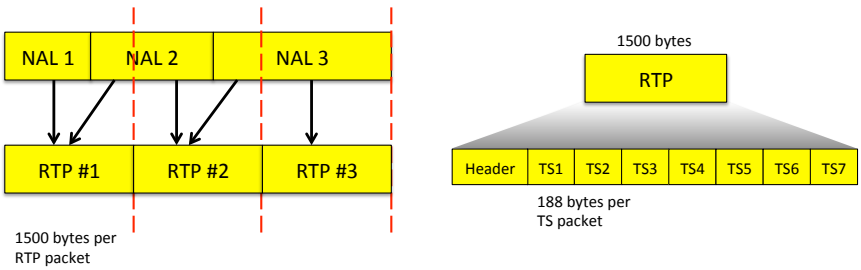
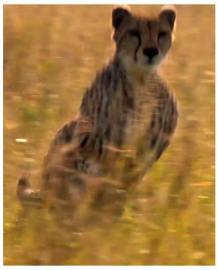
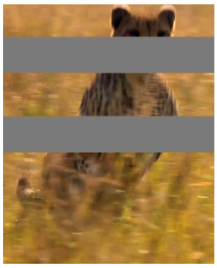


Figure 4.2: NAL Unit packetization using TS Packets.

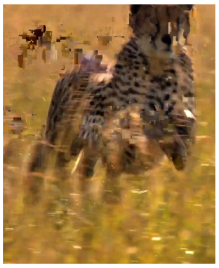


(a) Original frame.



(b) Corrupted frame (slices 2 and 4 are lost).

(c) EC via frame copy.



(d) Error propagation 5 frames later.

Figure 4.3: Portion of original frame vs. corrupted frame (slices 2 and 4 are lost). Poor EC via frame copy results in error propagation five frames later.

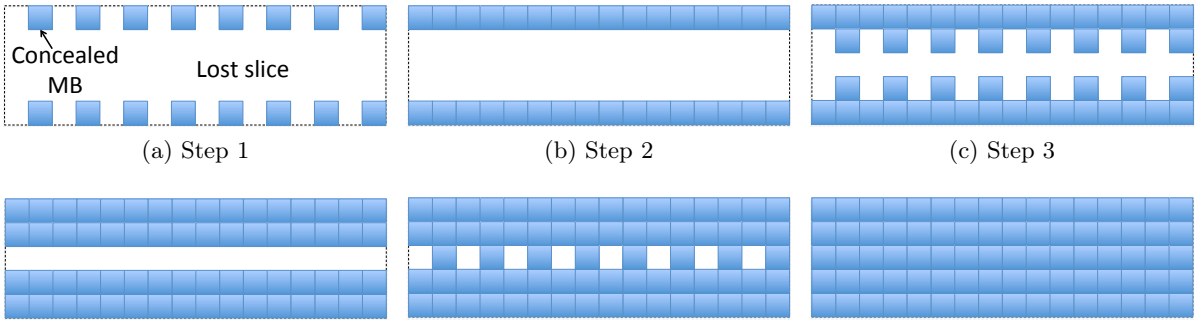
the slice information is properly received.

Aside from slice headers, most NAL Units directly correspond to actual video frame slices. Therefore, lost packets result in lost NAL Units, which lead to missing slices or parts of slices. A loss of an entire slice may occur due to severe packet loss or loss of packets containing slice header NAL Units.

For example, Fig. 4.3a shows part of an original frame and Fig. 4.3b shows the same frame sample with slices 2 and 4 missing due to packet losses. A simple form of EC is to perform *frame copy* as shown in Fig. 4.3c, which involves simply copying video data from a previous frame by using a zero MV for motion compensation. As can be seen in Fig. 4.3d, this is not visually sufficient and also results in significant error propagation. Therefore, more accurate EC techniques are needed.

4.2.3 Error Concealment Techniques

This section discusses the default H.264 EC techniques found in *FFmpeg*, the underlying software of *VLC Media Player*, which was the media player of choice for our previous work [16]. The FFmpeg H.264 decoder applies EC to each lost MV using orthogonally adjacent MVs, spatial mean MV, spatial median MV, last MV, and zero MV [17]. The orthogonally adjacent MVs consist of *Left MV*, *Right MV*, *Top MV* and *Bottom MV*. The *Mean MV* averages the orthogonally adjacent MVs to recover a lost MV, while *Median MV* uses the median instead. *Last MV* copies the MV from the same spatial location of the previous inter-predicted



(a) Step 1 (b) Step 2 (c) Step 3

(d) Step 4 (e) Step 5 (f) Step 6

Figure 4.4: FFmpeg error concealment MB sequence.

reference frame, while *Zero MV*, also known as frame copy, uses a zero-valued MV.

Each EC technique generates an estimated MV and resultant MB. Each estimated MB is compared against neighboring MBs via a *Boundary Matching Algorithm* (BMA) that calculates the total absolute pixel value difference between the edge pixels of the estimated MB and the adjacent pixels of surrounding MBs. The estimated MB that yields the least absolute difference is picked as the final estimated MB.

FFmpeg EC is based on available surrounding MVs, which are either properly decoded or lost and thus estimated. These surrounding MVs will be referred to as *MV estimators*. Each lost MV is surrounded by a maximum of four orthogonal MV estimators. If a neighboring MB is intra-coded, then it has no corresponding MV estimator, and thus ignored. The MV estimators are then used to generate multiple MV candidates.

Lost MBs are concealed in a convergent manner as shown in Fig. 4.4, which illustrates a simplified lost slice scenario. The outer lost MBs are concealed first as they are initially the only ones with neighboring MBs. This is followed by the next inner row of MBs and the process is repeated until the whole slice is completely concealed. Also, for each row of lost MBs, the even-entry MBs are concealed first, followed by the odd-entry MBs. The even-entry MBs are then re-concealed considering the new MV estimators provided by the recently concealed odd-entry MBs. Similarly, the odd-entry MBs are then re-concealed. This is repeated for each row until the estimated MVs do not change through the iterations or for up to 10 iterations, whichever comes first.

4.3 Related Work

EC techniques are categorized as spatial, temporal, or a combination of both. Spatial techniques utilize correlation between the lost MV and the MVs surrounding the lost MV within a frame. On the other hand, temporal techniques use information from past and future frames relative to the current frame.

Temporal EC techniques are used in situations where spatial information is missing, e.g., an entire frame is lost. The most basic temporal EC is to recover lost video frames through frame copy, where the visual contents of the previous frame are copied in place of the lost frame. This can be improved via motion copy, whereby the co-located MVs are copied instead [18]. Motion copy can be further improved via the *Motion Copy MV refinement* algorithm proposed by Chien *et al.* [19]. In this method, motion copy MVs (MCMVs) are initially used to conceal the lost frame. MV differences between previous successive frame pairs are then calculated. This is then followed by finding the maximum MV difference (MMVD) between MCMVs in the lost frame and valid MVs in the previous frame for both the x- and y-components of the MVs. MMVD_x and MMVD_y are then used to define the x and y dimensions of a refinement area in which to find new MMVD values. This process is repeated until the refinement area does not change through the iterations. The MV difference with the highest occurrence within the final refinement area is then added to the original MCMVs. Note that the MCMV refinement is used to recover lost frames based on the assumption that frame sizes are small, i.e, QCIF (176×144), CIF (352×288) and 4CIF (704×576) resolutions. Thus, each H.264-encoded frame is contained within a single network packet, and any lost packets would result in lost frames.

In another related work, Yan *et al.* proposed a method called *Hybrid Motion Vector Extrapolation* (HMVE) [20], where the MVs of lost pixels are formed by extrapolating MVs from the previously decoded frame. For each lost pixel, an estimated MV is formed by averaging the MVs corresponding to the extrapolated 4×4 block(s) that overlap the lost pixel. HMVE is also tested on QCIF and CIF video sequences and whole frames are contained within individual network packets.

Li *et al.* proposed an entire frame recovery algorithm using MV extrapolation and median filtering [21]. In this method, candidate MVs are formed for the lost frame by extrapolating MVs from previous available frames at the 8×8 block level. If an estimated 8×8 block in the lost frame has dissimilar candidate MVs, the process is repeated for the same block

at the 4×4 level and, if necessary, at the 2×2 level. Any areas without extrapolated MVs are accounted for via median filtering and linear interpolation. This algorithm is used to address lost network packets containing whole frames. The accompanying experiments involve randomly dropping packets containing P-frames at QCIF and CIF resolution. Our proposed WNVMVA method is spatial in nature and thus orthogonal to temporal methods. Therefore, it is possible to combine WNVMVA with these temporal EC techniques.

The most basic spatial EC techniques estimate the lost MV based on surrounding MVs. For example, the FFmpeg decoder utilizes eight interpolation techniques consisting of Left MV, Right MV, Top MV, Bottom MV, Zero MV, Mean MV, Median MV, and Last MV [17]. However, these methods perform estimations based on both valid and previously estimated MVs. On the other hand, WNVMVA uses exclusively valid MVs to estimate lost MVs.

As explained in Section 4.2.1, it is possible to have I-MBs within inter-frames (i.e., P- and B-frames). Therefore, there are also spatial EC methods based on estimating pixel information, rather than MVs, since I-MBs lack MVs. A common example of such spatial EC techniques is *Weighted Pixel Averaging* (WPA) [22], where each lost pixel is concealed by using a distance-weighted average of the four orthogonally closest pixels. WPA is based on the assumption of spatial continuity and produces blurry distortion. For this reason it is more suitable for I-frames. In fact, spatial EC methods are commonly applied towards I-frames, while our work is applied towards inter-frames.

An example of a spatial EC method geared towards I-frames is *Directional Interpolation* (DI) with edge preservation and smoothing [13]. This method performs edge detection by calculating a gradient field through convolution between a Sobel filter and luminance values of the boundary pixels surrounding the lost block. The boundary pixels are typically provided by only the top and left neighboring blocks since lost blocks are recovered in raster order, thus rendering the bottom and right neighboring ones unavailable. Dominant gradient directions are then calculated for the top and left boundary pixels. Corresponding partitions (top and left) are also determined for the lost block. Each pixel in the lost block is then interpolated along the dominant gradient direction corresponding to the partition in which it is located. Although spatial in nature, DI does not take advantage of available MVs. Also, it can be computationally intensive, especially in the context of HD video.

Jin-wang *et al.* combined WPA and DI into an algorithm that adaptively switches between the two using a threshold determined by directional entropy [23]. The more similar the edge directions of surrounding pixels, the lower the directional entropy and the higher

the likelihood of using DI rather than WPA. This not only preserves existing edges but also avoids introduction of significantly erroneous ones. WPA is also improved in [24], where multi-directional interpolation is used rather than just bilinear interpolation. In addition to traditional WPA, this method performs interpolation based on boundary pixel pairs obtained from gradient fields produced by applying the Sobel mask as in DI. Like DI, however, these WPA improvements can be computationally intensive as they are pixel-based operations and also disregard available MVs. Our proposed method is not only block-based, and thus less computationally intensive, but it also utilizes available MVs.

In addition to sub-HD video resolution, most of the existing work in EC, especially spatial EC, only accounts for isolated lost MBs rather than slice-based losses. For instance, Zheng *et al.* used Lagrange interpolation [25] and a polynomial model [14] to approximate a missing MV based on orthogonally adjacent valid (properly decoded) MVs. Similarly, Seth *et al.* used a B-spline method to estimate isolated lost MVs via statistical interpolation [26].

Furthermore, network packet losses in WHDVT typically result in loss of partial slices, whole slices, or multiple slices, rather than isolated lost MBs. Even a single bit error within an H.264 bitstream, which is variable-length-coded, can result in adverse visual effects manifested as slice-like losses. In addition, if a slice header, despite its limited size, is lost or corrupted, the entire slice will be undecodable, thus rendering the slice lost as described in Section 4.2.2. To address such issues, our proposed WNVMVA conceals slice-based losses.

4.4 The Proposed Method

The goal of WNVMVA is to exploit the spatial relationships among MVs by using only *valid MVs*, i.e., properly decoded MVs, that are spatially closest to the lost MVs. Note that MVs could be invalid due to intra-coding or packet loss. The spatially closest valid MVs are then used to estimate lost MVs. This is in contrast to the existing methods, which use both estimated and valid MVs. By analyzing only valid MVs, a more accurate spatial relationship among MVs can be exposed.

Since not all MBs contain MVs, the number and location of valid MVs for estimating each lost MV vary with the distribution of the valid MVs around a lost region. Furthermore, the unified distribution of both valid and lost MVs is unknown. Therefore, nonparametric regression is used to estimate lost MVs. A simple machine learning algorithm for implementing nonparametric regression is k-nearest neighbors (k-NN).The k-NN algorithm determines

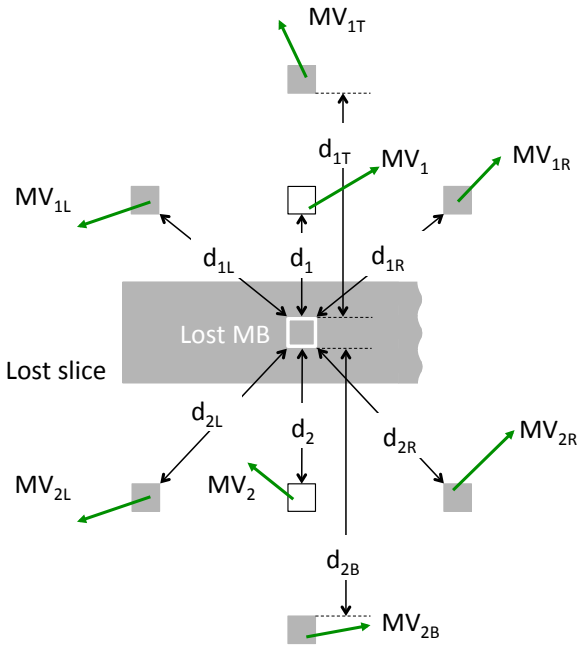


Figure 4.5: MV_1 and MV_2 and their neighbors are used to estimate the value of the lost MV via inverse distance weighting.

the number and location of valid MVs for estimating each lost MV. Ideally, this can be reiterated for each lost MV. However, this would be too computationally expensive. Therefore, using offline analysis, the optimal number of k neighboring valid MVs per lost MV is determined relative to the distribution of valid MVs.

Fig. 4.5 illustrates WNVMVA, where for each lost MV, one valid MV above (MV_1) and another below (MV_2) are initially selected based on the availability of properly decoded MVs. The distance between $MV_{1/2}$ and the lost MV is $d_{1/2}$. Note that the MVs between the lost MV and $MV_{1/2}$ are assumed to be invalid because either the MVs are lost or they are intra-MBs.

The corresponding valid MVs in the horizontal direction are usually unavailable because lost H.264 visual information stretches across the whole frame as illustrated in Fig. 4.3b. However, some of the top and/or bottom rows of lost slice regions could start or end in the middle of a frame, which would result in a limited number of horizontally available valid MVs. For example, a typical frame in a HD video contains 8 slices and each slice consists

of 8 to 9 rows of MBs. If one of these slices is lost and it begins or ends in mid-frame, then only about half a row of lost MVs (less than 10% of the total lost MVs in a slice) would have horizontally neighboring valid MVs. However, WNVMVA assumes that, in general, there are no available valid MVs in the horizontal direction from the perspective of a lost MV.

As shown in Fig. 4.5, once MV_1 and MV_2 are established, their respective orthogonal neighboring valid MVs, if available, are then determined. The nearest *valid* neighbors for $MV_{1/2}$ are the left ($MV_{1L/2L}$), the right ($MV_{1R/2R}$) and the top/bottom ($MV_{1T/2B}$) neighbors. Again, any MVs that may exist between MV_1, or MV_2, and its nearest valid neighbor, are assumed to be invalid due to intra-coding or packet loss.

The orthogonal neighboring valid MVs of MV_1 and MV_2 provide complimentary spatial dependence information that is useful in more accurately estimating the lost MV. Furthermore, the left and right neighbors, in particular, account for horizontal spatial dependence in the absence of valid MVs that horizontally neighbor the lost MV.

An estimate for the lost MV is then computed using MV_1 and MV_2 and their valid orthogonal neighbors via *inverse distance weighting* (IDW), which is an interpolation technique that estimates unknown values using a weighted average of known values. The weight of each valid MV is assumed to be inversely proportional to its distance from the lost MV. The formula for estimating a lost MV based on WNVMVA, MV_{est}, is given as

$$MV_{est} = \frac{\sum_i w_i MV_i}{\sum_i w_i},\ i \in \{1, 1L, 1R, 1T, 2, 2L, 2R, 2B\},$$

where $w_i = \frac{1}{d_i}$ and d_i represents the distance of i^{th} valid MV from the lost MV.

The effectiveness of the configuration of valid MVs chosen as MV estimators for WNVMVA (see Fig. 4.5) can be measured and analyzed using *spatial autocorrelation statistics*. This is shown to be superior to the corresponding results for the orthogonal arrangement of MV estimators as seen in existing methods. Such statistical analysis can achieved through *Moran's I*, which is a measure of the dependence among spatial units based on their values and locations simultaneously [27]. Moran's I calculations are done for each MV, yielding values that range between -1 and +1, i.e., perfect dispersion and perfect correlation respectively. Perfect correlation means that the MV being analyzed is completely dependent on its neighbors. A simple example of this would be if the MV being analyzed and its neighbors are all identical. On the other hand, perfect dispersion implies that the MV in question behaves

in direct opposition to its neighbors, e.g., if its value is very small while its neighbors are very large or vice versa. A zero value indicates no correlation and thus a random spatial relationship.

Moran's I is defined as

$$I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij}(X_i - \bar{X})(X_j - \bar{X})}{\sum_i (X_i - \bar{X})^2},$$

where N is the number of valid MV values indexed by i and j according to MB (x, y) coordinates in a frame, $X_{i/j}$ represents valid MV values (i.e., $X_{i/j} \in \{MV_1, MV_{1L}, \cdots, MV_{2B}\}$), $\bar{X}$ is the mean of X, and w_{ij} is an element of an $N \times N$ *spatial weights matrix* (SWM). Each SWM element (i, j) expresses the potential for spatial dependence between a pair of MV values. Each of these interactions is represented in the form of a numerical *weight*.

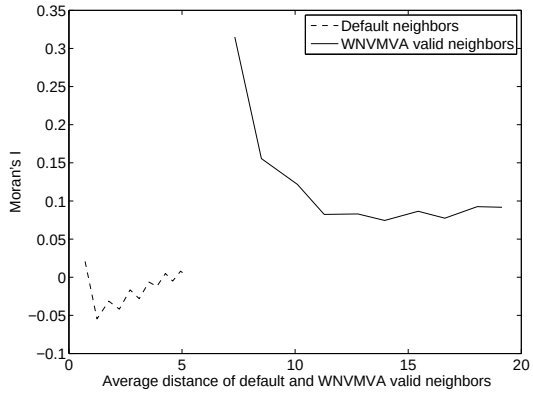
For the sake of describing Moran's I analysis for WNVMVA, consider the case when all 8 valid neighboring MVs are available as shown in Fig. 4.5. In addition, the supposedly lost MV is also considered as the 9^{th} MV by obtaining its value from an uncorrupted version of the lossy video. The initial 8 valid MVs will be tested for spatial dependence relative to the actual value and position of the supposedly lost MV. Note that, in practice, it is possible to perform the analysis with less than 8 neighboring MVs.

A total of 9 MVs implies that $N = 9$. Subsequently, the SWM is a 9×9 matrix such as the one shown below. By definition, diagonal elements (w_{ii}) are set to zero while the rest (w_{ij}) are distance-based weights. For example, the spatial weight relationship between MV_1 and MV_{1L} is represented by element w_{12} in the SWM. If these MVs are, for example, five MBs apart, then $w_{12} = \frac{1}{5}$.

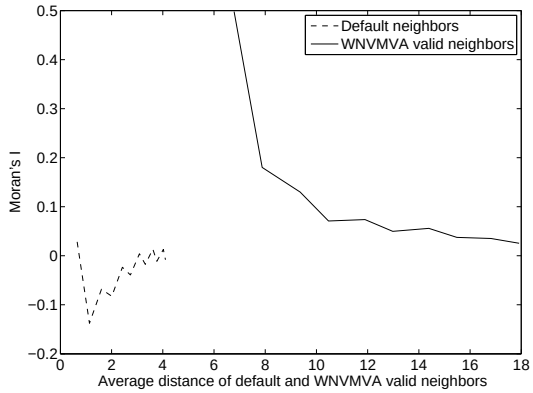
$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{19} \\ w_{21} & w_{22} & \cdots & w_{29} \\ \vdots & \vdots & \ddots & \vdots \\ w_{91} & w_{92} & \cdots & w_{99} \end{pmatrix}$$

Using the SWM, Moran's I is calculated for each MV x- and y-component with respect to the nearest valid MVs. This is repeated for the next nearest valid MVs, and so on, for up to 12 iterations. As a comparison, the same procedure is repeated for the default EC method, where there are up to four orthogonal neighboring MVs per MV to be analyzed.

For each iteration, the average Moran's I values are plotted against the average distance



(a) Single frame, x-component



(b) Multi-frame multi-video average for both x and y MV components

Figure 4.6: A comparison between average Moran's I values for the default method vs. WNVMVA.

of neighboring MVs as shown in Fig. 4.6a. The graph shows an analysis of MV x-component values for frame 30 of the `X-Games (gap)` test video (see Table 4.1). Similar behavior applies not only for the corresponding y-component values, but also for the other frames as well as the other test videos. This can be inferred from Fig. 4.6b, which shows the average Moran's I analysis for multiple videos and multiple frames for both x and y MV components.

Each graph shows that WNVMVA exhibits higher spatial correlation compared to the default method, even with increasing average distance of neighboring MVs. This means that the configuration of MV estimators for WNVMVA is more spatially dependent than

	Low1	**Low2**	**Low3**	**Med1**	**Med2**	**Med3**	**High1**	**High2**	**High3**
Title	Beautiful Nature	Planet Earth	Alpine	African Cats	X-Games (skate)	BMX (pipe)	X-Games (gap)	MotorX	BMX (gap)
Bitrate (Mbps)	9.6	9.3	3.8	24.5	12.7	12.4	10	22.1	13.6
fps	29.97	29.97	29.97	23.98	23.98	23.98	23.98	23.98	23.98

Table 4.1: 1080p HD test videos.

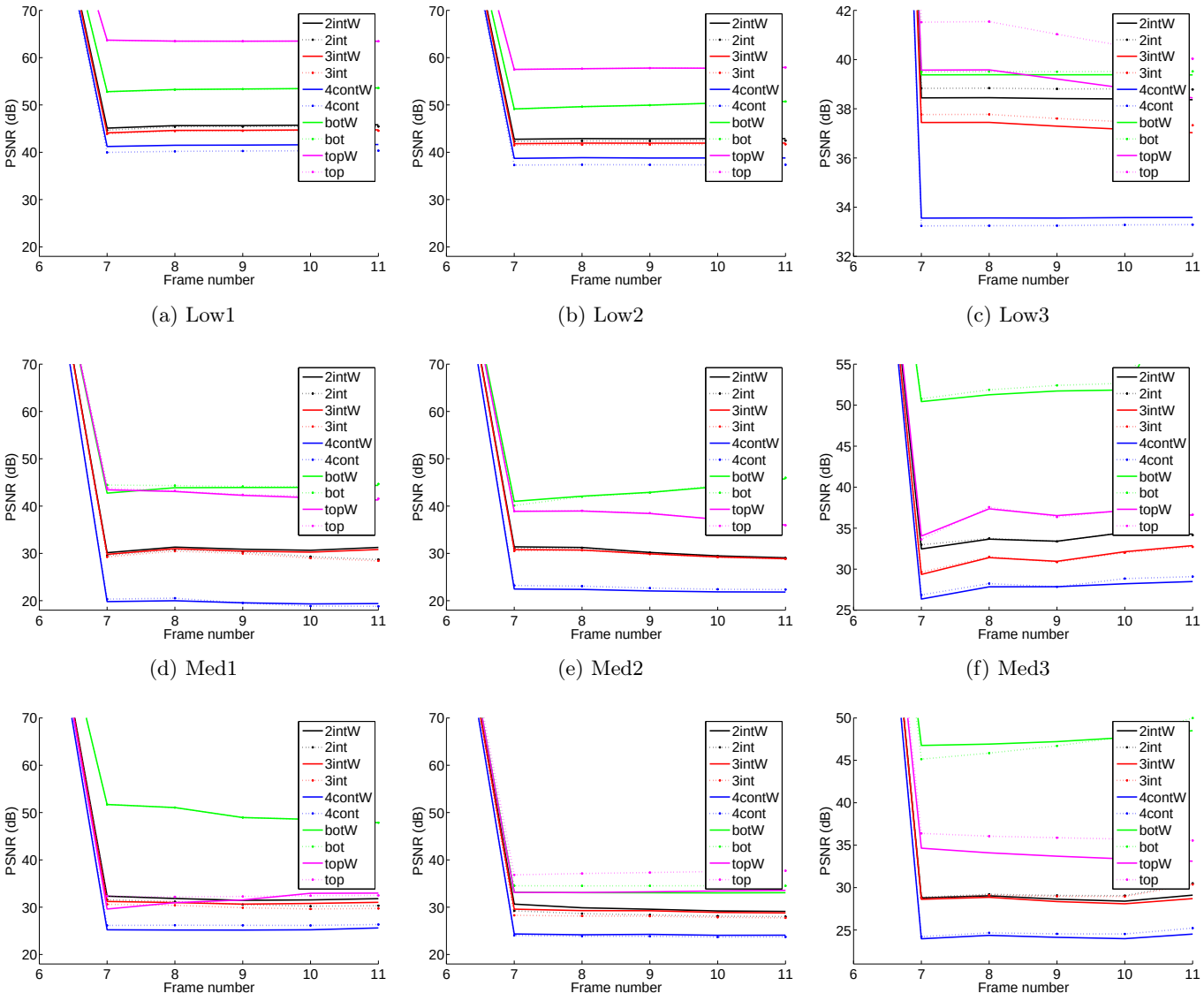
the orthogonal arrangement seen in the default method. The graphs also show that in the case of WNVMVA, the closer the nearest valid MVs, the more spatially correlated they are relative to a lost MV. This is further supported by the exponential decay of the graphs.

The range of x-values for WNVMVA is offset from that for the default method because valid neighboring MVs for WNVMVA are in adjacent slices as shown in Fig. 4.5, whereas neighboring MVs for the default method are much closer and orthogonally adjacent to the lost MB. The Moran's I values for the default method are close to 0 because of the presence of one or more intra-coded MBs orthogonally adjacent to a lost MB, which effectively reduces the average spatial correlation.

4.5 Evaluation

4.5.1 Experiment Environment

Table 4.1 summarizes the test videos used in our experiment. There are three videos per motion level – low, medium, and high. Each test video is 60 frames long and contains the frame sequence *IPPPP...* For each video, the 7^{th} frame is corrupted using five different loss scenarios as follows: two intermittent slices (2int), three intermittent slices (3int), four continuous slices (4cont), top slice (top), and bottom slice (bot). These simulated losses resemble the effects of actual network losses. The lossy videos are then decoded and concealed using three EC candidate systems. The first is the default method used by *FFmpeg* (*Default*), which consists of Zero MV, Left MV, Right MV, Top MV, Bottom MV, Mean MV, Median, MV, and Last MV. The second is the default method together with WNVMVA (*Default+WNVMVA*). The third candidate system (*Default+WNVMVA+PI+DI*) adds polynomial interpolation (PI) [14] and edge-preserving directional interpolation (DI) [13] to *Default+WNVMVA*. This provides further evaluation of WNVMVA because PI and DI (and



(a) Low1 (b) Low2 (c) Low3

(d) Med1 (e) Med2 (f) Med3

(g) High1 (h) High2 (i) High3

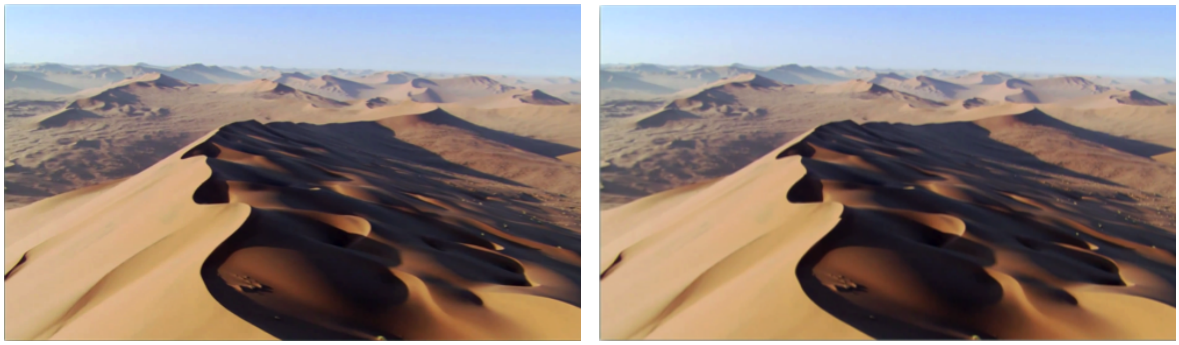
Figure 4.7: PSNR measurements for 6 test videos with 5 loss scenarios concealed using *Default* and *Default+WNVMVA*. 2int = 2 intermittent slices, 3int = 3 intermittent slices, 4cont = 4 continuous slices, top = top slice and bot = bottom slice. The legends appended with `W` indicate *Default+WNVMVA*

their variants) are popular spatial EC methods, which perform interpolation of MVs and pixel values respectively. The decoded results are compared against their corresponding original uncorrupted versions and their PSNR measurements are recorded.

Due to the lack of neighboring valid MVs in the horizontal direction, their estimated versions were incorporated into WNVMVA calculations for evaluation purposes. These estimated horizontally neighboring MVs were produced by the *Default* methods as described in Section 4.2.3. However, this produced negligible improvement in both PSNR and perceptual quality at the expense of significantly higher computation complexity. Therefore, horizontally neighboring MVs were ignored altogether.

Additional experiments were performed in order to investigate the validity of ignoring properly decoded intra-MBs when searching for valid MVs during WNVMVA calculations. This was done by applying WNVMVA recursively as follows. First, WNVMVA was used to estimate pseudo-MVs for the ignored intra-MBs as if these intra-MBs were lost MBs. The pseudo-MVs were then used as part of the nearest valid MV set for finally applying WNVMVA towards EC. The corresponding PSNR and visual results showed that recursively applying WNVMVA to estimate MVs based on estimated ones yields worse results than applying WNVMVA just once. Therefore, it is better to ignore intra-MBs when searching for valid MVs.

4.5.2 Results and Analysis



(a) Frame copy (b) Default+WNVMVA

Figure 4.8: Error concealment comparison for Low1 using (a) Frame copy and (b) Default+WNVMVA

Table. 4.2 shows the average PSNR values, including the corrupted frame and 5 subsequent frames. The results are reported for the nine different test videos, each with the five different loss scenarios. Fig. 4.7 shows the corresponding graphical comparison. For each video, the corrupted frame (i.e., frame 7) corresponds to where the PSNR results degrade sharply. These results compare *Default* against *Default+WNVMVA* for all five loss scenarios. The amount of video degradation is proportional to the number of lost slices as the average PSNR drops in the following order: top/bot, 2int, 3int, and 4cont. *Default+WNVMVA* offers PSNR improvement over *Default* for most cases by 0.5 to 2 dB. Although the improvement appears to be small, PSNR differences are most visually significant when they occur below about 35 dB. This is due to the logarithmic nature of the dB scale. In fact, even larger PSNR differences higher up on the dB scale do not exhibit any significant visual difference. For example, Fig. 4.8 shows an EC comparison between frame copy and *Default+WNVMVA* for the 4cont case of Low1. The results are virtually indistinguishable despite the PSNR result for *Default+WNVMVA* being 1 dB higher than *Default* as shown in Fig. 4.7a. This PSNR difference occurs above the 40-dB mark.

The results in Fig. 4.8 also show that simple frame copy provides sufficient EC for low motion video. This is because low motion results in small changes through successive frames, which in turn result in minimal MV differences.

Fig. 4.7 shows that the top/bot loss scenarios exhibit the best PSNR results. This is because these scenarios are limited to just single-slice losses resulting in the lowest degradation. Furthermore, the PSNR differences between *Default* and *Default+WNVMVA* occur above the 35-dB mark thus resulting in minimal corresponding visual differences. This is true even when *Default* outperforms *Default+WNVMVA*, e.g., in High2 as shown in Fig. 4.7h (PSNR) and Fig. 4.9 (visual).

On the other hand, the 4cont loss scenario results in the worst PSNR performance. This is because 4cont leads to the largest number of lost slices, i.e. four, which are also contiguous. The four-slice gap diminishes the spatial relationship between valid MVs above and below the lost slices, thus reducing the effectiveness of WNVMVA. In fact, the worst PSNR performance below the 35 dB mark can be found in the 4cont cases for Med2 and High1. In spite of these shortcomings, *Default+WNVMVA* shows PSNR improvement over *Default* for most cases.

In contrast to 4cont, the 2int and 3int loss scenarios result in intermittent lost slices, where there is only a one-slice gap between valid slices, and thus spatial correlation between

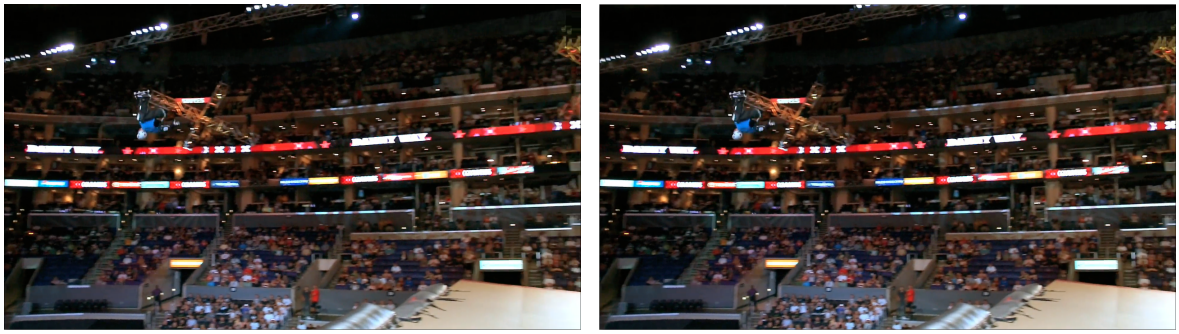
Video	Method	Loss Scenario				
		top	bot	2int	3int	4cont
Low1	Default	53.28	63.52	45.28	44.41	40.22
	Default+WNVMVA	53.30	63.52	45.58	44.56	41.49
Low2	Default	49.97	57.73	42.40	41.61	37.34
	Default+WNVMVA	49.97	57.73	42.85	41.90	38.79
Low3	Default	39.51	40.93	38.81	37.59	33.26
	Default+WNVMVA	39.38	39.13	38.41	37.27	33.56
Med1	Default	44.34	42.62	29.73	29.43	19.61
	Default+WNVMVA	43.77	42.37	30.85	30.47	19.61
Med2	Default	43.06	37.91	30.16	29.81	22.74
	Default+WNVMVA	43.17	37.89	30.27	29.90	22.12
Med3	Default	54.85	36.27	33.75	31.35	28.18
	Default+WNVMVA	54.29	36.33	33.65	31.34	27.75
High1	Default	49.62	32.34	30.76	30.05	26.19
	Default+WNVMVA	46.62	31.60	31.76	30.93	25.28
High2	Default	34.52	37.31	28.48	28.02	23.83
	Default+WNVMVA	33.07	33.29	29.68	29.16	24.18
High3	Default	47.10	35.92	29.34	29.24	24.64
	Default+WNVMVA	47.41	33.79	28.79	28.53	24.20

Table 4.2: Average PSNR (dB) values for each test sequence including corrupted frame and 5 frames later.

valid MVs is improved. Subsequently, the corresponding PSNR performance is better. For all intermittent loss cases, except for High3, *Default+WNVMVA* is better than *Default*. Isolated situations such as poor PSNR performance in High3 occur when the original lost slices have smaller and less visually significant portions, which happen to be responsible for the bulk of the poor PSNR performance. These portions mostly include intra-coded MBs, which are detrimental towards MV spatial correlation.

For example, Fig. 4.10 shows the MB energy map for the original version of the corrupted frame in High3. The MBs with higher amounts of data are brighter, i.e., have more energy, and are mostly intra-coded. Such MBs can be found in the highlighted area of slice 4, which

represents less than 12% of the slice. Considering the 3int case for High3, where slices 2, 4 and 6 are lost, the PSNR of just the highlighted area of slice 4 for *Default* is 23.6 dB while the corresponding PSNR for *Default+WNVMVA* is 23.4 dB. However, the PSNR values for the rest of the slice, which is more visually significant, are 25 dB and 25.4 dB respectively.



(a) Default (b) Default+WNVMVA

Figure 4.9: Error concealment comparison for top slice loss in High2 using (a) Default and (b) Default+WNVMVA

In conclusion, *Default+WNVMVA* works best for cases with intermittent lost slices (2int and 3int) as opposed to continuous lost slices (4cont) due to the relative differences in spatial MV correlation. In addition, the effect of *Default+WNVMVA* on top/bot cases relative to *Default* is similar to EC in low motion video, where more complex EC techniques are not necessary.

Tables 4.3, 4.4, and 4.5 show the percentage distribution of the EC techniques for the three MV candidate systems for 4cont, 3int, and 2int loss scenarios, respectively. The top and bottom slice loss scenarios are not included as there is no noticeable visual improvement by applying more sophisticated EC methods for these scenarios. These results show that WNVMVA offers significant improvement by contributing to approximately 10 to 20% in most cases. As a result, the impact of individual *Default* techniques is reduced in favor of WNVMVA. This shows that there is a considerable amount of spatial dependence between lost MVs and surrounding valid MVs. Furthermore, this trend continues when WNVMVA is further evaluated by including PI and DI. However, the effectiveness of WNVMVA is slightly reduced in *Default+WNVMVA+PI+DI*. This is because PI and DI create candidates that favor the choice of default EC techniques, especially *Last MV*. Nonetheless, PI and DI are

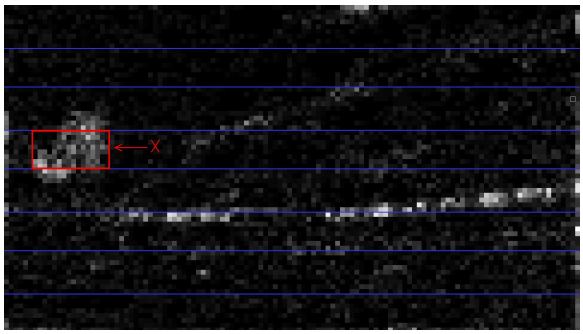


Figure 4.10: Heat map of MB sizes in the original version of the corrupted frame in High3. Brighter MBs are larger and thus more likely to be intra-coded. The high-energy region in slice 4 is highlighted and labeled *X*.

not used in the final EC scheme and they also contribute only approximately 0% (very few instances) to the candidate system and are thus not as effective as WNVMVA.

The visual significance of WNVMVA is shown in Fig. 4.11, which is compared against the default EC techniques by concealing a single corrupted frame shown in Fig. 4.3b. In addition, Fig. 4.12 shows the visual improvement in error propagation after 10 frames by employing *Default+WNVMVA* (Fig. 4.12e) over *Default* (Fig. 4.12b) for the 4cont case of Med1. Only a portion of the frame is shown in order to highlight the differences. In further experiments, PI and DI were individually substituted in place of WNVMVA as shown in Figs. 4.12c and 4.12d. The resultant visual results are inferior to *Default+WNVMVA*, which provides the best visual result.

The decoder may choose erroneous WNVMVA-based candidates due to the following two related reasons. First, the larger the lost slice region, the further apart and less spatially dependent the valid MVs are, resulting in poor WNVMVA. Second is the limited efficiency of BMA. WNVMVA-based MVs can end up generating MBs that have better BMA performance within their immediate surroundings but end up degrading overall video quality.

4.6 Conclusion and Future Work

The future of H.264 HD video over 802.11 wireless networks is bright. However, network losses necessitate EC, which can be performed through spatial MV recovery by estimation based on MVs available within the same frame. This paper proposed a new method called

Video	Method	Contribution of Individual EC Technique (%)										
		Zero	Left	Right	Top	Bottom	Mean	Median	Last	WNVMVA	Intra	Poly
Low1	Default	7.1	23.9	13.9	4.5	4.1	14.8	24.7	7.0	-	-	-
	Default+WNVMVA	6.2	18.7	12.7	4.2	3.2	13.7	21.1	4.2	16.0	-	-
	Default+WNVMVA+PI+DI	0.0	7.7	5.6	1.8	1.1	5.6	15.6	51.9	10.5	0.0	0.1
Low2	Default	3.2	25.2	12.6	4.3	4.8	11.6	29.5	8.8	-	-	-
	Default+WNVMVA	1.9	21.8	11.4	5.3	2.5	9.7	23.3	6.0	18.1	-	-
	Default+WNVMVA+PI+DI	0.0	8.9	5.9	2.7	1.3	6.5	18.2	44.8	11.7	0.0	0.0
Med1	Default	2.1	29.0	12.8	4.1	2.5	13.0	35.8	0.6	-	-	-
	Default+WNVMVA	1.1	28.6	12.4	4.4	2.5	12.7	34.7	0.3	3.3	-	-
	Default+WNVMVA+PI+DI	0.0	12.3	7.1	3.5	1.5	7.3	30.3	35.7	2.1	0.0	0.0
Med2	Default	2.1	31.4	10.2	4.0	2.2	12.2	31.1	6.7	-	-	-
	Default+WNVMVA	1.8	28.1	10.6	3.7	2.1	12.2	24.6	4.2	12.7	-	-
	Default+WNVMVA+PI+DI	0.0	9.3	5.9	2.4	0.7	6.3	18.0	49.6	7.8	0.0	0.0
High1	Default	1.6	32.2	9.7	3.3	1.8	11.2	33.0	7.3	-	-	-
	Default+WNVMVA	1.4	31.0	9.4	3.3	1.9	10.9	31.2	2.2	8.7	-	-
	Default+WNVMVA+PI+DI	0.0	10.7	6.4	3.0	1.5	6.3	22.4	42.4	7.3	0.0	0.0
High2	Default	5.2	28.2	14.6	3.4	3.2	11.5	30.0	4.0	-	-	-
	Default+WNVMVA	5.2	24.3	16.6	3.4	2.4	11.1	27.1	2.0	7.9	-	-
	Default+WNVMVA+PI+DI	0.0	7.5	6.4	1.5	1.1	5.8	16.9	50.5	10.2	0.0	0.1

Table 4.3: Percentage distribution of estimated MVs for Default, Default+WNVMVA, and Default+WNVMVA+PI+DI for 4cont.

Video	Method	Contribution of Individual EC Technique (%)										
		Zero	Left	Right	Top	Bottom	Mean	Median	Last	WNVMVA	Intra	Poly
Low1	Default	7.2	15.5	13.0	4.2	6.9	13.9	20.9	18.4	-	-	-
	Default+WNVMVA	6.4	12.3	12.3	4.8	3.2	13.9	16.4	7.6	23.2	-	-
	Default+WNVMVA+PI+DI	0.0	5.2	5.8	2.1	1.1	6.2	11.4	53.9	14.1	0.0	0.2
Low2	Default	4.2	23.7	12.6	5.1	5.8	12.4	27.2	8.9	-	-	-
	Default+WNVMVA	3.0	19.0	11.2	6.1	3.7	10.9	20.8	4.3	20.9	-	-
	Default+WNVMVA+PI+DI	0.0	8.3	6.9	2.5	1.5	5.5	15.2	45.8	14.2	0.0	0.1
Med1	Default	2.1	29.0	12.8	4.1	2.5	13.0	35.8	0.6	-	-	-
	Default+WNVMVA	1.0	23.9	11.0	6.3	4.1	9.9	38.0	1.2	4.6	-	-
	Default+WNVMVA+PI+DI	0.0	11.8	7.7	5.4	3.0	6.0	32.4	30.2	3.5	0.0	0.1
Med2	Default	0.4	24.4	11.7	6.5	4.3	11.5	29.6	11.7	-	-	-
	Default+WNVMVA	0.6	19.5	11.4	6.6	3.8	10.8	22.8	8.1	16.4	-	-
	Default+WNVMVA+PI+DI	0.0	8.6	6.9	3.0	1.7	5.2	18.9	42.4	13.3	0.0	0.0
High1	Default	1.5	27.8	11.1	6.1	3.5	9.9	34.0	6.0	-	-	-
	Default+WNVMVA	1.1	24.4	11.1	5.8	3.6	11.0	32.4	1.1	9.5	-	-
	Default+WNVMVA+PI+DI	0.0	9.6	6.1	3.1	1.7	4.0	21.4	41.8	12.2	0.0	0.1
High2	Default	1.9	25.0	13.6	4.7	3.9	10.7	34.2	6.0	-	-	-
	Default+WNVMVA	1.5	21.5	13.2	4.8	3.5	9.4	31.0	5.4	9.7	-	-
	Default+WNVMVA+PI+DI	0.0	9.9	8.2	2.5	1.7	5.1	23.0	39.3	10.2	0.0	0.0

Table 4.4: Percentage distribution of estimated MVs for Default, Default+WNVMVA, and Default+WNVMVA+PI+DI for 3int.

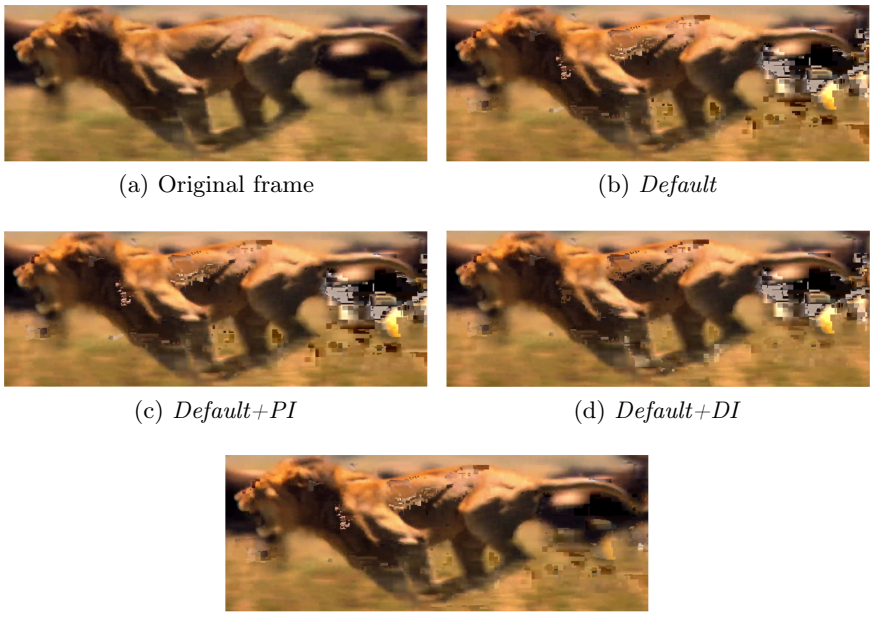
Video	Method	Contribution of Individual EC Technique (%)										
		Zero	Left	Right	Top	Bottom	Mean	Median	Last	WNVMVA	Intra	Poly
Low1	Default	8.3	18.9	15.2	4.8	4.0	15.0	22.5	11.3	-	-	-
	Default+WNVMVA	6.7	14.4	13.4	5.1	3.3	13.2	19.4	7.5	16.9	-	-
	Default+WNVMVA+PI+DI	0.0	6.2	6.4	2.4	1.2	6.0	14.0	52.1	11.6	0.0	0.2
Low2	Default	5.2	24.5	13.3	5.9	3.4	13.3	30.3	4.1	-	-	-
	Default+WNVMVA	3.8	19.0	11.9	6.0	3.4	11.9	22.1	4.7	17.2	-	-
	Default+WNVMVA+PI+DI	0.0	8.9	7.1	2.8	1.5	6.7	16.3	40.9	15.9	0.0	0.1
Med1	Default	1.4	25.5	9.9	8.0	4.1	7.5	42.4	1.1	-	-	-
	Default+WNVMVA	1.2	23.1	10.5	6.5	4.3	8.9	39.6	0.9	5.1	-	-
	Default+WNVMVA+PI+DI	0.0	14.4	8.2	6.2	3.6	5.4	36.2	22.4	3.5	0.0	0.1
Med2	Default	0.5	23.7	11.7	6.5	4.4	10.4	31.6	11.4	-	-	-
	Default+WNVMVA	0.5	18.2	11.4	6.5	4.1	10.4	24.2	8.4	16.4	-	-
	Default+WNVMVA+PI+DI	0.0	9.4	6.9	3.0	2.1	4.7	20.9	38.5	14.5	0.0	0.0
High1	Default	1.8	27.6	10.1	5.7	3.7	9.2	33.2	8.7	-	-	-
	Default+WNVMVA	1.3	24.8	10.1	5.5	3.6	10.2	31.8	1.5	11.1	-	-
	Default+WNVMVA+PI+DI	0.0	9.4	5.7	2.8	1.7	3.7	19.7	44.2	12.9	0.0	0.1
High2	Default	1.9	25.4	14.1	4.7	4.6	9.9	33.9	5.5	-	-	-
	Default+WNVMVA	1.6	21.6	13.6	4.8	4.2	7.8	32.1	5.7	8.6	-	-
	Default+WNVMVA+PI+DI	0.0	10.1	8.4	2.5	1.9	4.9	24.7	37.7	9.8	0.0	0.0

Table 4.5: Percentage distribution of estimated MVs for Default, Default+WNVMVA, and Default+WNVMVA+PI+DI for 2int.

(a) Surrounding MVs (b) Zero MV (c) Last MV

(d) Mean MV (e) Median MV (f) WNVMVA

Figure 4.11: WNVMVA compared against pre-existing EC techniques (surrounding MVs, zero MV, last MV, mean MV and median MV)



(a) Original frame (b) *Default*

(c) *Default+PI* (d) *Default+DI*

(e) *Default+WNVMVA*

Figure 4.12: Comparison between (a) the original frame and error propagation 10 frames after the corrupted frame for the 4cont case of Med1 concealed with (b) *Default*, (c) *Default+PI*, (d) *Default+DI*, and (e) *Default+WNVMVA*.

WNVMVA, which captures the spatial dependence among MVs by exclusively analyzing valid MVs. Our experiment showed that among the multiple candidates in CMVR for estimating lost MVs, WNVMVA was chosen about 10 to 20% of the time and provided up to 2 dB in PSNR improvement compared to common spatial EC techniques.

There are a number ways WNVMVA can be improved. For example, since intra-prediction modes in H.264 are directional gradients potentially related to the direction of lost MVs, neighboring intra-MBs can be considered by predicting their MVs. We also plan to investigate a temporal technique for MV recovery where estimated MVs can be forecasted using time series analysis. Temporal MV recovery can in turn supplement spatial MV recovery in a decision algorithm that chooses between spatial and temporal MV estimates. Alternatively, a unified spatial-temporal realization can be used.

Acknowledgements

This research was supported in part by LG Display Co., Korea and Ministry of Education Science and Technology (MEST) and the Korean Federation of Science and Technology Societies (KOFST).

Experimental Study of Low-Latency HD VoD Streaming using Flexible Dual TCP-UDP Streaming Protocol

Kevin Gatimu, Arul Dhamodaran, Taylor Johnson and Ben Lee

Proceedings of the 15th Annual IEEE Consumer Communications and Networking Conference (CCNC),
Las Vegas, NV, USA, January 12-15, 2018
https://doi.org/10.1109/CCNC.2018.8319234

Chapter 5: Experimental Study of Low-Latency HD VoD Streaming using Flexible Dual TCP-UDP Streaming Protocol

The Flexible Dual TCP-UDP Streaming Protocol (FDSP) combines the reliability of TCP with the low latency characteristics of UDP. FDSP delivers the more critical parts of the video data via TCP and the rest via UDP. Bitstream Prioritization (BP) is a sliding scale that is used to determine the amount of TCP data that is to be sent. BP can be adjusted according to the level of network congestion. FDSP-based streaming achieves lower rebuffering time and less rebuffering instances than TCP-based streaming as well lower packet loss than UDP-based streaming. Our implementation and experiments on a real testbed shows that FDSP with BP delivers high quality, low-latency video, which is especially suitable for live video and subscription-based video.

5.1 Introduction

Global Internet traffic is projected to increase nearly threefold until 2021, with video accounting for 82% of the total traffic [1]. Currently, consumer video is dominated by High Definition (HD), but higher resolutions such as 4K are gaining mainstream popularity [2]. Furthermore, there is an increasing number of video-capable devices and platforms being added globally everyday. For instance, the current 2 billion LTE subscribers are expected to double by 2021 [3]. Together, these factors will continue to increase global network congestion and pose even greater challenges to seamlessly delivering video at HD resolution and beyond.

This situation is further exacerbated by the unicast delivery model in major Video on Demand (VoD) services such as Netflix, Hulu, and Amazon Video, where each client requests video directly from a server. Therefore, as more clients connect to the server, the bandwidth requirements grow rapidly. VoD content providers have mitigated increased bandwidth demands by decentralizing their infrastructure through Content Delivery Networks (CDNs), which brings proxy servers closer to the end-user.

Another major development in managing VoD network resources is HTTP Adaptive

Streaming (HAS). In HAS, the client requests video from a selection of multiple quality versions based on its perceived network conditions. Several HAS implementations exist, including proprietary ones such as Microsoft Smooth Streaming (MSS) [4], Adobe HTTP Dynamic Streaming [5], Apple's HTTP Live Streaming (HLS) [6], and the open-source standard, Dynamic Adaptive Streaming over HTTP (DASH) [7].

However, even the combination of HAS and CDNs is challenged by extremely large audiences, resulting in high bandwidth requirements for Internet video content providers. This is especially the case for live video streaming for events such as sports (e.g., the Olympics and the World Cup) and presidential debates. Furthermore, HAS suffers from high latency – often 20 seconds or more [8]. This is because two or more substreams, typically 10 seconds each, need to be buffered prior to playout. Such initial startup delay is acceptable for pre-recorded content (e.g., movies) as this maximizes the client's video quality with reduced rebuffering. However, the latency for live events needs to be minimized. Low latency is also required for subscription-based live video services such as Internet Protocol television (IPTV). When a client switches between different channels of streaming video, the transition needs to be as close as possible to traditional broadcast television, with hardly any noticeable delay.

The Transmission Control Protocol (TCP) is the transport layer protocol used in HAS. When outstanding packets are acknowledged by the receiver, TCP additively increases the transmission rate of the sender by a constant amount. On the other hand, when acknowledgments are lost due to congestion, the sender retransmits the lost packets and halves the transmission rate. This is detrimental towards meeting playout deadlines for achieving low-latency video streaming. The User Datagram Protocol (UDP) is better suited for low-latency applications compared to TCP. As a result, there have been hybridization efforts at the transport layer in order to combine the reliability of TCP with the low latency of UDP, pioneered by *Reliable UDP* [9] and culminating in the more advanced *Quick UDP Internet Connections* (QUIC) [10]. However, QUIC has been shown to have higher protocol overhead than TCP at low bitrates [11]. UDP has also been useful from an infrastructural point-of-view by supplementing CDNs with UDP-based peer-to-peer (P2P) networks [12, 13].

Based on the aforementioned discussion, the objective of this paper is to show that low-latency VoD streaming can be achieved using a hybrid streaming protocol called *Flexible Dual Streaming Protocol* (FDSP). Our previous work showed that FDSP is suitable for

improving direct device-to-device streaming using simulation studies [14, 15, 16]. In this paper, FDSP is tailored for a physical testbed with network emulation for a VoD streaming environment. Our findings show that FDSP-based streaming achieves lower latency than pure-TCP-based streaming while having less packet loss than pure-UDP-based streaming.

5.2 Related Work

HAS is the most popular streaming mechanism for delivering Internet video today. For this reason, there has been research and development in trying to reduce the latency that is caused by video segmentation. A client maintains a video buffer of two or more segments of typically 10 seconds each [6, 17], which results in latency of 20 seconds or more. Reducing the segment size to just a few seconds can reduce the size of a client's playout buffer, which in turn reduces latency. However, this increases the total number of segments and, therefore, the number of HTTP requests that the client sends to the server in order to retrieve the video segments. These requests use precious bandwidth at a rate of one round-trip time (RTT) per video segment. For instance, a client that requests 2-second video segments on a network path with an RTT delay of 300 ms will experience 300 ms of additional delay every 2 seconds. In [18], Swaminathan *et al.* use HTTP chunked encoding to disrupt this correlation between live latency and segment duration by using partial HTTP responses. However, the persistent connections that are needed for chunked encoding transfer are prone to timeout issues and security concerns such as injection attacks and denial-of-service attacks [19]. Alternatively, HTTP/2 provides server push mechanisms such that the client receives multiple video segments per request [20, 21, 22]. However, HTTP/2 is not as widely available as legacy HTTP. HTTP/2 only has 15% worldwide deployment and, at a current growth rate of 5% additional coverage every year, it has a long way to go before becoming a widely recognized standard [23].

Other improvements in reducing video latency include modifications to the transport layer. For instance, Chakareski *et al.* used multiple TCP connections in conjunction with Scalable Video Coding (SVC) [24]. More important packets were transmitted via better quality TCP connections and were, therefore, less prone to retransmissions. While this method addresses delay within the transport layer, there is still significant delay in the application layer due to the typical video segment sizes in HAS. On the other hand, Houze *et al.* proposed a multi-path TCP streaming scheme based on the application layer, where

larger video frames were subdivided based on media container formats [25]. They were then transmitted across two concurrent TCP connections and reassembled by the client. However, this method uses HTTP chunked encoding.

Peer-to-peer (P2P) networks have been used to supplement CDNs and help content providers save on deployment and maintenance costs [26]. This also reduces HTTP requests made to CDN servers thus lowering the latency for live streaming [27, 28]. In fact, CDN caching increases delay by 15-30 seconds [29]. CDN-P2P architectures have been commercialized for some time now by global CDN companies such as ChinaCache [12] and Akamai [13]. These hybrid architectures primarily rely on CDNs for HTTP-based retrieval of initial or critical video segments while using P2P networks for bandwidth relief or for retrieving future segments. Even though the P2P networks are UDP-based, standardized NAT/firewall traversal for UDP-based transmission is gaining traction primarily through WebRTC [30], which is a collection of protocols and browser APIs.

This paper shows that FDSP-based streaming achieves much lower latency compared to HTTP-based streaming at comparable video quality levels. Our study also shows that FDSP transmission results in lower packet loss compared to UDP-based streaming, even in congested networks. Furthermore, FDSP is orthogonal to adaptive streaming and can thus be used as a transport protocol for today's segment-based video delivery systems.

5.3 FDSP Overview

This section provides an overview of FDSP, including its architectural features and video streaming using substreams. For more details, see [14], [15] and [16]. FDSP is a hybrid streaming protocol that combines the reliability of TCP with the low latency characteristics of UDP. Figure A.3 shows the FDSP architecture consisting of a server and a client.

At the server, the *H.264 Syntax Parser* processes video data in order to detect critical H.264 video syntax elements (i.e., Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and slice headers). The *MPEG-TS Packetizer* within the *Demultiplexer* (DEMUX) then encapsulates all the data according to the RTP MPEG-TS specification. The DEMUX then directs the packets containing critical data to a TCP socket and the rest to a UDP socket as *Dual Tunneling* keeps both TCP and UDP sessions simultaneously active during video streaming. The *BP Selection* module sets the Bitstream Prioritization (BP) parameter, which is a percentage of I-frame data that is to be sent via TCP in addition to the original

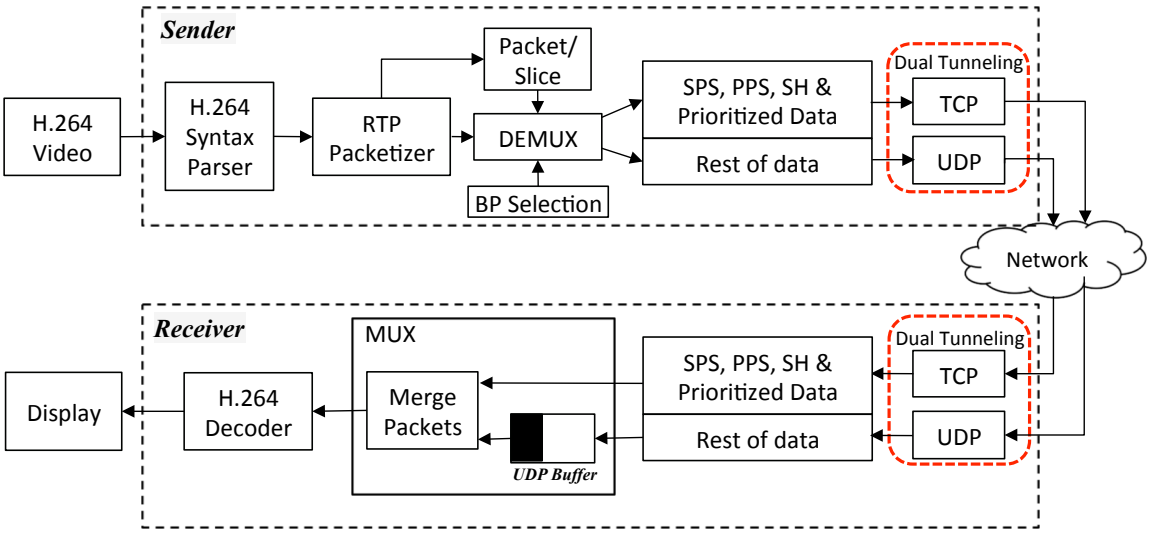


Figure 5.1: Flexible Dual TCP-UDP Streaming Protocol (FDSP) Architecture [14].

critical data. At the client, the *Multiplexer* (MUX) sorts TCP and UDP packets based on their RTP timestamps. This reordering is essential for the *H.264 Decoder* to decode incoming data correctly.

When a stream is initiated, the FDSP server transmits the packets for the first 10-second substream. All the TCP packets for this substream must be received (i.e., buffered) before playback begins. This startup delay (T_{init}) is low since only the TCP portion of the data is sent rather than the whole 10 seconds of video. In order to minimize rebuffering, the TCP packets for the next substream are sent at the same time as the UDP packets for the current substream through a process called *substream overlapping* as illustrated in Figure A.4. Substream overlapping is repeated throughout the duration of the stream. However, when playback for a particular substream is complete and the TCP packets for the upcoming substream are not yet all available, the client has to wait thus causing a rebuffering instance. The playout deadline for all subsequent packets is then incremented by the rebuffering time.

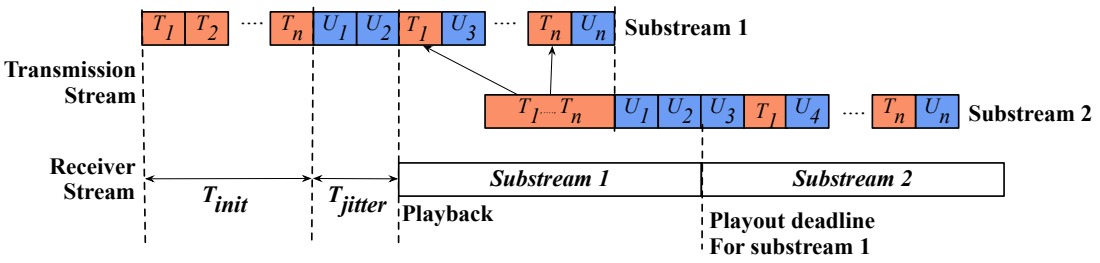


Figure 5.2: Substream overlapping. Each packet is either UDP (U) or TCP (T), where the subscript represents the packet number within a substream.

Figure 5.3: Experiment testbed.

5.4 Experiment Setup

Our experimental testbed is shown in Figure 6.3, which consists of a client-server pair and a traffic controller. The client-server pair is running *VLC Media Player* [31] on Mac OS X. The following modifications were made to integrate FDSP with BP into VLC:

1. Simultaneous streaming via UDP and TCP protocols.
2. Parsing H.264 video data at the server and subdividing it into TCP-bound and UDP-bound elements.
3. Reordering TCP and UDP packets and reconstructing the H.264 bitstream at the client prior to decoding.

The traffic controller, running on CentOS, connects the server to the client via a network bridge across interfaces `eth2` and `eth3`, respectively. The Linux *traffic control* (tc) utility was then used to perform traffic control on the network bridge. The *tc* configures the Linux

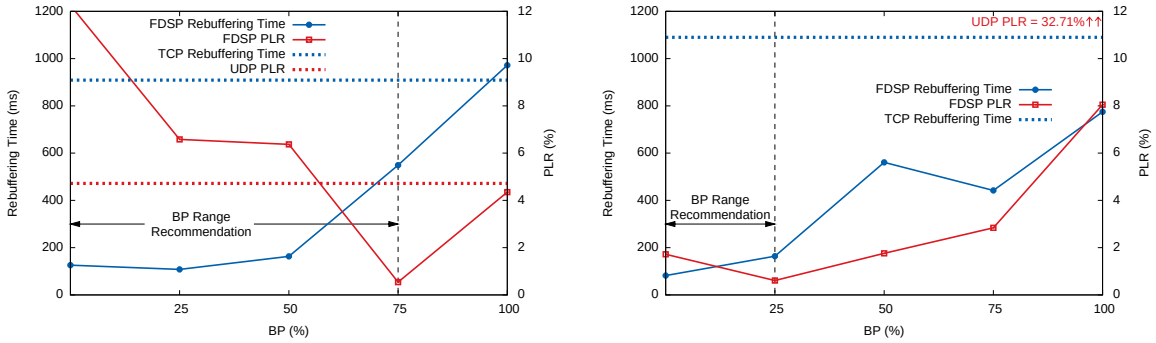
Parameters	Value(s)
Bridge interface	eth2, eth3
Delay (ms)	0, 25, 50, 75, 100, 125
Jitter (ms)	0, 5, 10, 15, 25
Loss	0.2%
Duplicate	0.2%
Corrupt	0.2%
Reorder	0.2%

Table 5.1: Network emulation settings for traffic control (tc).

kernel primarily through *queueing disciplines* (qdiscs). A *qdisc* is an interface between the kernel and a network interface, where packets are queued and released according to *tc* settings. For example, a loss setting drops packets from the *qdisc* according to a specified percentage, while a delay setting keeps the packets in the *qdisc* longer. Multiple settings can be used together. A summary of *tc* settings used in the experiments is shown in Table 6.1.

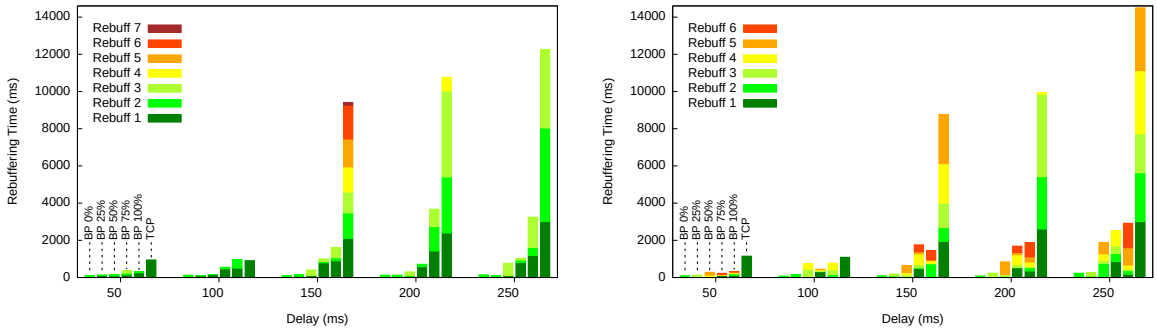
The *tc* parameters chosen represent an array of Wide Area Network (WAN) scenarios, which would typically plague Internet video streaming performance. The *Delay* setting was primarily used to simulate different levels of real-world Internet congestion [32]. The core network RTT latency is about 30 ms within Europe, 45 ms within North America, and 90 ms for Trans-Atlantic routes [33]. However, the edge network introduces additional latency. Therefore, *Delay* ranging from 0 to 125 ms in increments of 25 ms was used for each of the two bridged interfaces (`eth2` and `eth3`), resulting in a total RTT delay range of 0 to 250 ms. The corresponding random *Jitter* value was set at 20% of the delay. The *Duplicate* setting simulates duplicate packets, e.g., due to TCP retransmissions. The *Loss* setting simulates packets randomly dropped by the network. The *Corrupt* setting introduces a random bit error in a specified percentage of the packets. Finally, the *Reorder* setting simulates multi-hop routing by further delaying a specified percentage of packets according to the delay and jitter settings.

The test videos used for streaming are two full HD (1920×1080 @30fps) 30-second clips from an animation video, *Bunny*, and a documentary video, *Nature*. These videos are encoded using x264 with an average bit rate of 4 Mbps and four slices per frame. They are then streamed from the server to the client using FDSP, TCP, and UDP. For each streaming protocol, the five different levels of network congestion are created via the network delay settings (i.e., 50 ms, 100 ms, 150 ms, 200 ms, and 250 ms). Furthermore, FDSP-based



(a) *Nature* video (b) *Bunny* video

Figure 5.4: Rebuffering time and PLR for FSDP, TCP and UDP at 100 ms delay.



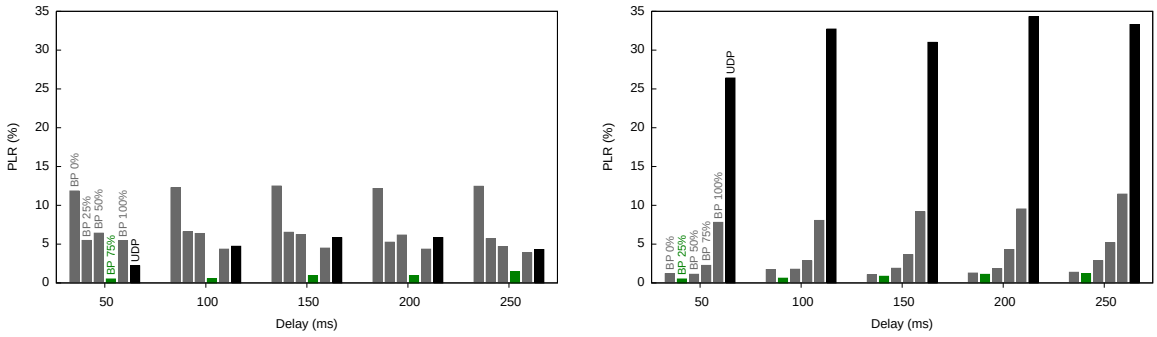
(a) *Nature* video (b) *Bunny* video

Figure 5.5: Rebuffering for different levels of network congestion for FDSP-based streaming at different values of BP and TCP-based streaming.

streaming is done for five different BP values (i.e., 0%, 25%, 50%, 75%, and 100%) per congestion level.

5.5 Results

This section discusses the results of our experiments. FDSP-based streaming generally outperforms TCP-based streaming in terms of both rebuffering time and number of rebuffering instances. FDSP also incurs lower PLR than UDP. Figure 6.4 shows a sample of the video



(a) *Nature* video (b) *Bunny* video

Figure 5.6: PLR for different levels of network congestion for FDSP-based streaming at different values of BP and UDP-based streaming.

streaming improvements of FDSP over either TCP or UDP at 100 ms delay. The other levels of network congestion show similar results. Overall, FDSP rebuffering time is significantly lower than TCP rebuffering time. In addition, as BP increases within a recommended range, PLR decreases. The BP range recommendations are 0% to 75% for *Nature* and 0% to 25% for *Bunny*. Since the overall rebuffering of FDSP-based streaming is significantly lower than that of TCP-based streaming, BP range recommendation was based on minimizing PLR. The rest of this section provides more details in the context of the two major improvements, i.e., lower rebuffering and lower PLR.

5.5.1 FDSP Improvement over TCP in Rebuffering

Reduction in both rebuffering time and instances is important towards improving the user's Quality of Experience (QoE). Figure 6.5 shows the total amount of rebuffering time and the number of rebuffering instances for the different levels of network congestion. For each congestion level, rebuffering is shown for FDSP with different values of BP as well for TCP. For instance, in *Nature* at 150 ms delay, FDSP rebuffering time ranges from 108 ms to 1,616 ms, compared to 9,410 ms in TCP. In addition, the number of rebuffering instances ranges from 2 to 3 for FDSP compared to 7 for TCP. Meanwhile, in *Bunny* at 150 ms delay, FDSP rebuffering time ranges from 92 ms to 1,441 ms with 1 to 6 instances, compared to 8,764 ms with 5 instances for TCP. Note that the first rebuffering instance (*Rebuff 1* in Figure 6.5) is

(a) UDP (b) Basic FDSP (0% BP)

Figure 5.7: Visual comparison between UDP-based streaming and FDSP-based streaming for *Bunny*.

the startup delay. As can be seen, FDSP exhibits lower startup delay than TCP at almost all BP levels.

While FDSP is significantly better than TCP in terms of rebuffering, it is important to note that rebuffering does increase with BP.

5.5.2 FDSP Improvement over UDP in PLR

FDSP-based streaming results in not only less rebuffering, but it also produces better video quality by reducing PLR. Figure 6.7 shows the effect of BP on PLR across different levels of network congestion for both *Nature* and *Bunny*. For each congestion level, PLR is shown for FDSP with different values of BP as well as for UDP. As BP increases, there is less PLR and thus better video quality. For *Nature*, the best BP value is 75% while for *Bunny* it is 25%. This implies that there is an optimal range of BP values based on the type of video.

As BP increases within the optimal range, more packets are sent via TCP rather than UDP. This protects them from network-induced losses. Since the bulk of PLR is due to lost UDP packets, the overall PLR decreases as BP increases. For example, in *Nature*, the PLR at 50 ms delay decreases from 9% to 0.32% as BP increases from 0% to 75%. Similarly, in *Bunny*, the PLR decreases from 1.19% to 0.51% as BP increases from 0% to 25%. Figure 6.8 shows a sample of the visual improvement of FDSP-based streaming with 0% BP over pure-UDP streaming in *Bunny*. The video frame in Figure 6.8a is intact while the frame in

Figure 6.8b shows the effects of packet loss under UDP-based streaming. In such situations, the loss of just a slice header or the first few bytes of a slice renders the rest of the slice data useless to the decoder, thus resulting in error concealment as shown in slice 4 of Figure 6.8b. On the other hand, FDSP-based streaming, even with no BP, protects slice headers through TCP transmission thus producing better quality video frames as shown in Figure 6.8a.

If BP surpasses the optimal range and becomes too high, the network can become saturated with TCP packets. This is because when there is network congestion, more packets are delayed, reordered or lost. The TCP packets are then more prone to retransmissions so as to guarantee in-order, reliable delivery. Meanwhile, the IP queue is filled with staged TCP and UDP packets. As the IP queue fills up with TCP packets, additional UDP packets are dropped. This is the cause of most of the PLR when BP becomes too high. In addition, some packets (both UDP and TCP) arrive at the client too late, past the decoder's playout deadline, and are thus also considered lost.

The frequency of I-frames can be used to categorize the type of video and determine the optimal range of BP. For videos such as *Bunny*, where there are many scene changes, there is usually a corresponding higher number of I-frames. In fact, there are 37 I-frames in *Bunny* compared to just 5 in *Nature*. Since I-frames contain significantly more data than other frames, the probability of network saturation increases with the frequency of I-frames, which leads to high PLR. For instance, Figure 6.7 shows much higher PLR for UDP-based streaming in *Bunny* (26.4%∼33.3%) compared to *Nature* (2.2%∼4.3%). In such scenarios (*Bunny*), small BP values (0%∼25%) are effective towards reducing PLR while higher values (>25%) will saturate the network with TCP packets from I-frame data.

In comparison, videos exemplified by *Nature* have lower PLR to begin with for UDP-based streaming. This is because of less network saturation as a result of lower I-frame frequency. When such videos are streamed through FDSP, the introduction of TCP packets increases the likelihood of network saturation and UDP PLR. However, higher BP values (up to 75% in the case of *Nature*) can be applied to the point of lowering UDP PLR below that of UDP-based streaming.

5.6 Conclusion and Future Work

This paper shows that FDSP with BP is suitable for low-latency HD video streaming over the Internet while maintaining high video quality by combining the reliability of TCP with the

low-latency characteristics of UDP. Our implementation and experiments on a real testbed consisting of a server and a client and an intermediate node for network emulation through the Linux traffic control utility showed that FDSP with BP results in significantly less rebuffering than TCP-based streaming and much lower PLR than UDP-based streaming.

As future work, BP will be dynamically adjusted with varying network conditions. A separate QoE study based on FDSP streaming is currently in progress. Its results will be used to determine when BP should be changed based on variation in PLR and rebuffering.

Experimental Study of QoE Improvements Towards Adaptive HD Video Streaming using Flexible Dual TCP-UDP Streaming Protocol

Kevin Gatimu, Arul Dhamodaran, Taylor Johnson and Ben Lee

Under review

Chapter 6: Experimental Study of QoE Improvements Towards Adaptive HD Video Streaming using Flexible Dual TCP-UDP Streaming Protocol

The Flexible Dual TCP-UDP Streaming Protocol (FDSP) combines the reliability of TCP with the low latency of UDP, thus providing transport layer improvements towards maintaining high QoE of multi-bitrate videos in adaptive streaming. FDSP delivers the more critical parts of the video data via TCP and the rest via UDP. FDSP also uses Bitstream Prioritization (BP), a sliding scale that determines the proportion of video data that is sent using TCP. BP can be adjusted according to the level of network congestion. FDSP-based streaming reduces total rebuffering time by over 90%, and rebuffering instances by 50% in many cases compared to TCP-based streaming. At the same time, packet loss reduces by over 75% for most BP levels compared to UDP-based streaming. In addition, FDSP-based streaming is potentially more suitable for adaptive streaming compared to the state-of-the-art TCP-based HTTP Adaptive Streaming (HAS), which is often plagued by high latency and high bandwidth requirements. In contrast, FDSP requires significantly less bandwidth than TCP in congested networks while exhibiting more stable client buffers.

6.1 Introduction

Global Internet traffic is projected to increase nearly threefold between 2016 and 2021, with video accounting for 82% of the total traffic, of which 13% will be live video [1]. Currently, consumer video is dominated by High Definition (HD), but higher resolutions such as 4K are gaining mainstream popularity, with up to 10% market penetration in the US alone [2]. Furthermore, there is an increasing number of video streaming devices and platforms being added globally everyday. For instance, the current 2.7 billion LTE subscribers are expected to double by 2023, including 1 billion 5G subscribers. All these factors will continue to increase global network congestion and pose even greater challenges to seamlessly delivering video at HD/4K resolution and beyond.

The unicast delivery model used in major Video on Demand (VoD) services such as Net-

flix, Hulu, and Amazon Video further exacerbates this situation. Since each client requests video directly from a server, the bandwidth requirements grow rapidly as the number of clients increases. VoD content providers have mitigated some of the increased bandwidth demands by decentralizing their infrastructure through Content Delivery Networks (CDNs), which brings proxy servers closer to end-users.

Another major development in streaming high quality video over networks with limited and varying bandwidth resources is *HTTP Adaptive Streaming* (HAS). In HAS, the client dynamically adjusts the video quality according to perceived network conditions by requesting video from a selection of different bitrate versions. That is, the higher the available bandwidth, the higher the selected video bitrate and its corresponding quality.

The HAS model introduces a number of factors that influence viewers' Quality of Experience (QoE). These include startup delay, rebuffering time and instances, bitrate switching frequency, and average video bitrate [3]. The main goal of improving QoE is minimizing startup delay and rebuffering as they have the greatest impact on viewers and are significantly affected by network congestion and thus latency [4]. Startup delay in HAS is often 20 seconds or more because two or more substreams, typically 10 seconds each, need to be buffered prior to playout [5]. Such a high startup delay is acceptable for pre-recorded content (e.g., movies) as this maximizes a client's video quality with reduced rebuffering. However, every 1-second increase in startup delay increases the video abandonment rate by 5.8%, and viewing time decreases by 5.02% when rebuffering exceeds just 1% of the video duration. [4]. Furthermore, low latency is critically important for live streaming as well as subscription-based live video services such as Internet Protocol television (IPTV), where channel switches need to be performed with hardly any noticeable delay.

Transmission Control Protocol (TCP) is the underlying transport protocol of HAS, and it provides transport services such as reliable in-order delivery, congestion control, and flow control. As a result, applications which rely on TCP often experience high latency, and this adversely affects the HAS model as well. On the other hand, *User Datagram Protocol* (UDP) is a low-latency alternative without the services provided by TCP. Our hybrid streaming protocol, called *Flexible Dual TCP-UDP Streaming Protocol* (FDSP), combines the reliability of TCP with the low latency of UDP through a simple application-layer combination, thus eliminating special network-layer modifications or additional protocols [6, 7, 8, 9]. This is especially important for media content providers who need to deploy videos to heterogenous networks and diverse devices. Our initial simulation studies of FDSP have shown that

it is effective in improving direct device-to-device (D2D) streaming in a wireless local area network [6]. The basic FDSP was then improved by adding *Bitstream Prioritization* (BP), where a percentage of more important elements of the H.264 bitstream were prioritized via TCP transmission [7, 8]. This was followed by a study using a client-server VoD testbed, which showed that FDSP-based streaming achieves lower latency and less packet loss than TCP-based and UDP-based streaming, respectively [9].

This paper extends the work in [9] to show that FDSP is a suitable protocol for future integration into the transport layer of today's overwhelmingly TCP-based adaptive streaming systems for VoD. Therefore, in addition to providing a discussion of FDSP in the context of video streaming server-client systems, for completeness, this paper presents a performance comparison among FDSP, TCP, and UDP for multiple bitrate versions of videos in congested networks. Our study shows that FDSP utilizes significantly less bandwidth resulting in better QoE than TCP for different video bitrate versions. Therefore, FDSP has the potential for improving adaptive streaming in the following ways:

1. FDSP can sustain a particular bitrate version of video longer than TCP in congested networks with less packet loss and rebuffering. This can decrease the frequency of bitrate switches and increase average video bitrate.

2. The FDSP client buffer is more stable than in TCP-based streaming. This provides the client with a more reliable measure for assessing the available bandwidth and developing more accurate buffer-based adaptation algorithms [10]. This is in accordance with the DASH implementation guidelines, which emphasizes the importance of a rate adaptation algorithm to smooth out fluctuations in available bandwidth [11].

The rest of this paper is organized as follows. Section A.3 provides a background of HAS, FDSP with BP, and UDP firewall traversal. Section B.2 discusses the related work. Section B.5 describes the experiment setup using a physical testbed. This is followed by a discussion of the results in Section B.6. Finally, Section A.6 concludes the paper and discusses possible future work.

6.2 Background

6.2.1 HAS

Several HAS implementations exist, including proprietary ones such as Microsoft Smooth Streaming (MSS) [12], Adobe HTTP Dynamic Streaming [13], and Apple's HTTP Live Streaming (HLS) [14], as well as the open-source standard Dynamic Adaptive Streaming over HTTP (DASH) [15]. In HAS, each video on the server is encoded into different bitrate versions called *representations*. Each representation is subdivided into 2∼10-second *segments*. The basic idea is for a client to send an HTTP request to a server for a segment whose encoded bitrate can be supported by the current available bandwidth. The client adapts to the varying available bandwidth using a *bitrate adaptation algorithm* to request segments from different representations. In general, the higher the available bandwidth, the higher the bitrate of the requested segment.

Bitrate adaptation algorithms can be broadly classified into three major categories: client-side, server-side, and network-level [16]. This paper will focus on client-side algorithms, which can be further classified into throughput-based, buffer-based, and hybrid [16]. In general, *throughput-based* methods select video bitrates according to bandwidth estimation while *buffer-based* methods do so based on a target client buffer occupancy. Hybrid methods are a combination of the two. There is a growing consensus on the greater importance of analyzing buffer occupancy compared to bandwidth estimation towards developing bitrate adaptation algorithms. For instance, Huang *et al.* demonstrated the ineffectiveness of bandwidth estimation [17], especially when there are competing flows, and proposed a buffer-based approach to bitrate adaptation [10]. Similarly, Spiteri *et al.* ignored bandwidth estimation in favor of buffer occupancy [18]. Furthermore, Yin *et al.* formulated an optimization model between buffer occupancy and bandwidth estimation, and found that their effectiveness for bitstream adaptation was limited by the latter [19]. Our study on FDSP-based streaming showed that it exhibits a more stable client buffer compared to TCP-based streaming. Therefore, FDSP provides a more reliable reference in the transport layer for designing better buffer-based bitrate adaptation algorithms. This is important, especially given the growing evidence of how unreliable bandwidth estimation can be.

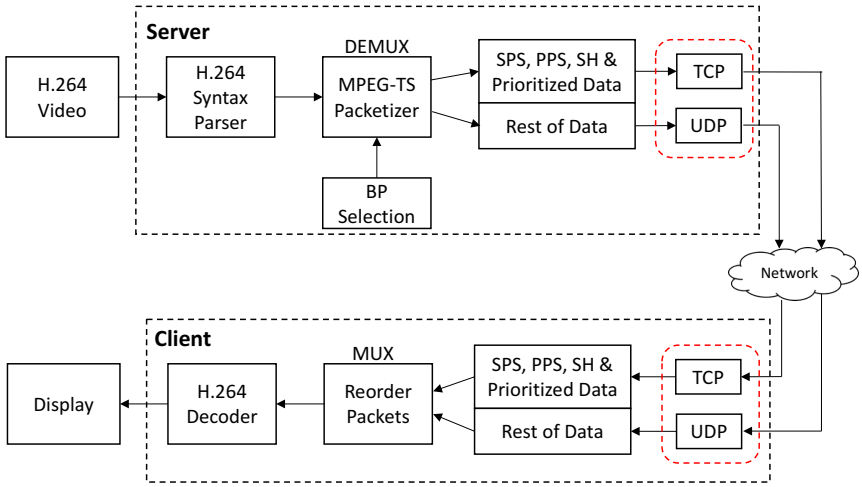


Figure 6.1: Flexible Dual TCP-UDP Streaming Protocol (FDSP) Architecture [6].

6.2.2 FDSP Overview

This section provides an overview of FDSP, including its architectural features and video streaming using substreams. For more details, see [6], [7] and [8]. FDSP is a hybrid streaming protocol that combines the reliability of TCP with the low latency characteristics of UDP. Figure A.3 shows the FDSP architecture consisting of a server and a client.

At the server, *H.264 Syntax Parser* processes video data in order to detect critical H.264 video syntax elements (i.e., Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and slice headers). *MPEG-TS Packetizer* within the *Demultiplexer* (DEMUX) module then encapsulates all the data according to the RTP MPEG-TS specification. DEMUX then directs the packets containing critical data to a TCP socket and the rest to a UDP socket as *Dual Tunneling* keeps both TCP and UDP sessions simultaneously active during video streaming. The *BP Selection* module sets the Bitstream Prioritization (BP) parameter, which is a percentage of I-frame data that is to be sent via TCP in addition to the original critical data. At the client, *Multiplexer* (MUX) sorts TCP and UDP packets based on their RTP timestamps. This reordering is essential for *H.264 Decoder* to decode incoming data correctly.

When a stream is initiated, the FDSP server transmits the packets for the first 10-second substream. All the TCP packets for this substream ($T_1 \ldots T_n$) must be received

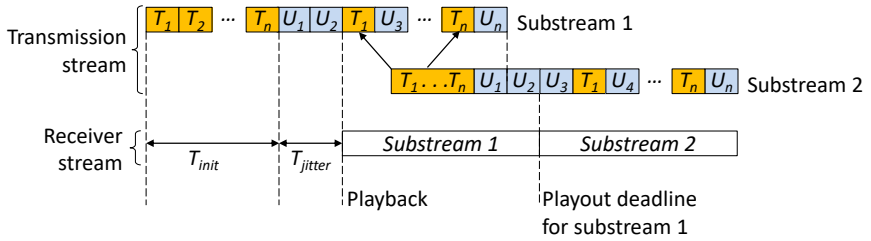


Figure 6.2: Substream overlapping. Each packet is either UDP (U) or TCP (T), where the subscript represents the packet number within a substream.

(i.e., buffered) before playback begins. This startup delay (T_{init}) is low since only the TCP portion of the data is sent rather than the whole 10 seconds of video. In order to minimize rebuffering, the TCP packets for the next substream are sent at the same time as the UDP packets ($U_1 \ldots U_n$) for the current substream through a process called *substream overlapping* as illustrated in the transmission stream section of Figure A.4. In this particular example, note that the TCP packets for substream 2 are all transmitted together with UDP packets of substream 1. This is done before transmitting the UDP packets for substream 2. Substream overlapping is repeated throughout the duration of the stream. However, when playback for a particular substream is complete and the TCP packets for the upcoming substream are not yet all available, the client has to wait, thus causing a rebuffering instance. The playout deadline for all subsequent packets is then incremented by the rebuffering time.

6.2.3 UDP Firewall Traversal

FDSP's TCP-UDP hybrid form is a useful and relevant technology for current innovations in HAS and video streaming transport in general. Even though HAS is primarily built on HTTP/TCP mainly due to HTTP's ability to traverse network address translators (NATs) or firewalls [20], UDP-based technologies are also critical to supplementing the latest HAS systems, e.g., as in CDN-P2P architectures (see Section B.2 for more details on UDP-based streaming). As a result, there exist several UDP firewall traversal techniques.

For example, Peer5 offloads up to 98% of CDN bandwidth [21] via Web Real-Time Communications (WebRTC), which is a popular open-source project that provides APIs for UDP-based peer-to-peer (P2P) connections. WebRTC provides a practical example of how UDP-based transport can traverse NATs. Similarly, UDP NAT Traversal can also be

extended to additional networking scenarios including the UDP portion of FDSP streaming servers and clients.

For instance, consider two hosts within individual NAT-protected private networks that wish to establish a UDP connection. This could be a video server attempting to send a client UDP data via FDSP streaming. An intermediate well-known globally reachable server can be used to establish UDP addresses and ports for both hosts prior to direct communication. This is achieved using protocols such as Session Traversal Utilities for NAT (STUN), Traversal Using Relay around NAT (TURN), and Interactive Connectivity Establishment (ICE) [22]. Each host requests a public IP address and port number from a STUN server. This creates an external NAT address that can be used for direct connections, including UDP, between the hosts. In some special cases (e.g., symmetric NATs), a proxy server connection for data transport is also needed via TURN. A host can build multiple IP and port pair candidates for connecting to other hosts by making a series of requests to a STUN server. Finally, ICE determines the best candidate for creating a connection. STUN is preferred over TURN since it is faster and does not require a relay service.

6.3 Related Work

TCP is the default transport protocol for HAS, but it exhibits shortcomings mainly due to latency. TCP's reliable transmission requires available bandwidth that is about twice the bitrate of video for satisfactory streaming performance [23]. In addition, the slow-start mechanism results in initial low throughput that requires pre-buffering as well as rebuffering when idle connections are restarted [24]. Furthermore, when congestion occurs, the sender retransmits TCP packets while halving the transmission rate. These factors result in low TCP throughput, which further jeopardizes low-latency streaming.

Since low latency is vital towards meeting the most significant QoE metrics in HAS, i.e., startup delay and rebuffering [4], several strategies have been proposed in order to either decrease latency or improve startup delay and rebuffering. For instance, reducing the segment size to just a few seconds is commonly used to decrease startup delay. However, this increases the total number of segments and, therefore, the number of client HTTP requests. These requests use precious bandwidth at a rate of one round-trip time (RTT) per video segment. For instance, a client that requests 2-second video segments on a network path with an RTT delay of 300 ms will experience 300 ms of additional delay every 2 seconds.

FDSP drastically decreases latency by transmitting most of the data via UDP rather than HTTP/TCP.

Chakareski *et al.* used multiple TCP connections in conjunction with Scalable Video Coding (SVC) [25] in order to decrease latency. Packets belonging to higher quality bitstreams in the SVC hierarchy were transmitted via better quality TCP connections. Therefore, these packets were less prone to retransmissions thus reducing delay in the transport layer. However, there is still potential for significant delay in the application layer due to buffering video segments (typically 10-second). FDSP's dual streaming significantly reduces application layer delay by only buffering the TCP portions of future segments while streaming the UDP portion of the current segment. In addition, FDSP's implementation is orthogonal to multipath-TCP schemes.

Swaminathan *et al.* used HTTP chunked transfer encoding to disrupt the correlation between latency and segment duration, particularly in live streaming [26]. This was done by using partial HTTP responses rather than waiting for complete responses (i.e., full segments) to be generated by the server. Houze *et al.* also used HTTP chunked encoding, but to supplement an application layer multi-path TCP streaming scheme [27]. Here, video frames were subdivided in proportion to network path speeds and reassembled by the client, thereby reducing latency and rebuffering. However, HTTP transfer chunked encoding can lead to extra overhead due to the increased volume of HTTP transfers. This is especially the case in congested networks, where timeout issues can occur when complete HTTP responses are not assembled punctually. Rather than requiring chunked encoding, FDSP is readily compatible with basic HTTP while still reducing latency and rebuffering. At the same time, HTTP chunked encoding is an orthogonal issue and could optionally be implemented on FDSP.

Alternatively, HTTP/2 provides server push mechanisms that allow the client receives multiple video segments per request instead of just a single segment [28, 29, 30]. This reduces the overall time needed for the client-server request-response mechanism, thus reducing latency. However, HTTP/2 is not as widely available as the more established HTTP/1.1. HTTP/2 only has 15% worldwide deployment and, at a current growth rate of 5% additional coverage every year, it has a long way to go before becoming a widely recognized standard [31]. However, FDSP is a simple application-layer combination of UDP and TCP, making it compatible with HTTP/1.1. FDSP's straightforward implementation also makes it compatible with HTTP/2.

UDP is well-suited for low-latency applications as it lacks the extra overhead necessary for TCP's features, such as flow control and reliable delivery. However, UDP's simplicity can result in packet loss, especially in congested networks. In addition, the lack of congestion control can lead to depletion of network resources due to bandwidth over-utilization, placing UDP out of favor compared to TCP in the broader Internet landscape. Nevertheless, UDP's low-latency offers an attractive option for contemporary HAS systems by supplementing CDNs with UDP-based P2P networks [32, 33]. This helps content providers lower deployment and maintenance costs [34]. At the same time, P2P networks improve live streaming latency by decreasing HTTP requests made to CDN servers [35, 36]. In fact, CDN caching can increase live streaming delay by 15-30 seconds [37]. CDN-P2P architectures have been commercialized for some time now by companies such as ChinaCache [32] and Akamai [33]. These hybrid architectures primarily rely on CDNs for HTTP-based retrieval of initial or critical video segments while using P2P networks for bandwidth relief or for retrieving future segments. Similarly, FDSP prioritizes the more important parts of the video bitstream via TCP while offloading the rest to UDP. Therefore, it can be integrated into a CDN-P2P-like framework, where the UDP portion of FDSP, in particular, can be reserved for a P2P network.

The work closest to ours are hybridization efforts at the transport layer in order to supplement the low latency and low overhead of UDP with TCP-like features [38, 39, 40, 41]. Velten *et al.* initially proposed the *Reliable Data Protocol* [38], which was designed to be a minimal variation of TCP for bulk data transfer with simplified flow control, buffering, and connection management. Bova and Krivoruchka followed this with the improved *Reliable UDP* (RUDP) [39]. RUDP extends UDP mainly by making some features mandatory, e.g., packet retransmissions, in-order delivery, and flow control. However, it is not standardized and is primarily limited to specific tasks, e.g., Microsoft's proprietary version for its TV software, Mediaroom [40]. There are also several RUDP-like protocols, but they are mostly application-specific. For example, Floyd *et al.* proposed the *Datagram Congestion Control Protocol* (DCCP), which adds congestion control to streaming media but without reliable in-order delivery [41]. FDSP is also a hybrid streaming protocol, in that the services it provides fall somewhere between pure TCP and pure UDP. However, FDSP has an advantage over other hybrid methods because it simply uses the existing TCP and UDP protocols without any modification to either.

More recently, Google has done work on an experimental transport protocol built on

Figure 6.3: Experiment testbed. The client consumes video provided by the server, while the traffic controller sets the level of network congestion.

UDP called *Quick UDP Internet Connections* (QUIC) [42]. Its main goal is to improve the performance of TCP-based applications by reducing latency and connection time. This is achieved by customizing UDP with encryption support, real-time bandwidth estimation, and bitstream compression. However, QUIC has been shown to have higher protocol overhead than TCP at low video bitrates [43]. On the other hand, FDSP demonstrates good performance across a wide range of video bitrates, including low ones.

6.4 Experiment Setup

As in our previous work [9], the experimental testbed is shown in Figure 6.3, which consists of a client-server pair and a traffic controller. The client and the server are each running *VLC Media Player* [44] on Mac OS X. The following modifications were made to integrate FDSP with BP into VLC:

1. Simultaneous streaming via UDP and TCP protocols.
2. Parsing H.264 video data at the server and subdividing it into TCP-bound and UDP-bound elements.
3. Reordering TCP and UDP packets and reconstructing the H.264 bitstream at the client prior to decoding.

The traffic controller, running on CentOS, connects the server to the client via a network bridge across interfaces `eth2` and `eth3`, respectively. The Linux *traffic control* (*tc*) utility

Parameters	**Value(s)**
Bridge interface	eth2, eth3
Delay (ms)	25, 50, 75, 100, 125
Jitter (ms)	5, 10, 15, 20, 25
Loss	0.2%
Duplicate	0.2%
Corrupt	0.2%
Reorder	0.2%

Table 6.1: Network congestion settings for for *tc*.

is then used to perform traffic control on the network bridge, which, in turn, sets the available bandwidth. The *tc* configures the Linux kernel primarily through *queueing disciplines* (*qdiscs*). A *qdisc* is an interface between the kernel and a network interface, where packets are queued and released according to *tc* settings. These settings are then used to create different levels of network congestion. For example, a loss setting drops packets from the *qdisc* according to a specified percentage, while a delay setting prolongs the duration the packets spend in the *qdisc*.

This testbed provides a physical platform for two sets of experiments. The first is a fundamental video streaming comparison among the relevant protocols, i.e., FDSP, UDP, and TCP. This is used to corroborate our findings in favor of FDSP from our simulation-only device-to-device studies [6, 7, 8]. The second set of experiments are then used to explore the advantages of FDSP over TCP towards future integration into HAS. The rest of this section details each set of experiments, while the corresponding results are discussed in Section B.6.

6.4.1 Basic Comparison of FDSP, UDP and TCP

A summary of *tc* parameters used to compare the three protocols is shown in Table 6.1. The values chosen represent an array of Wide Area Network (WAN) scenarios that would typically plague Internet video streaming performance. The *Delay* setting, in conjunction with the other settings, was primarily used to generate five different levels of real-world Internet congestion [45]. The core network RTT latency[1] is about 30 ms within Europe, 45 ms within North America, and 90 ms for Trans-Atlantic routes [46]. However, edge networks

[1] *RTT latency* and *RTT delay* are used interchangeably.

introduce additional latency. Therefore, *Delay* ranging from 25 to 125 ms in increments of 25 ms was used for each of the two bridged interfaces (`eth2` and `eth3`), resulting in a total RTT latency of 50 to 250 ms in increments of 50 ms. This corresponds to five different levels of congestion dictated by RTT latency, i.e., 50 ms, 100 ms, 150 ms, 200 ms, and 250 ms. In addition, for each delay setting, the corresponding random *Jitter* value was set at 20% of the delay, while the *Duplicate* setting generates duplicate packets, e.g., due to retransmissions. The *Loss* setting causes a minimum ratio of packets to be randomly dropped by the network. The *Corrupt* setting introduces a random bit error in a specified percentage of the packets. Finally, the *Reorder* setting simulates multi-hop routing by further delaying a specified percentage of packets according to the *Delay* and *Jitter* settings. For brevity, the *Delay* setting will be used to represent the 6-tuple settings.

The test videos used for streaming were two full HD (1920×1080 @30fps) 30-second clips from an animation video, *Bunny*, and a documentary video, *Nature*. These videos are encoded using x264 with an average bit rate of 4 Mbps and four slices per frame. They were then streamed from the server to the client using FDSP, TCP, and UDP. For each streaming protocol, the five different levels of network congestion were created via the network congestion settings. Furthermore, FDSP-based streaming was done for five different BP values (0%, 25%, 50%, 75%, and 100%) per congestion level. The general structure of the *tc* command applied to each interface is illustrated in the example below.

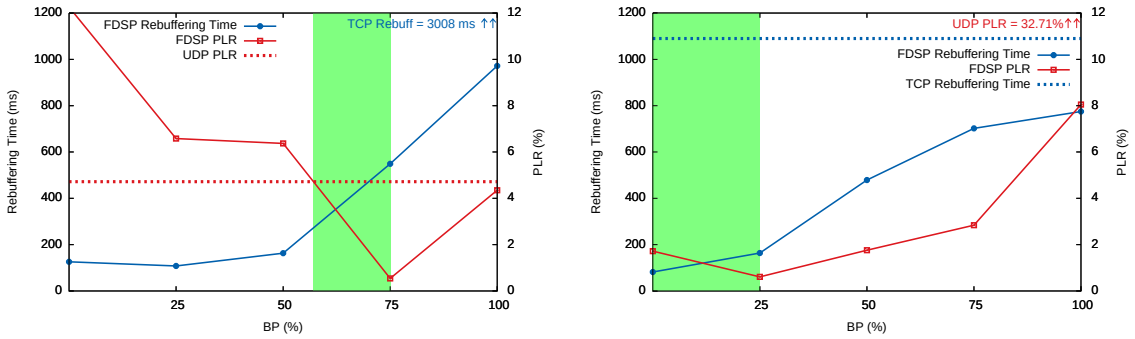
```
tc qdisc add dev eth2 root netem delay 50 ms 10 ms loss 0.2% duplicate 0.2%
corrupt 0.2% reorder 0.2%
```

6.4.2 Comparison of FDSP and TCP in Multi-Bitrate Streaming

For multi-bitrate streaming, the *tc* settings were chosen to control the amount of available bandwidth corresponding to different video bitrates. The general structure of the *tc* command applied to each interface is given by the example below:

```
tc qdisc add dev eth2 root tbf rate 5mbit latency 50ms burst 625
```

The token bucket filter (`tbf`) is a packet queue model that releases tokens according the ratio between the `rate` parameter, i.e., the desired available bandwidth, and the kernel



(a) *Nature* video (b) *Bunny* video

Figure 6.4: Rebuffering time and packet loss ratio (PLR) for FDSP, TCP, and UDP at 100 ms delay.

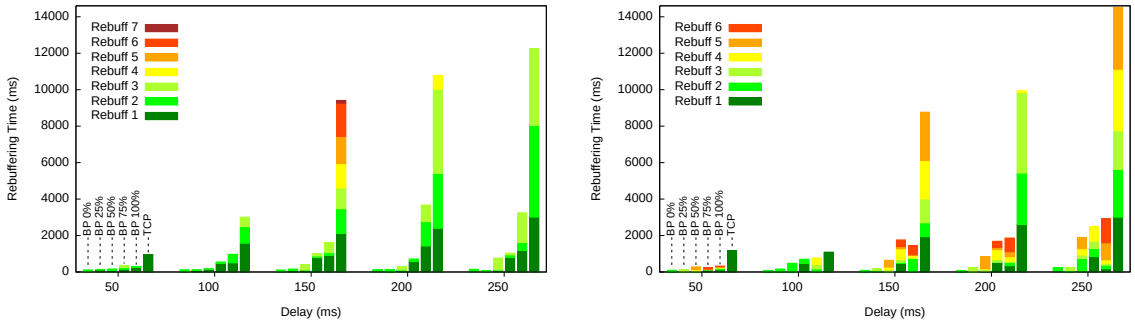
frequency. The `burst` parameter must be set to at least this ratio in bytes. In this example, the `rate` is 5 Mbps and the kernel frequency is 1000 Hz for our testbed, which yields a `burst` of 625 bytes. A packet at the head of the queue is transmitted once there are enough tokens corresponding to the packet size in bytes.

The test videos consist of three sets of full HD (1920× 1080 @30fps) clips (approximately 2.5 minutes in length). They are *Bunny2* (an animation), *Bears* (a documentary), and *Hobbit* (a movie trailer). Each video set includes three bitrate representations of 1 Mbps, 2 Mbps, and 4 Mbps. These videos were also encoded using x264 with four slices per frame. The videos were then streamed using FDSP and TCP in four different available bandwidth settings as shown in Table 6.2, which can be categorized as static and dynamic. The static settings indicate constant available bandwidth, while the dynamic settings include upper and lower limits of available bandwidth that oscillate with a 4-second duty cycle, which is similar to the recommended network profiles provided by the DASH Industry Forum Guidelines [47].

Each category is further subdivided into congested and non-congested settings. Although our results are primarily focused on congested settings, the experiments were also repeated for non-congested settings for completeness. In our prior simulation studies, congestion was set at the average bitrate of the video, which proved sufficient. However, in our physical testbed, this severely hampered streaming, particularly with the addition of the traffic control `latency` parameter, which specifically refers to network latency. Our experi-

	Encoded Bitrate		
	1 Mbps	2 Mbps	4 Mbps
Static congested	1.25	2.5	5
Dynamic congested	1-1.5	2-2.5	4-6
Static non-congested	2	4	10
Dynamic non-congested	1.75-2.25	3-5	7.5-12.5

Table 6.2: Available bandwidth settings (Mbps) for comparing FDSP to TCP



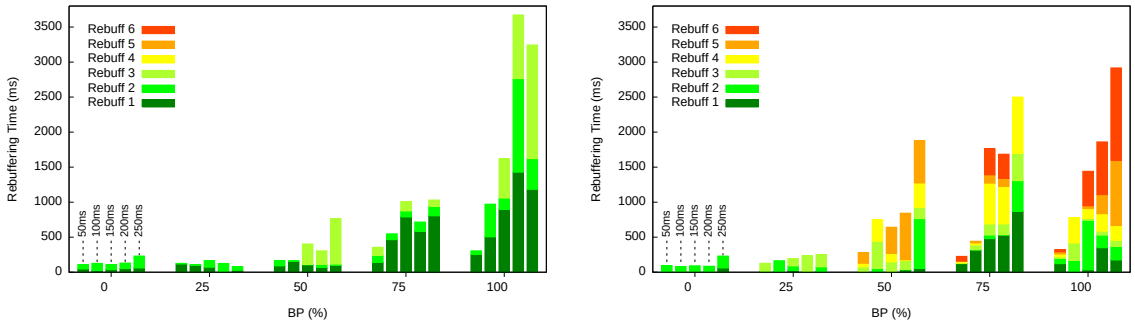
(a) *Nature* video (b) *Bunny* video

Figure 6.5: Rebuffering for different levels of network congestion for FDSP-based streaming at different values of BP and TCP-based streaming.

ments showed that available bandwidth set at 25% above the average video bitrate provided sufficient congestion for making useful comparisons between FDSP and TCP.

6.5 Results

This section discusses the results of our experiments. Section 6.5.1 analyzes the improvements of FDSP relative to both TCP and UDP in rebuffering and packet loss ratio (PLR) as BP increases from 0% to 100%. The corresponding tradeoffs are also discussed in detail. In summary, as BP increases, PLR decreases while rebuffering increases. Then, Section 6.5.2 shows why FDSP is more suitable than TCP for multi-bitrate streaming and, consequently, its potential as a transport protocol for improved adaptive streaming.



(a) *Nature* video. (b) *Bunny* video.

Figure 6.6: An in-depth look at rebuffering time for different levels of network congestion at different values of BP. TCP has been omitted here as it has much higher rebuffering than FDSP.

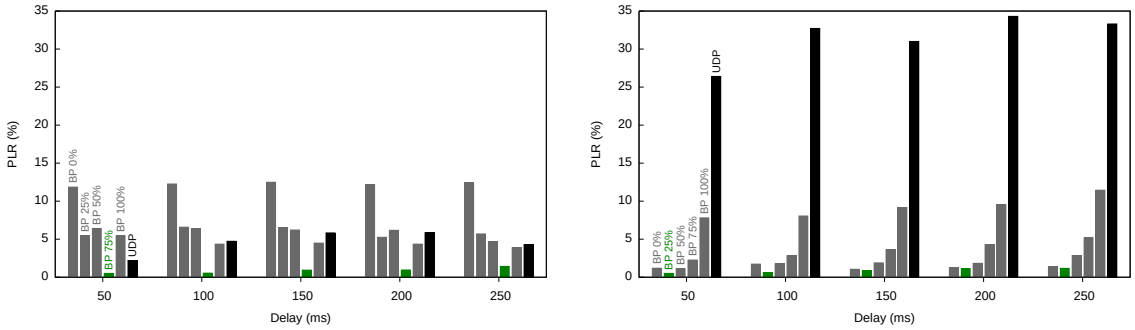
6.5.1 FDSP Improvement over both UDP and TCP

Figure 6.4 shows an example of basic video streaming improvements of FDSP over TCP and UDP at 100 ms RTT delay for *Nature* and *Bunny*. Note that this and every other RTT delay setting is accompanied by corresponding *tc* parameters as described in Table 6.2 and Section 6.4.1. In general, FDSP rebuffering time is significantly lower than TCP rebuffering time even though it increases with BP. At the same time, PLR also decreases within a particular range of BP. The other levels of network congestion chosen for our experiments (i.e., 50 ms, 150 ms, 200 ms and 250 ms) show similar results.

The rest of this subsection provides more details on results that demonstrate the fundamental improvements of FDSP over UDP and TCP, i.e., lower rebuffering and lower PLR, as well as the fact that there is an optimal range of BP that provides these improvements.

6.5.1.1 FDSP Improvement over TCP in Rebuffering

Reducing both rebuffering time and the number of rebuffering instances is important towards improving the user's QoE. Figure 6.5 shows the total amount of rebuffering time and instances for the different levels of network congestion. For each congestion level, rebuffering for FDSP is shown with different values of BP as well for TCP. For instance, for *Nature* with 150 ms RTT delay, FDSP rebuffering time ranges from 108 ms to 1,616 ms, compared



(a) *Nature* video (b) *Bunny* video

Figure 6.7: PLR for FDSP-based streaming at different values of BP and UDP-based streaming for different levels of network congestion.

to 9,410 ms in TCP. In addition, the number of rebuffering instances ranges from 2 to 3 for FDSP compared to 7 for TCP. Meanwhile, for *Bunny* with 150 ms RTT delay, FDSP rebuffering time ranges from 92 ms to 1,441 ms with 2 to 6 instances, compared to 8,764 ms with 5 instances for TCP. The rest of the rebuffering results for the two videos are summarized in Tables 6.3 and 6.4. Note that the first rebuffering instance (*Rebuff 1*) is the startup delay. As can be seen, FDSP exhibits lower startup delay than TCP for all BP values.

While FDSP is significantly better than TCP in terms of rebuffering, it is important to note that rebuffering does increase with BP. This is because higher BP values result in more TCP data, which increases the chance of retransmissions and thus rebuffering. A closer look at the behavior of just FDSP is illustrated by Figure 6.6, which shows the rebuffering time and instances across the different network congestion levels for each BP value.

6.5.1.2 FDSP Improvement over UDP in PLR

FDSP-based streaming results in not only less rebuffering, but better video quality by reducing PLR. Figure 6.7 shows the effect of BP on PLR across different levels of network congestion for both *Nature* and *Bunny*. For each congestion level, PLR is shown for FDSP with different values of BP as well as for UDP. PLR can be minimized by increasing BP, which leads to better video quality. For example, in *Nature*, the PLR for 50 ms RTT delay decreases from 9% to 0.32% as BP increases from 0% to 75%. Similarly, in *Bunny*, the PLR

(a) Basic FDSP (0% BP) (b) UDP

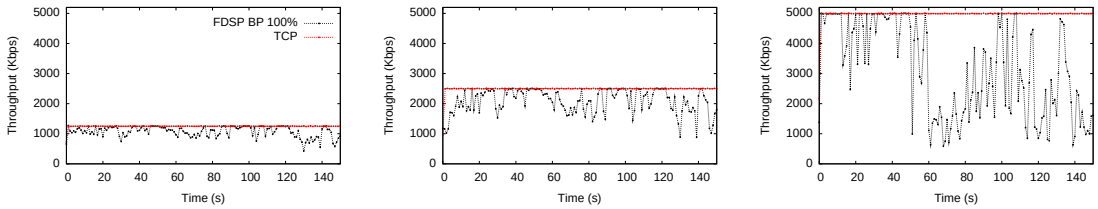
Figure 6.8: Visual comparison between FDSP-based streaming and UDP-based streaming for *Bunny*.

decreases from 1.19% to 0.51% as BP increases from 0% to 25%. In addition, this implies that there is an optimal range of BP operation based on the type of video as shown in Figure 6.4. As BP increases, more packets are sent via TCP rather than UDP. This protects them from network-induced losses. Since PLR is due to lost UDP packets, the overall PLR decreases as BP increases from 0% through the optimal range. Further details are discussed in Section 6.5.1.3.

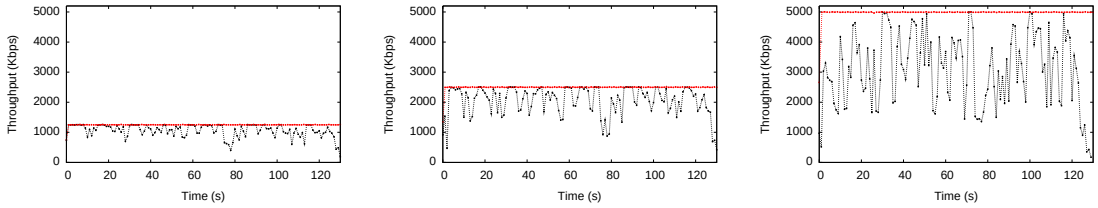
Figure 6.8 shows a sample of the visual improvement of FDSP-based streaming with 0% BP over UDP-based streaming in *Bunny*. The video frame in Figure 6.8a is intact while the frame in Figure 6.8b shows the effects of packet loss under UDP-based streaming. In such situations, the loss of just a slice header or the first few bytes of a slice renders the rest of the slice data useless to the decoder, even if properly received. This results in the decoder performing error concealment on the lost data as shown in slice 4 of Figure 6.8b. On the other hand, FDSP-based streaming, even at 0% BP, protects at least the slice headers through TCP transmission, thus making any available slice data useful to the decoder. As a result, FDSP produces better quality video frames as shown in Figure 6.8a.

6.5.1.3 Optimal Range of BP

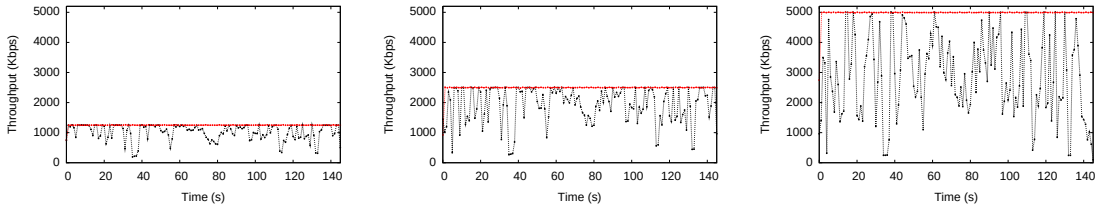
Since overall rebuffering time is significantly lower in FDSP than TCP across all BP values as shown in Figure 6.4, there is an optimal range of BP values for decreasing PLR as shown.



(a) Bunny2 – 1 Mbps, 1.25 Mbps bw (b) Bunny2 – 2 Mbps, 2.5 Mbps bw (c) Bunny2 – 4 Mbps, 5 Mbps bw



(d) Bears – 1 Mbps, 1.25 Mbps bw (e) Bears – 2 Mbps, 2.5 Mbps bw (f) Bears – 4 Mbps, 5 Mbps bw



(g) Hobbit – 1 Mbps, 1.25 Mbps bw (h) Hobbit – 2 Mbps, 2.5 Mbps bw (i) Hobbit – 4 Mbps, 5 Mbps bw

Figure 6.9: Throughput for FDSP-based streaming with 100% BP under static congested conditions (bw = bandwidth).

If BP surpasses the optimal range and becomes too high, the network can become saturated with TCP packets. This will cause more packets to be delayed, reordered, or lost when there is network congestion. The TCP packets are then more prone to retransmissions so as to guarantee in-order, reliable delivery. Meanwhile, the IP queue at the sender is filled with staged TCP and UDP packets. These additional TCP packets in the IP queue then cause subsequent UDP packets to be dropped. This is the cause of most of the PLR when BP becomes too high. Note that additional PLR also occurs when some UDP packets arrive at the client too late, past the decoder's playout deadline, and are thus also considered lost.

The frequency of I-frames can be used to categorize the type of video, and therefore

determines the optimal range of BP. Videos such as *Bunny* are characterized by fast motion, many scene changes, and a corresponding higher number of I-frames. In fact, there are 37 I-frames in *Bunny* compared to just 5 in the low-motion *Nature*. UDP-based streaming PLR in low-motion videos is much lower than in high-motion videos. For instance, as shown in Figure 6.7, *Nature* has 2.2%∼4.3% UDP-based streaming PLR across all congestion levels compared to 26.4%∼33.3% in *Bunny*. This is because *Nature*'s lower I-frame frequency reduces the likelihood of making network saturation worse.

However, when low-motion videos are streamed via FDSP, the introduction of TCP packets at low BP values increases network saturation, thus increasing PLR. However, applying higher BP values (up to 75% in the case of *Nature*) lowers PLR significantly below that of UDP-based streaming. On the other hand, low BP values (0%∼25% for *Bunny*) are effective towards minimizing PLR for high-motion videos. In both cases, BP values beyond the optimal range (>25% for *Bunny* and >75% for *Nature*) will tend to saturate the network with TCP packets containing I-frame data and increase PLR. Nevertheless, PLR in this range is still less than that of UDP-based streaming. Determining an optimal BP range that minimizes PLR while keeping rebuffering low based on the type of video will be left as future work.

6.5.2 FDSP Improvement over TCP for Multi-bitrate Streaming

FDSP-based streaming improves the transmission of multi-bitrate video representations for HAS in two ways: (1) lowers bandwidth requirements compared to TCP-based streaming and (2) maintains a more stable client buffer occupancy. The lower bandwidth requirement makes FDSP-based streaming more suitable in congested networks. This also lays a foundation for more fairness when there is additional traffic, including multiple video streams. At the same time, a more stable client buffer provides a reliable reference for developing buffer-based adaptation algorithms for adaptive streaming. The following discusses these two improvements in more detail.

6.5.2.1 Lower Bandwidth Requirement

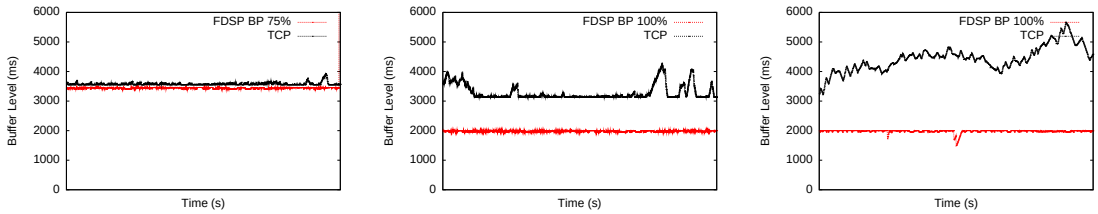
Figure 6.9 shows the throughput results for the three sets of videos streamed via FDSP with 100% BP in static congested scenarios. The throughput profiles for the BP values 0%,

Delay (ms)	Protocol	Rebuff 1	Rebuff 2	Rebuff 3	Rebuff 4	Rebuff 5	Rebuff 6	Total (ms)	Instances
	0%	8	85	-	-	-	-	93	2
	25%	12	2	114	-	-	-	128	3
50	50%	8	6	62	52	151	-	279	5
	75%	122	3	7	23	3	68	226	6
	100%	122	76	20	42	31	31	322	6
	TCP	1147	-	-	-	-	-	1147	1
	0%	9	73	-	-	-	-	82	2
	25%	10	154	-	-	-	-	164	2
100	50%	6	473	-	-	-	-	479	2
	75%	465	237	-	-	-	-	702	2
	100%	9	159	249	358	-	-	775	4
	TCP	1090	-	-	-	-	-	1090	1
	0%	9	83	-	-	-	-	92	2
	25%	25	65	105	-	-	-	195	3
150	50%	7	5	135	120	375	-	642	5
	75%	479	51	158	584	118	371	1761	6
	100%	34	706	26	142	40	493	1441	6
	TCP	1941	755	1309	2114	2645	-	8764	5
	0%	8	77	-	-	-	-	85	2
	25%	9	3	226	-	-	-	238	3
200	50%	33	5	129	11	665	-	843	5
	75%	532	3	155	537	115	342	1684	6
	100%	349	180	56	250	273	752	1860	6
	TCP	2611	2813	4420	103	-	-	9947	4
	0%	60	171	-	-	-	-	231	2
	25%	8	68	177	-	-	-	253	3
250	50%	54	710	165	345	600	-	1874	5
	75%	869	437	394	801	-	-	2501	4
	100%	175	196	85	210	934	1317	2917	6
	TCP	3009	2626	2109	3376	3377	-	14497	5

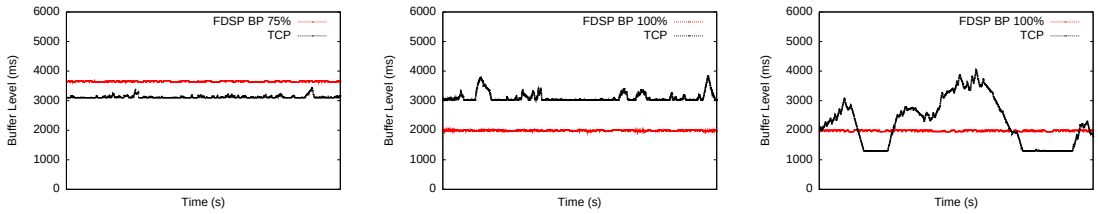
Table 6.3: Rebuffering time and instances for *Bunny*. The percentage entries under Protocol represent BP values for FDSP.

Delay (ms)	Protocol	Rebuff 1	Rebuff 2	Rebuff 3	Rebuff 4	Rebuff 5	Rebuff 6	Rebuff 7	Total (ms)	Instances
	0%	46	64	-	-	-	-	-	110	2
	25%	117	10	-	-	-	-	-	127	2
50	50%	91	77	-	-	-	-	-	168	2
	75%	141	100	115	-	-	-	-	356	3
	100%	253	53	-	-	-	-	-	306	2
	TCP	948	-	-	-	-	-	-	948	1
	0%	26	100	-	-	-	-	-	126	2
	25%	100	8	-	-	-	-	-	108	2
100	50%	156	7	-	-	-	-	-	163	2
	75%	464	85	-	-	-	-	-	549	2
	100%	505	467	-	-	-	-	-	972	2
	TCP	1578	910	520	-	-	-	-	3008	3
	0%	37	71	-	-	-	-	-	108	2
	25%	71	93	-	-	-	-	-	164	2
150	50%	104	10	289	-	-	-	-	403	3
	75%	789	85	130	-	-	-	-	1004	3
	100%	901	158	557	-	-	-	-	1616	3
	TCP	2112	1357	1129	1345	1507	1810	150	9410	7
	0%	52	79	-	-	-	-	-	131	2
	25%	9	116	-	-	-	-	-	125	2
200	50%	68	43	195	-	-	-	-	306	3
	75%	585	131	-	-	-	-	-	716	2
	100%	1428	1335	911	-	-	-	-	3674	3
	TCP	2395	3012	4619	734	-	-	-	10760	4
	0%	60	171	-	-	-	-	-	231	2
	25%	27	54	-	-	-	-	-	81	2
250	50%	99	16	650	-	-	-	-	765	3
	75%	805	137	89	-	-	-	-	1031	3
	100%	1183	439	1623	-	-	-	-	3245	3
	TCP	3014	5023	4219	-	-	-	-	12256	3

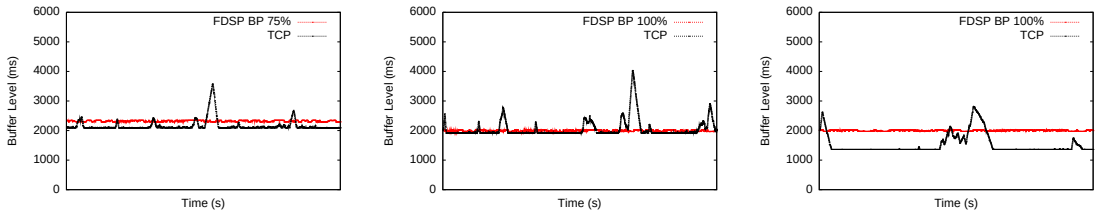
Table 6.4: Rebuffering time and instances for *Nature*. The percentage entries under Protocol represent BP values for FDSP.



(a) Bunny2 – 1 Mbps, 1.25 Mbps bw (b) Bunny2 – 2 Mbps, 2.5 Mbps bw (c) Bunny2 – 4 Mbps, 5 Mbps bw



(d) Bears – 1 Mbps, 1.25 Mbps bw (e) Bears – 2 Mbps, 2.5 Mbps bw (f) Bears – 4 Mbps, 5 Mbps bw



(g) Hobbit – 1 Mbps, 1.25 Mbps bw (h) Hobbit – 2 Mbps, 2.5 Mbps bw (i) Hobbit – 4 Mbps, 5 Mbps bw

Figure 6.10: Client buffer occupancy for FDSP- and TCP-based streaming at 100% BP for 2 and 4 Mbps videos and 75% BP for 1 Mbps.

25%, 50%, and 75% are very similar with minor proportional differences. FDSP-streaming required much less bandwidth than TCP-streaming in congested networks. For instance, the average throughput for *Hobbit* encoded at 4 Mbps was 2.87 Mbps at 0% BP and 2.91 Mbps at 100% BP. The average throughput was lower than the average encoded bitrate for two reasons: (1) packet loss reduced the overall amount of data; and (2) rebuffering lengthened the playback time and, therefore, increased the time value used in the average throughput calculations.

In contrast, TCP-based streaming utilized practically all of the available bandwidth. For example, the TCP throughput for 4 Mbps *Bunny2* streamed on 5 Mbps of available

bandwidth varied between ∼4.9 and 5 Mbps. This is due to a couple of reasons. First, even when there is no congestion, TCP streaming requires bandwidth that is about twice the average video bitrate as discussed in Section B.2. Second, TCP transmission requires significantly more bandwidth than FDSP due to retransmissions. For instance, when streaming the 4 Mbps version of *Bunny2* in 5 Mbps of available bandwidth, 94,477 KB of video data was transmitted and received for FDSP-based streaming compared to 155,441 KB for TCP-based streaming. This shows that FDSP would be more suitable for adaptive streaming in congested networks than TCP due to its lower bandwidth requirements. This would result in less bitrate switching, especially in congested networks. Furthermore, less bitrate down-switching would also raise the average bitrate.

FDSP-based streaming also exhibited very similar results for the dynamic congestion scenarios. However, the video completely stalled in the case of TCP-based streaming. This is because pure-TCP throughput is adversely affected by the available bandwidth oscillation due to the inability of TCP slow-start to achieve a sufficiently large congestion window, which results in persistently low throughput and increased retransmissions. In contrast, FDSP's UDP portion utilizes the oscillating bandwidth more effectively. Finally, neither FDSP nor TCP was negatively affected by non-congested scenarios.

6.5.2.2 More Stable Client Buffer Occupancy

FDSP-based streaming showed more stable client buffer levels than TCP-based streaming. Figure 7.2 shows the client buffer occupancy for all the test videos and encoded bitrates streamed via FDSP and TCP at different static congestion levels. In general, FDSP-based streaming shows a consistent client buffer occupancy compared to more erratic results exhibited by TCP. This is the case for 0%∼100% BP for the videos encoded at 2 or 4 Mbps and 0%∼75% BP for the 1 Mbps videos. Since the client buffer behaves similarly for these BP ranges, the results for the highest respective BP values are shown in Figure 7.2.

Among the three encoded bitrates, 1 Mbps videos exhibit the highest proportion of TCP data when streamed via FDSP. This is because most of the frames are so small that they have very little data beyond the slice headers. Consequently, protecting their slice headers via TCP transmits almost all of the corresponding frame data via TCP. As a result, FDSP-based streaming and the corresponding buffer occupancy at higher BP values (beyond 75%) greatly resembles TCP-based streaming for 1 Mbps videos. Therefore, the recommendation

is to use the FDSP client buffer occupancy for buffer-based adaptation at the full BP range only when streaming at higher video bitrates. In this regard, lower video bitrates call for a more limited BP range. A scheme that maps a given video bitrate to the BP range that best leverages the buffer occupancy towards buffer-based adaptation is beyond the scope of this experimental study and reserved for future work.

6.6 Conclusion and Future Work

This paper shows that FDSP is suitable for high-quality, low-latency HD video streaming over the Internet by combining the reliability of TCP with the low-latency of UDP. Our implementation and experiments on a real testbed showed that FDSP results in significantly less rebuffering than TCP-based streaming and much lower PLR than UDP-based streaming.

Our testbed experiments also show that FDSP-based streaming outperforms TCP-based streaming of multi-bitrate videos in terms of lower bandwidth requirements and more stable client buffer occupancy. As a result, FDSP can be used to potentially improve QoE in adaptive streaming by reducing bitrate switches, increasing the average video bitrate and providing a platform for more precise buffer-based adaptation algorithms.

As future work, BP will be dynamically adjusted in a physical testbed based on the results of an ongoing simulation study that looks at the interaction of PLR and rebuffering and its effect on user QoE. Furthermore, FDSP will be integrated into adaptive streaming to provide an orthogonal option for adaptation algorithms via BP adjustment. Key developments would include an optimal BP range based on the network condition and the type of video in order to minimize both PLR and rebuffering while also providing a reliable buffer occupancy for an adaptation algorithm.

qMDP: DASH Adaptation using Queueing Theory within a Markov Decision Process

Kevin Gatimu, Ben Lee

Under consideration

Chapter 7: qMDP: DASH Adaptation using Queueing Theory within a Markov Decision Process

7.1 Introduction

Online video consumption is rising rapidly. The average viewer is projected to consume close to 90 minutes of video a day by 2020 [1]. The Internet landscape is also shifting rapidly to keep up with increasing video demands. For instance, video and content delivery networks will account for 82% and 71% of all Internet traffic by 2021, respectively [2]. In the wake of a video-dominated Internet, HTTP has become the *de facto* mechanism for video delivery due to its ubiquity. This has led to the state-of-the-art in video streaming, i.e., *HTTP Adaptive Streaming* (HAS).

There are several proprietary implementations of HAS, such as Microsoft Smooth Streaming (MSS) [3], Adobe HTTP Dynamic Streaming [4], and Apple's HTTP Live Streaming (HLS) [5]. There is also the open-source standard called Dynamic Adaptive Streaming over HTTP (DASH) [6]. The DASH standard defines a streaming framework, where a client makes a series of HTTP requests to a server for a series of video chunks called *segments*. The segments are classified according to different quality versions called *representations*. The client requests segments from different representations based on its perceived available bandwidth. This process is dynamically repeated between segments in a process called *bitrate adaptation*.

The main goal of a bitrate adaptation is to ensure the highest possible quality of experience (QoE) for the viewer. This includes maximizing the average bitrate while minimizing bitrate switches, rebuffering time and instances, and startup delay [7]. There are three major categories of bitrate adaptation algorithms: server-side, in-network, and client-side [8]. Client-side methods tend to be the most applicable across a wide range of streaming architectures since they are agnostic of both the server and network configurations. Therefore, this paper will focus on client-side bitrate adaptation. There are three main types of client-side bitrate adaptation algorithms: throughput-based, buffer-based, and hybrid. In general, *throughput-based* methods select video bitrates according to estimated bandwidth, while

buffer-based methods do so based on a target client buffer occupancy. *Hybrid* methods are a combination of the two.

Buffer-based adaptation algorithms typically drive the client buffer level towards a predetermined threshold. When the available bandwidth is lower than the requested segment's bitrate, the buffer gets filled at a slower rate than the normal playback speed. As a result, when the buffer level falls below the threshold, the client requests a segment whose bitrate is lower than the available bandwidth so as to refill the buffer to avoid potential buffer starvation and rebuffering. Conversely, when the available bandwidth is higher than the requested segment's bitrate, the buffer level rises beyond the threshold. The client then requests the next segment at an even higher bitrate (if possible) in order to avoid buffer overflow. This reduces the buffer level towards the threshold.

This is important because buffer overflow causes off periods (i.e., pauses) in an individual client's download stream, which negatively impacts its bandwidth utilization. Furthermore, this leads to poor bandwidth estimation when there are competing TCP flows [9]. On the other hand, throughput-based methods perform bandwidth estimation for bitrate adaptation. However, bandwidth estimation just by itself is error-prone due to inaccuracies [9] thus leading to hybrid methods [10, 11].

Most bitrate adaptation algorithms, however, tend to have significant *ad hoc* components [12], often having parameters that, are in many cases, set arbitrarily and require lots of fine-tuning and heuristics, e.g., buffer threshold values [13]. In response to this shortcoming, several adaptation algorithms based on *Reinforcement Learning* (RL) have recently been proposed. RL involves an agent learning from experience the actions that will yield the best rewards in an environment [14]. In the context of a video streaming environment, a DASH adaptive bitrate controller represents the agent, while its actions are optimal bitrate choices that maximize the rewards respresented by Quality of Experience (QoE).

RL in DASH streaming exhibits the Markov property, i.e., future states depend only on the current state. Therefore it can be modeled as a *Markov Decision Process* (MDP). An MDP is a memoryless stochastic process with decisions that yield rewards and transitions between states. Most RL-based adaptation algorithms, however, suffer from a tradeoff between speed and efficiency [15]. An MDP that models an environment with high accuracy tends to be complex due to keeping track of many states, i.e., the variables in the environment. As result, the MDP converges too slowly towards an optimal solution. This paper proposes a *queueing-theory-based Markov Decision Process* (qMDP), which features

an enhanced state characterization based on an $M/D/1/K$ queueing model for the playback buffer. qMDP has two advantages. First, its fast convergence towards an optimal solution relative to pre-existing RL methods, which makes it well-suited for online optimization. Second, by focusing on the buffer, qMDP minimizes latency by limiting the buffer size.

The rest of this paper is organized as follows. Section B.2 discusses the related work, and Section 7.3 describes qMDP. Section 7.5 details the experiments, followed by results in Section B.6. Finally, Section A.6 summarizes the paper and recommends future work.

7.2 Related Work

This section presents some prominent, more traditional bitrate adaptation algorithms followed by recent proposals in the realm of reinforcement learning.

Huang *et al.* demonstrated the ineffectiveness of using bandwidth estimation alone by analyzing three major HAS services, i.e., Netflix, YouTube, and Vudu [9]. They show that the negative interaction between competing TCP flows and a video flow with an on-off cycle exhibited by periodic chunk scheduling leads to severe bandwidth underestimation. For example, when a client buffer becomes full, downloading pauses, which results in an off period. Meanwhile, competing TCP flows greedily use up the available bandwidth. When the client buffer has more room, the video flow resumes, but its TCP congestion window is stunted due to congestion brought about by greedy competing TCP flows. As a result, the client underestimates the available bandwidth as its buffer becomes depleted faster, and then requests a lower video bitrate in order to restore the buffer to a healthy level. This process repeats during the next off period of the video flow resulting in an even lower bitrate selection. This is referred to as the *downward spiral effect*.

Huang *et al.* then used the rate of change of buffer occupancy to infer the difference between the available bandwidth and the requested bitrate [16]. For instance, a rapidly growing buffer indicates a high available bandwidth and a relatively low requested bitrate. They then developed a fixed rate map that linearly maps buffer occupancy to video bitrate based on a simplified model that used segments encoded with constant bitrate (CBR). Segments with variable bitrate (VBR) were normalized relative to the average segment size in order to use the rate map. This approach required bandwidth estimation at startup in order to determine a suitable initial buffer level that corresponds to the available bandwidth [10]. Moreover, in order to account for VBR, a large buffer is required on the order of minutes,

thus making it unsuitable for shorter videos.

Our proposed qMDP method also analyzes the buffer occupancy as an indicator of the available bandwidth in comparison to the current requested bitrate. However, rather than using a fixed rate map, our method incorporates bandwidth estimation beyond the startup phase to create a more dynamic relationship between the buffer occupancy and the requested bitrate via an MDP. In addition, rather than initially assuming CBR, qMDP works with both CBR and VBR videos.

Jiang *et al.* proposed the *Fair, Efficient, and Stable adapTIVE* (FESTIVE) algorithm to address the fairness issue related to the downward spiral effect [17]. They performed randomized chunk scheduling, stateful bitrate selection, delayed update, and harmonic mean bandwidth estimation. FESTIVE resolves unfairness caused by pre-determined periodic chunk scheduling by randomizing chunk scheduling. It also chooses a random buffer occupancy from a uniform distribution around a target buffer level. This creates a variation in chunk requests. In addition, FESTIVE's stateful bitrate adaptation accounts for the current bitrate trend (going up or down) in addition to just looking at the current available bandwidth. Otherwise, the downward spiral effect would afflict players resuming chunk requests. qMDP features a deterministic target buffer rather than a randomized one. This provides the option of future work involving a platform for coordinating multiple clients deterministically.

Li *et al.* also addressed the fairness issue via *Probe AND Adapt* (PANDA) by computing segment inter-request times [18]. The network is probed during off periods by increasing the sending rate until the available bandwidth is fully utilized. The resultant bandwidth estimation is then used to perform chunk scheduling. As previously noted, however, throughput spikes can lead to the downward spiral effect. In that regard, qMDP estimates the available bandwidth without exploiting network resources.

Cicco *et al.* proposed *fEedback Linearization Adaptive StreamIng Controller* (ELASTIC). This method uses client-side feedback control theory to address the on-off traffic pattern that leads to unfairness and bandwidth underutilization when multiple video flows share a bottleneck. ELASTIC uses a single controller to throttle the video level in order to drive the buffer to a set-point. As a result, it achieves better bandwidth utilization than PANDA in spite of having more bitrate switches. At the same time, ELASTIC achieves fair share regardless of the number of video flows compared to PANDA and FESTIVE, which are more prone to the downward spiral effect. qMDP optimizes both bitrate selection and buffer level

instead of controlling the bitrate in order to achieve a desired buffer level. This leads to better bandwidth utilization and less bitrate switches.

Yin *et al.* proposed a *Model Predictive Control* (MPC) algorithm that maximized QoE via optimal bitrate selection based on throughput prediction. This method requires extensive offline optimization over an exhaustive set of scenarios. In addition, the optimization was done over a fixed horizon, i.e., with respect to a limited number of future video segments. A compressed table was then used online instead of repeating the optimization calculations for each bitrate selection decision. The authors proposed using MDPs for analyzing throughput and buffer state transitions. The proposed qMDP addresses the limitations of MPC by not only using an MDP-based optimization, but also a limited set of states, making it suitable for online optimization.

Yadav *et al.* proposed *QUEuing Theory-based Rate Adaptation* (QUETRA), which analyzes the effect of buffer occupancy on rate adaptation [13]. It uses an $M/D/1/K$ queueing model to study the relationship between buffer occupancy, buffer capacity, network throughput, and the selected video bitrate. In particular, it calculates the expected buffer occupancy given a bitrate, throughput, and buffer capacity. The segment arrival is modeled as a Poisson distribution (M) and serviced at a deterministic (D) rate of one segment at a time by a single (1) server with a finite (K) buffer capacity. QUETRA's queueing model partially addresses the *ad hoc* nature of many adaptation algorithms. However, it arbitrarily sets the buffer slack or availability (i.e., the empty buffer space) equal to the current buffer occupancy. This is done by using a slack-bitrate mapping derived from an $M/D/1/K$ queue occupancy formula to pick a bitrate that will drive the buffer towards the desired slack. As a result, high buffer occupancy leads to a bitrate selection that aims to increase buffer slack, which subsequently results in decreased occupancy. Conversely, low buffer occupancy leads to a bitrate choice that aims towards low slack and a correspondingly high buffer occupancy. Ultimately, the buffer occupancy converges to $K/2$. This prevents buffer underflow (which results in buffering), or overflow (which causes off periods in traffic). QUETRA performs throughput estimation and rate adaptation every 5 segments rather than every segment in order to minimize video bitrate switches leading to better QoE. Throughput smoothing techniques can either overestimate (e.g., EMA, average of last 3) or underestimate (e.g., gradient-based EMA, low pass EMA, and KAMA). The former methods are preferred since they follow the throughput more closely and call for a higher bitrate selection. They are also simpler as they operate on a smaller time scale. The latter methods are more conservative

(and more complex) due to a larger time scale and result in initially low bitrates compared to the available bandwidth. The buffer then fills up faster because low bitrate segments are smaller in size and can be downloaded relatively quickly. This then leads to requests for higher bitrates, which results in more bitrate changes and thus lower QoE. qMDP also features an $M/D/1/K$ queue to model the buffer dynamics. However, rather than setting the buffer slack arbitrarily equal to the buffer occupancy, the deviation between the actual and model-based buffer levels are factored into the MDP's decision-making process.

More recently, dynamic learning algorithms have been proposed to counter the pre-exisiting methods, which are often hard-coded and based on heuristics. Jarnikov *et al.* proposed a client-based MDP controller that lets users trade off between average bitrate and bitrate switches. The MDP uses two states: previous quality level and relative progress, i.e., the time difference between download completion and playout deadline. The reward function penalizes deadline misses and quality level changes. This method provides high average quality, but at the expense of high bitrate switches. Furthermore, the MDP model depends on approximated network behavior and average segment sizes. Drastic changes in either would require a new model. The relative progress metric considers buffer occupancy implicitly, but qMDP does so explicitly using a queueing model. In addition, the control strategy used in qMDP has more flexibility without being limited by specific network and video characteristics.

Claeys *et al.* proposed a client-based Q-Learning adaptation algorithm [19]. The main contribution of this method is a weighted sum of reward functions that is used to measure QoE. The first reward function is based on segment quality levels and the total number of quality levels, while the second is based on bitrate oscillations that are described by duration and amplitude. The third reward function heavily penalizes buffer occupancy that is at or below 10% while rewarding or penalizing a growing or decaying buffer, respectively. This Q-Learning-based method, however, had a large state space and converged too slowly. *Frequency-Adjusted Q-Learning* offered improvements by increasing sensitivity to changes in video and network characteristics [20]. This was done by reducing the state space to just buffer occupancy and estimated bandwidth, which significantly decreased convergence time. However, this simplified model was detrimental to overall QoE. qMDP addresses this tradeoff by offering fast convergence without adversely affecting QoE. In addition, while the Q-Learning-based methods proposed by Claeys *et al.* are useful for adaption, their benefits over other state-of-the-art methods are not clear as they were only shown to outperform

MSS by about 10% based on a QoE metric. qMDP shows improvements over state-of-the-art methods that already outperform proprietary solutions such as MSS.

Chiariotti *et al.* proposed an online MDP-based algorithm that addresses the tradeoff between convergence speed and model complexity [15]. Rather than solving an exact optimization problem that would require prior knowledge of the network conditions, this method uses soft constraints that penalize rebuffering as well as deviation from a safe buffer level. Convergence is sped up using compact state definitions, and intermediate deterministic (non-stochastic) states that are updated in parallel. However, this method is simulation-based and lacks any of the dynamics of TCP network transmission. On the other hand, qMDP will be deployed on a physical testbed, while also utilizing a limited state space.

Zhou *et al.* proposed an MDP-based DASH adaptation algorithm called *mDASH* [21]. This method features a state vector containing current buffer occupancy, buffer changing rate, video bitrates for previous segments, a bitrate consistency function, and bandwidth conditions. This extensive state space necessitates a sub-optimal version of the original method for efficiency. On the other hand, qMDP uses a limited state space and an $M/D/1/K$ queue to model buffer occupancy instead of using additional states in the MDP.

Mao *et al.* proposed *Pensieve* to address the use of fixed control rules based on inaccurate models in state-of-the-art adaptation algorithms [22]. Pensieve uses RL to automatically generate adaptation decisions based on a neural network whose inputs are client observations, e.g., buffer occupancy, last selected bitrate, previous throughput measurements, etc. Pensieve is server-based and is based on a simulator that simplifies network dynamics by assuming 100% bandwidth utility, which is achieved by disabling TCP slow-start on the server. On the other hand, qMPD is client-based, thus requiring no modification to video servers while also providing more immediate feedback for bitrate adaptation.

7.3 The Proposed Method

The proposed qMDP method models the DASH client as an MDP and the client buffer as an $M/D/1/K$ queue [23]. The rest of this section includes a brief overview of an MDP, followed by a contextualized description of the DASH client model. The client buffer is then presented as an $M/D/1/K$ queue as a part of the MDP model.

7.3.1 Markov Decision Process

An MDP is made up of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a state transition probability matrix $\mathcal{P}$, and a reward function $\mathcal{R}$. The probability of transitioning from state s to s' via an action a at time t is given by

$$\mathcal{P}^a_{ss'} = Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\},$$

where $\{s_t, s_{t+1}\} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. The value of a state is a measure of how good it is to be in that state and is defined by the expected future reward, which, in turn, consists of the immediate reward $\mathcal{R}^a_{ss'}$ and the discounted values of future states $V(s')$. Therefore,

$$\begin{aligned} V(s) &= \sum_{s'} \mathcal{P}^a_{ss'} \left\{\mathcal{R}^a_{ss'} + \gamma V(s')\right\} \\ &= E\left\{\mathcal{R}^a_{ss'} + \gamma V(s')\right\} \\ &= E\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}\right\} \\ &= E\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right\} \end{aligned} \tag{7.1}$$

where $\gamma \in [0, 1]$ is a discounting factor that is applied to future state rewards in order to account for prediction uncertainty. Furthermore, a policy π maps each state to an action with probability $\pi(s, a)$. Therefore,

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{V(s)\right\} \\ &= \sum_a \pi(s, a) \left(\sum_{s'} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V\pi(s')\right)\right), \end{aligned} \tag{7.2}$$

where E_π is the expected value under the probability distribution given by policy π. Therefore, $V^\pi(s)$ is the probability-weighted average of all possible actions while in state s. The value of just a single action, $Q^\pi(s, a)$, is V^πwithout the expectation E_π. Therefore,

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right). \tag{7.3}$$

Combining Equations (7.2) and (7.3) gives

$$V^{\pi}(s) = \sum_{a} \pi(s,a) Q^{\pi}(s,a). \tag{7.4}$$

Since $Q^{\pi}(s,a)$ accounts for just a single action, Equation (7.3) can be reformulated by focusing the probabilty-weighted average on just the values of future states to give

$$Q^{\pi}(s,a) = \mathcal{R}^{a}_{ss'} + \gamma \sum_{s'} \mathcal{P}^{a}_{ss'} V^{\pi}(s'). \tag{7.5}$$

Combining Equations (7.4) and (7.5) yields a recursive formula that forms the basis for solving MDPs, i.e.,

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left(\mathcal{R}^{a}_{ss'} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} V^{\pi}(s') \right) \tag{7.6}$$

$$Q^{\pi}(s,a) = \mathcal{R}^{a}_{ss'} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \sum_{a' \in \mathcal{A}} \pi(s',a') Q^{\pi}(s',a'). \tag{7.7}$$

Solving an MDP means finding an optimal policy that maximizes V^{π} and Q^{π}. Therefore, the optimal state value function $V^{*}(s)$ is defined as:

$$\begin{aligned} V^{*}(s) &= \max_{\pi} V^{\pi}(s) \\ &= \max_{a} \mathcal{R}^{a}_{ss'} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} V^{\pi}(s'). \end{aligned} \tag{7.8}$$

Similarly, the optimal action value function $Q^{*}(s,a)$ is defined as:

$$\begin{aligned} Q^{*}(s,a) &= \max_{\pi} Q^{\pi}(s,a) \\ &= \mathcal{R}^{a}_{ss'} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \max_{a'} Q^{\pi}(s',a'). \end{aligned} \tag{7.9}$$

V^π and Q^π can be estimated using Monte Carlo simulation by averaging many random samples of state and action values. However, for large state spaces, function approximators can be used to represent the states using a finite state of parameters, which are then adjusted according to observed returns. The latter is the basis for solving qMDP as discussed in Section 7.3.5.

7.3.2 Modeling a DASH Client as an MDP

Consider a DASH client that has just downloaded segment i of playback length L seconds and encoded with bitrate q. The resulting buffer level, B_{i+1}, measured in seconds of video data, is given by:

$$B_{i+1} = B_i + L - \frac{S(q_{i+1})}{w_i}, \tag{7.10}$$

where B_i is the previous buffer level, w_i is the measured bandwidth during downloading, q_i is the encoded bitrate of the segment, and $S(\cdot)$ is a byte size indicator function. Therefore, the new buffer level is determined by adding the playback length of the downloaded segment and subtracting the time it takes to download the segment.

Let $a_i(q_i)$ be the action associated with choosing segment i from the representation whose encoded bitrate is q. Then, Equation (7.10) can be rewritten as

$$B_{i+1} = B_i + L - \frac{S(a_i(q_i))}{w_i}, \tag{7.11}$$

and the segment bitrate transition q_{i+1} can be expressed as

$$q_{i+1} = a_i(q_i). \tag{7.12}$$

Let the state of the DASH client be $s_i = \{B_i, B_i^*, w_i\}$, where B_i^* is the desired buffer occupancy for the upcoming state. Based on QoE requirements, 3 out of 4 reward functions are defined. The higher the average bitrate, the higher the average quality of the video, which leads to higher QoE. Therefore, the reward function of choosing bitrate q for segment i, r_i^q, is an average of the bitrates of the previous j segments defined as

$$r_i^q = \frac{1}{j} \sum_{k=i-j+1}^{i} q_k. \tag{7.13}$$

The average is limited to j past segments to avoid averaging bitrates over the whole video duration. Since a bitrate switch is detrimental to QoE, it constitutes a negative reward, which is defined as the difference between the last bitrate and the upcoming bitrate selection r_i^s given as

$$r_i^s = |a_i(q_i) - q_i|. \tag{7.14}$$

Finally, r_i^r, is a negative reward based on rebuffering time, which is calculated as the excess of download time over buffer capacity.

7.3.3 Modeling the Optimal Buffer Occupancy as an $M/D/1/K$ Queue

The desired buffer occupancy B_i^* is modeled as the average occupancy of an $M/D/1/K$ queue with finite capacity (K), Poisson (M) arrivals and a deterministic (D) service time of one (1) segment at a time. Poisson arrivals are characterized by an exponential probability distribution with arrival rate λ, which is also the parameter for an exponential distribution, $X \sim \exp(\lambda)$, given by

Furthermore, the Poisson process is characterized by independent and exponentially distributed arrival times. Therefore, the expected value of the Poisson process, $E[X]$, is given by

$$\begin{aligned} E[X] &= \int_{-\infty}^{\infty} f_X(t)dt,\ t \geq 0 \\ &= \int_{0}^{\infty} \lambda \mathrm{e}^{-\lambda t} dt \\ &= \frac{1}{\lambda} \end{aligned} \tag{7.15}$$

The segment arrival rate is given as the ratio of the network throughput, w, and the segment size. At the same time, the segment size is a product of the segment's encoded bitrate q and playback length L, i.e., $\lambda = w/(qL)$. For example, a 2-second segment of 4

Mbps quality downloaded at 5 Mbps of network throughput yields the following arrival rate:

$$\begin{aligned}\lambda &= \frac{w}{qL} \\ &= \frac{5\text{Mbps}}{(4\text{Mbps})(2\text{s})} \\ &= 0.625/\text{s}\end{aligned}$$

Assuming normal playback speed, the service rate μ is 1 segment per segment duration, i.e., $\mu = 1/L$. Therefore, the utility of the queue ρ is given by

$$\begin{aligned}\rho &= \frac{\lambda}{\mu} = \frac{\frac{w}{qL}}{\frac{1}{L}} \\ &= \frac{w}{q}.\end{aligned} \tag{7.16}$$

Brun *et al.* derived the steady state distribution equations for an $M/D/1/K$ queue [24] (see Appendix for the derivation). Let X_K be the average number of customers in an $M/D/1/K$ queue. Therefore,

$$X_K = K - \frac{\sum_{i=0}^{K-1} z_i}{1 + \rho z_{K-1}} \tag{7.17}$$

where

$$z_i = \sum_{j=0}^{i} \frac{(-1)^j}{j!}(i-j)^j e^{(i-j)\rho}\rho^j$$

The queue capacity X_K depends on the utility ρ, which in turn depends on the ratio of the available bandwidth w and the encoded bitrate of the requested segment q. Therefore, choosing q determines the buffer occupancy. However, since there is no prior knowledge of w, a bandwidth estimation algorithm is required. This paper follows the recommendations laid out by Yadav *et al.* [13] and uses an exponential moving average with the last three throughput measurements. This method leads to the highest average bitrate with relatively low bitrate switches for their method, which also models the client buffer as an $M/D/1/K$

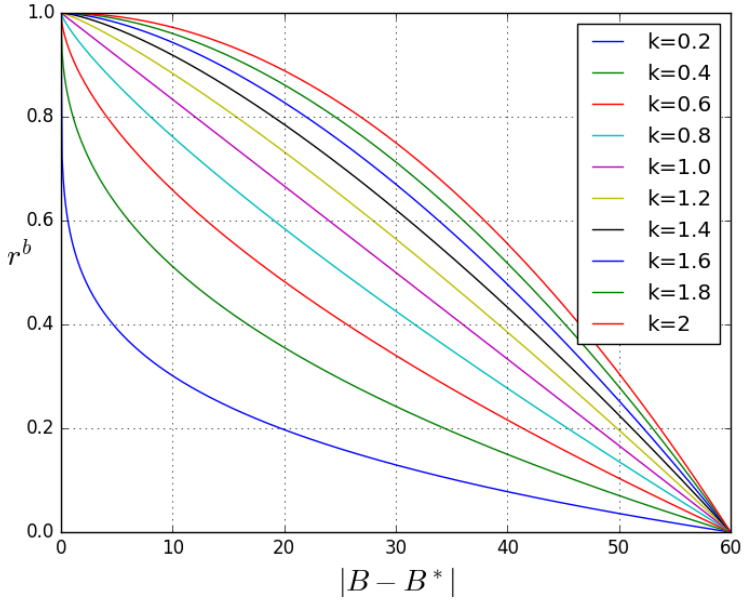


Figure 7.1: A graph of buffer reward (r^b) vs. the difference between the current buffer (B) and buffer target (B^*) for a 60-second buffer.

queue.

Our queueing model provides an optimal buffer occupancy, B_i^*, according to the estimated bandwidth. This is used in calculating the final reward function, r_i^b, defined by how closely the current buffer occupancy, B_i, matches the optimal buffer level B_i^*, i.e.,

$$r_i^b = 1 - \left(\frac{| B_i - B_i^* |}{K} \right)^k, \tag{7.18}$$

where $k \in (0, 1)$ and is proportional to the rate of reward increase as the buffer approaches its target. For instance, Figure 7.1, shows r^b for different values of k. The graph gets steeper as $|B_i - B_i^*|$ approaches 0, which corresponds to the maximum reward of 1. This incentivizes the agent to seek higher rewards by choosing actions that minimize $| B_i - B_i^* |$.

The total reward r_i is a weighted sum of the individual reward functions expressed as

$$r_i = c_q r_i^q - c_s r_i^s - c_f r_i^f + c_b r_i^b \tag{7.19}$$

7.3.4 The qMDP State Space

Based on Sections 7.3.2 and 7.3.3, let the state of the DASH client be $s = \{B, B^*, w\}$. The average queue length, X_K, given in Equation (7.17), can be used to obtain the desired buffer occupancy, B^*. The buffer generally needs to be filled up when the occupancy is low and drained when the occupancy is high. Such control is usually done from the perspective of one or more typically fixed thresholds that distinguish between starvation and overflow as discussed in Section B.2. However, the optimal buffer occupancy in the proposed qMDP is based on observed experience instead.

The buffer slack will be denoted as $\hat{B}^*$ such that $B^* + \hat{B}^* = K$. Recall that X_K is a function of ρ, which, in turn, depends on bandwidth and bitrate as shown in Equation (7.16). B^* can be set in two ways. On one hand, choosing a particular bitrate explicitly sets the desired buffer occupancy. This strategy will be referred to as ψ and the corresponding buffer occupancy B^*_ψ. On the other hand, B^* can also be determined implicitly using strategy ω to set the desired slack, $\hat{B}^*_\omega$. This strategy includes a simple heuristic from QUETRA, which outperforms many state-of-the-art methods [13], but with an important addition. In order to sufficiently explore our RL environment, the slack is sampled from a normal distribution, rather than simply setting the slack equal to the current buffer occupancy. The QUETRA slack target (current buffer occupancy) is used as the mean parameter, μ_ω, in a normal distribution, $f(\hat{B}^*_\omega \mid \mu_\omega, \sigma^2_\omega)$, where σ^2_ω is the variance. In addition, σ^2_ω is proportional to the distance, $D(X, \mu_\omega)$, between μ_ω and the buffer terminal (i.e., zero or maximum capacity), depending on which one is closer. As a result, the further μ_ω is from either buffer terminal, the larger the sample space for $\hat{B}^*_\omega$. $D(X, \mu_\omega)$ is defined as

$$D(X, \mu_\omega) = \begin{cases} \mid \mu_\omega - 0 \mid & \mu_\omega < \frac{X_K}{2} \\ \mid \mu_\omega - X_K \mid & \mu_\omega > \frac{X_K}{2}. \end{cases}$$

B^* is then picked using the strategy ψ with probability P_B and the strategy ω with probability $1 - P_B$. The state space is then upgraded to include the parameter P_B, which ranges from 0.1 to 0.9 at intervals of 0.1. With the definition of B^*, the state space is now completely defined as $s = \{B, B^*, w, P_B\}$. Figure 7.2 shows the difference between $\hat{B}^*_\omega$ and the QUETRA slack target.

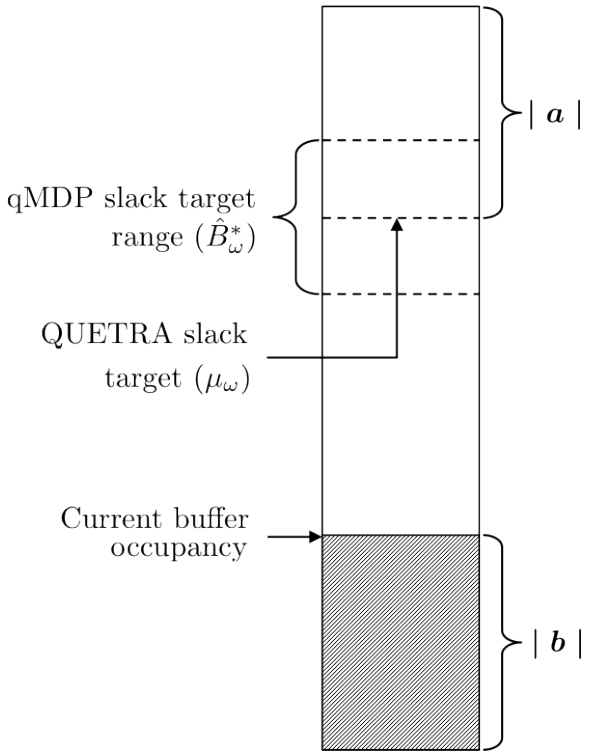


Figure 7.2: A client buffer showing the difference between $\hat{B}^*_\omega$ and the QUETRA slack target. Note that $\mid a \mid = \mid b \mid$.

7.3.5 Solving qMDP

There are several approaches to solving qMDP. Since the probability transitions and the rewards are unknown, a model-free solution is required. RL can be used to obtain the optimal action-value function, $Q^*(s, a)$. In particular, an agent learns the optimal policy through interactions with an environment. It observes a state then takes a corresponding action. It then receives an immediate reward and transitions to a new state. This is illustrated in Figure 7.3 in the context of qMDP, where the adaptive bitrate controller is the agent, the actions are bitrate decisions, rewards are based on QoE metrics, and the state is a set of observed metrics as described in Section 7.3.4.

RL can be used to obtain the optimal action-value function, $Q^*(s, a)$ by iteratively updating the the action-value function. A greedy policy achieves this by always selecting an action, a^*, that maximizes Q, i.e,

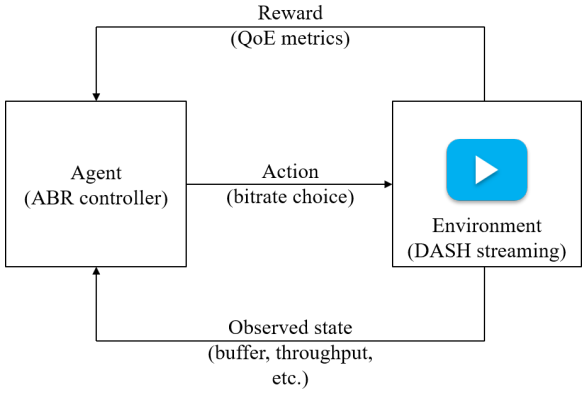


Figure 7.3: Reinforcement learning in the context of qMDP.

$$a^* = \underset{a}{\operatorname{argmax}}\, Q(s, a).$$

This is referred to as *Q-learning* [25]. However, a purely greedy policy ignores states that result from actions with lower Q-values, thus limiting potential policy improvement. In order to explore these additional states, a random action is chosen with a small probability, ϵ, while the greedy action is picked with probabilty $1 - \epsilon$. This is referred to as an ϵ-greedy policy and it converges to the optimal policy better than a purely greedy one [26].

A basic approach is to build an initial Q-table with arbitrarily set Q-values (i.e., state-action pairs) for the function $Q(s, a)$. An action is selected using the ϵ-greedy policy, the reward is measured, and a Q-update is performed as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)], \tag{7.20}$$

where $\alpha \in [0, 1]$ is the learning rate. This is repeated until an optimal Q-table, $Q^*(s, a)$, is obtained.

In practice, however, this approach is too inefficient for our application, which has a continuous state space, i.e., an effectively infinite number of states, each comprising a network bandwidth estimation and a buffer occupancy measurement. Even if the available bandwidth range was just 1,000 bps and the buffer capacity was 10 seconds, this would result in 10,000 states. In reality, available bandwidth can range from hundreds of Kbps to tens of Mpbs, while the buffer capacity can be on the order of minutes. Furthermore, the number

of state-action pairs is vast and is equal to the number of states multiplied by the number of actions (i.e., the available video bitrate choices).

Besides the inefficiency, this simple approach is unstable as small Q-updates lead to drastic changes in the policy. At the same time, $Q^*(s,a)$ tends to be biased towards a specific sequence of states and actions and would need to be repeated for each different sequence [27]. The large state space can be fit into a compact representation using *function approximation*, while *experience replay* can be used to reduce instabilility by removing correlations in the state-action sequence. Furthermore, this produces a generalized version of $Q^*(s,a)$ that can be applied to different state-action sequences.

A typical function approximator is a deep neural network (DNN), which results in $Q(s,a;\theta) \approx Q^*(s,a)$, where θ is a limited set of parameters used to represent a large state space. A DNN is a multi-layer representation of data that becomes increasingly abstracted with the number of layers. In the context of Q-learning, it is referred to as a deep Q-network (DQN) [27]. Experience replay adds stability to the DQN by storing experiences such that $D_t = \{e_1, \ldots, e_t\}$, where $e_t = (s_t, a_t, r_t, s_{t+1})$ is an experience at at time t. During learining, Q-updates are done only on samples of experience drawn uniformly at random from D rather than the complete sequence.

A DQN can be trained towards Q^* by optimizing a loss function across a series of iterations. Each iteration consists of reducing the mean square error between an approximate current action-value function, $Q^*(s,a;\theta_i)$, and an approximate target value $r+\gamma \max_{a'} Q(s',a';\theta_i^-)$, where θ_i is the set of parameters to be adjusted at iteration i, while θ_i^- is a set of parameters from some previous iteration. This results in the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right], \tag{7.21}$$

Since the loss function is quadratic in nature, optimization is reduced to finding a global minimum. This can be done using *gradient descent*, which involves tracing loss function in the direction of the steepest decline by taking its derivative with respect to the weights, θ, for each iteration i as follows:

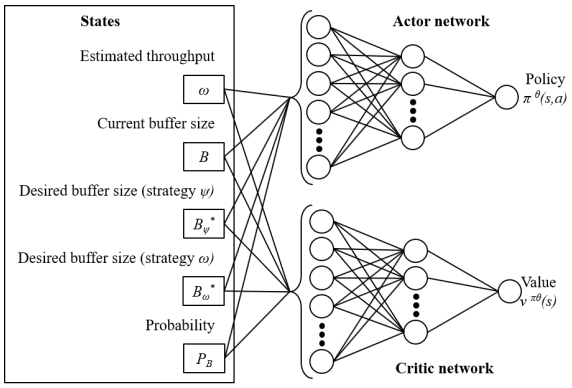


Figure 7.4: Asynchronous advantage actor-critic training model.

$$\nabla_{\theta_i} L(\theta_i) = \begin{bmatrix} \frac{\partial L(\theta_i)}{\partial \theta_{i,1}} \\ \frac{\partial L(\theta_i)}{\partial \theta_{i,2}} \\ \vdots \\ \frac{\partial L(\theta_i)}{\partial \theta_{i,n}} \end{bmatrix}.$$

The resulting gradient descent is:

$$\mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]. \tag{7.22}$$

Once the L is minimized, the resulting weights consitute the parameters for a model of Q, given by $Q(s; a; \theta)$.

7.4 Training Algorithm

This section describes the training algorithm for qMDP. It employs a state-of-the-art method called asynchronous advantage actor-critc (A3C) [28]. It includes two neural networks, i.e., an *actor network* and a *critic network* as shown in Figure 7.4. Both networks take the state variables as input. The output of the actor network is the optimal policy, $\pi_\theta(s, a)$, i.e., the probability distribution over possible actions given a state input.

Meanwhile, the critic network evaluates the performance of the actor via an advantage

function,

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s). \tag{7.23}$$

Based on Equation 7.2, $V^{\pi_\theta}(s)$ is the expected total reward from state s under policy π_θ. Therefore, $A^{\pi_\theta}(s,a)$ shows how good the value of an action a is compared to the average action taken. Temporal Difference (TD) learning [14] is used to learn an estimate of V^{π_θ} from the critic network by training its own set of parameters. The actor network parameters are then updated via policy gradient. Let a policy objective function, $J(\theta)$, be the expected advantage under policy π_θ. If π_θ is differentiable then the gradient of $J(\theta)$ maximizes the advantage, i.e.,

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}(A^{\pi_\theta}) \\ \nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right]. \end{aligned} \tag{7.24}$$

Similar to the vanilla Q-update in Equation 7.25, the actor network parameters, θ, are then updated as follows:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log \pi_\theta(s_t,a_t) A(s_t,a_t), \tag{7.25}$$

where $\alpha \in [0,1]$ is the learning rate. Once θ is optimal, the critic network's job is done and the actor network can be used to make bitrate decisions in a streaming environment.

7.5 Experiment Setup

A Python-based simulator was used to implement and test qMDP. Simulation offered quick prototyping by being able to play several hundered hours of videos in less than an hour. The streaming environment was based on the DASH reference player [29]. When the client requested a video bitrate, its buffer was filled with the playback time of the video chunk it received. At the same time, the buffer was drained by the time it took to download the chunk and the concurrent playback time. When the buffer was empty while waiting for a chunk to be downloaded this was registered as a rebuffering instance. The time it took for

playback to resume was the corresponding rebuffering time.

The network traces used in our experiment consist of 269 4G/LTE traces collected on trips by mass transit, bicycle, and foot [30]. Video encoded in H.264 was served continuously with bitrate choices of $\{300, 750, 1200, 1850, 2850, 4300\}$ Kbps. The NN architecture was implemented using TensorFlow [31].

QoE Metrics

Video quality is highly subjective and there are several ways of measuring QoE. In our experiments, we used a common version proposed by Yin et al. [11], i.e.,

$$\sum_{n=1}^{N} q(R_n) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} \mid q(R_{n+1}) - q(R_n) \mid, \tag{7.26}$$

where N is the total number of video segments, R_n is the bitrate of segment n, $q(R_n)$ is a corresponding subjective quality rating, and T_n is rebuffering time due to downloading segment n. The first term rewards higher bitrates, while the second and third terms penalize rebuffering and bitrate changes respectively. We then add a third

qMDP Settings

In order to speed up convergence, 16 agents are trained in parallel, each with its own actor and critic networks. Each agent experiences video streaming on a random network trace. At each training step, video metrics are used to calculate reward and the actor network generates a probability distribution for the different bitrates. At every 100 training steps, a central agent gathers experience from all the parallel agents and create a mini-batch that is used for performing gradient descent. The central agent then updates the parallel agents with the updated network parameters. A discount factor $\gamma = 0.99$ is used during training. The learning rate, α, is 10^{-4} for the actor network and 10^{-3} for the critic network.

7.6 Results

Our results compare qMPD against a RL base case provided by Pensieve [22]. Figure 7.5 shows the average reward per network trace for qMDP and Pensieve. On average, qMDP's

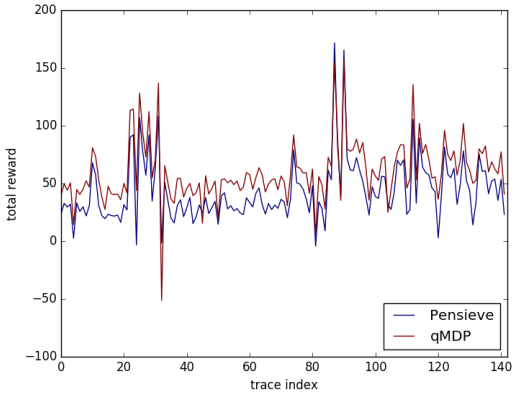


Figure 7.5: Total average reward of qMDP and Pensieve for 142 traces.

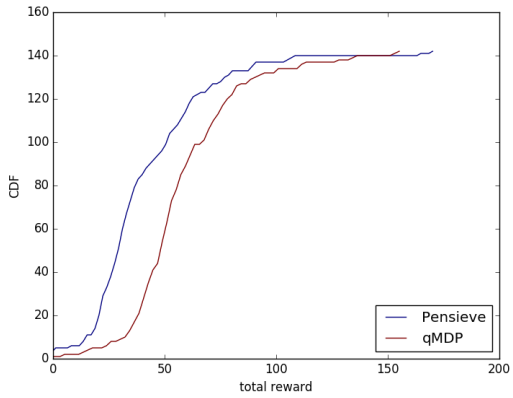


Figure 7.6: Cumulative Distribution Function of total rewards for Pensieve and qMDP.

reward is 16.6 points better than Pensieve, with a maximum of 46.95 points at trace index 24. Figure 7.6 shows a cumulative distribution funciton (CDF) of total rewards for both Pensieve and qMDP. Since we want to maximize reward, the more the graph is to the right, the better. For instance, a CDF value of 50 shows that about 99 videos have a maximum reward of 50, compared to just 49 for qMDP. This means that there is greater proportion of videos whose reward exceeds 50 for qMDP compared to Pensieve. Similarly, 128 videos have a maximum reward of 75 compared to 117 for qMDP.

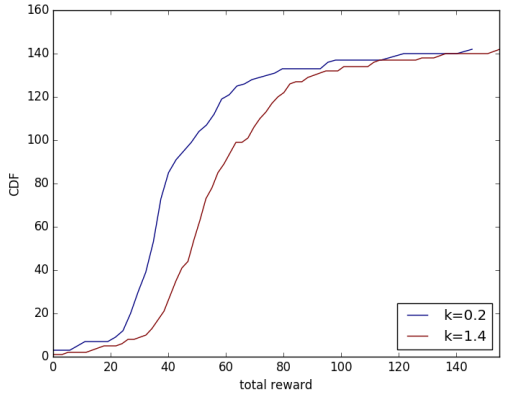


Figure 7.7: Cumulative Distribution Function of total rewards for $k \in \{0.4, 1.4\}$ in r^b.

7.6.1 Buffer Reward Function Parameter

The buffer reward function, r^b includes an exponential parameter, k, that was evaluated to provide the optimal results. We found that, in general, r^b improves as k increases. Each of the values shown in Figure 7.1 was evaluated with performance increasing from $k = 0.4$ and plateauing at $k = 1.4$. Figure 7.7 shows CDF of the total reward between these two values.

7.7 Conclusion

This paper has demonstrated a queueing-theory-based approach to building a state space for reinforcement learning towards improving adaptive streaming. A reward signal based on conventional QoE metrics has been augmented with a reward function based on an ideal buffer target as dictated by the $M/D/1/K$ queue. Our results show that this offers reinforcement learning an alternative approach to constructing the state space while also delivering improved QoE.

APPENDICES

This appendix presents the derivations for the steady-state distribution equations for an $M/D/1/K$ queue for completeness, which is based on the analysis provided by Brun *et al.* [24]. Let q_j be the probability that j customers are left in the queue and α_n be the probability that there are n arrivals during a customer service. Therefore, based on Poisson arrivals,

$$\alpha_n = \frac{\rho^n}{n!}\mathrm{e}^{-\rho}.$$

The probability transition matrix Π is given by [32]:

$$\Pi = \begin{bmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{K-2} & 1-\sum_0^{K-2}\alpha_n \\ \alpha_0 & \alpha_1 & \alpha_2 & \cdots & \alpha_{K-2} & 1-\sum_0^{K-2}\alpha_n \\ 0 & \alpha_0 & \alpha_1 & \cdots & \alpha_{K-3} & 1-\sum_0^{K-3}\alpha_n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_0 & 1-\alpha_0 \end{bmatrix}.$$

Each entry in Π represents a transition probability $\pi_{i,j}$ from state i to j, where the rows are indexed by i and the columns by j. Since Π is ergodic [33], there exists a stationary distribution $Q = [q_0, \cdots, q_{K-1}]$ that is an eigenvector to Π. Therefore, $Q\Pi = Q$, which produces the following system of linear equations:

$$\begin{aligned} \alpha_0 + \alpha q_1 &= q_0 \\ \alpha_1 + \alpha_1 q_1 + \alpha_0 q_2 &= q_1 \\ &\vdots \\ \alpha_{K-2} + \alpha_{K-2} q_1 + \cdots + \alpha_0 q_{K-1} &= q_{K-2}. \end{aligned} \tag{27}$$

Let $q_n = a_n q_0$ for all $n \geq 0$. Therefore, $a_0 = 1$, $a_1 = \mathrm{e}^{\rho} - 1$, and

$$a_k = \mathrm{e}^{\rho}\left(a_{k-1} - \sum_{i=1}^{k-1}\alpha_i a_{k-i} - \alpha_{k-1}a_0\right), \tag{28}$$

where $2 \leq k \leq K-1$. Let $A(z)$ be the z-transform for the sequence a_n, i.e.,

$$A(z) = \sum_{n=0}^{\infty} a_n z^n. \tag{29}$$

Combining Equations (28) and (29), gives

$$\sum_{n=0}^{\infty} a_n z^n = \sum_{n=0}^{\infty} \alpha_n a_0 z^n + \sum_{i=1}^{\infty} \sum_{n=0}^{\infty} \alpha_n a_i z^{n+i-1},$$

which leads to

$$\begin{aligned} A(z) &= a_0 \mathrm{e}^{-\rho} \sum_{n=0}^{\infty} \frac{(\rho z)^n}{n!} + \frac{1}{z} \sum_{i=1}^{\infty} a_i z^i \sum_{n=0}^{\infty} \alpha_n z^n \\ &= a_0 \mathrm{e}^{\rho(z-1)} + \frac{\mathrm{e}^{\rho(z-1)}}{z} (A(z) - a_0) \\ &= \frac{1-z}{1 - z\mathrm{e}^{\rho(1-z)}}. \end{aligned}$$

Furthermore, let $A(z)$ take the form $(1-z)B(z)$, where $B(z) = 1/(1 - z\mathrm{e}^{\rho(1-z)})$. Therefore,

$$\begin{aligned} \sum_{n=0}^{\infty} a_n z^n &= \sum_{n=0}^{\infty} b_n z^n - z \sum_{n=0}^{\infty} b_n z^n \\ &= b_0 + z \sum_{n=1}^{\infty} (b_n - b_{n-1}) z^n), \end{aligned}$$

which implies $a_0 = 1$ and $a_n = b_n - b_{n-1}$ for $n \geq 1$. In order to obtain an explicit expression for b_n, let

$$F(z) = \sum_{m=0}^{\infty} \mathrm{e}^{m\rho(1-z)} z^m,$$

where $m = n - k$. Setting $F(z) = B(z)$ yields

$$b_n = \sum_{k=0}^{n} \frac{(-1)^k}{k!}(n-k)^k \mathrm{e}^{(n-k)\rho}\rho^k$$

—

By normalization, $\sum_{k=0}^{N-1} q_k = 1$. Therefore,

$$q_0 = \frac{1}{\sum_{k=0}^{N-1} a_k} = \frac{1}{b_{N-1}},$$

and

$$q_n = a_n q_0 = \frac{b_n - bn - 1}{b_{N-1}}, \qquad n > 0.$$

The queue length distribution is given by $P = [P_0(N), \cdots, P_N(N)]$, where $P_K(N)$ is the probability that the queue length is K out of a capacity N. Therefore, $P_0(N)$ is the probability that the queue is empty and $P_N(N)$ is the probability that it is full. It thus follows that the probability that the queue is not full is given by $(1-P_N(N)$, while $(1-P_0(N)$ denotes the probability that the queue is not empty. Therefore, the arrival and service rates follow the following conservation law:

$$\begin{aligned}
\lambda(1 - PN(N)) &= \mu(1 - P_0(N)) \\
1 - P_N(N) &= \frac{1}{\rho}(1 - (1 - P_N(N))q_0) \\
1 - P_N(N) &= \frac{b_{N-1}}{1 + \rho b_{N-1}} \\
P_N(N) &= 1 - \frac{b_{N-1}}{1 + \rho b_{N-1}}.
\end{aligned}$$

Let X_N be the average queue length, which can also be viewed as the expected length of the queue. Therefore,

$$
\begin{aligned}
X_N &= E[N] \\
&= \sum_{k=0}^{N} k P_k(N) \\
&= N + \frac{\sum_{k=1}^{N-1} k(b_k - b_{k-1}) - N b_{N-1}}{1 + \rho b_{N-1}} \\
&= N - \frac{\sum_{i=0}^{N-1} b_k}{1 + \rho b_{N-1}}
\end{aligned}
\tag{30}
$$

Appendix A: General Conclusion

The ubiquity of high quality video along with the proliferation of wireless devices has necessitated advancements in both video technology and networking infrastructure. The state-of-the-art in video streaming is HTTP adaptive streaming. Through several manuscripts, this dissertation presented various solutions for adaptive streaming using both dual protocols and reinforcement learning.

Chapter 2 evaluated H.264-encoded video streaming in 802.11 networks, and characterized the problems posed by congestion and hidden nodes. From this study, the H.264 specification demonstrated specific ways of addressing degraded video quality. One of these was through error concealment. Chapters 3 and 4 showed that spatial motion vector recovery using WNVMVA is able to capture properly received motion vectors and conceal lost ones better than most spatial methods. VLC, a popular multimedia framework favored WNVMVA around 50% of the time from a pool of 8 candidate error concealment algorithms.

Chapters 5 and 6 reviewed FDSP and showed that it is suitable for for high quality, low-latency HD video streaming over the Internet by combining the reliability of TCP with the low-latency of UDP. Our implementation and experiments on a real testbed showed that FDSP results in significantly less rebuffering than TCP-based streaming and much lower PLR than UDP-based streaming. Furthermore, FDSP was shown to provide improved QoE relative to HAS through a higher average bitrate and a more stable client buffer. Chapter 7 showed that there is great potential in constructing a reinforcement learning state space beyond observable streaming metrics. By characterizing the client buffer using queueing theory, we are able to not only achieve an enhanced state space, but also improve QoE relative to pre-existing reinforcement learning algorithms.

Appendices A and B analyzed the effect of varying FDSP substream lengths and the extent to which UDP and TCP data streams are overlapped. These studies showed that substream length can be increased while decreasing rebuffering.

The future of video streaming is bright as advances in networking and video content continue to rise simultaneously. As 5G technology marches on towards being mainstream, immersive media such as 360 video and augmented/virtual/mixed reality is becoming in-

creasingly common. As a result, corresponding advances in codecs and content delivery will be required to not only accommodate these new technologies, but also rising video demand.

APPENDICES

Analysis of Optimum Substream Lengths for Dual TCP/UDP Streaming of HD H.264 Video Over Congested WLANs

Arul Dhamodaran, Kevin Gatimu, and Ben Lee

Proceedings of the 14th Annual IEEE Consumer Communications and Networking Conference (CCNC),
Las Vegas, NV, USA, January 8-11, 2017
https://doi.org/10.1109/CCNC.2017.8015366

Appendix A: Analysis of Optimum Substream Lengths for Dual TCP/UDP Streaming of HD H.264 Video Over Congested WLANs

Flexible Dual-TCP/UDP Streaming Protocol (FDSP) is a new method for streaming H.264-encoded High-definition (HD) video over WLANs. FDSP streaming is done in sequential video segments or chunks called *substreams*. Substreams are typically used in HTTP/TCP streaming for efficient switching between different quality levels of video. However, in FDSP, substream lengths are used to control the amount of TCP data that needs to be sent prior to the playback of that substream. Substream length choice also affects the corresponding amount of UDP data. This paper presents an analysis of substream length modification in the context of FDSP. Our results show that as substream length increases, TCP rebuffering time decreases while the number of TCP rebuffering instances increases. However, our results also show that substream length alone does not have a clear correlation with UDP packet loss.

A.1 Introduction

High-definition (HD) video streaming is a critical technology for today's multimedia applications. These video streaming applications can be broadly classified into Video on Demand (VoD) services and direct device-to-device steaming over WLANs. VoD services generally involve streaming servers that deliver HD video content to the end-user via the Internet, e.g., Netflix, Hulu, Amazon Video, etc. On the other hand, direct device-to-device streaming over WLANs is typically seen in home entertainment systems. They facilitate HD video sharing through screen mirroring and other N-Screen applications [1]. However, as more wireless devices become inter-connected, networks will need to support multiple video streams. This will pose significant challenges for providing uninterrupted HD video streaming.

Video streaming is based on either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). TCP facilitates HTTP-based streaming as is the case with VoD services. Meanwhile, direct device-to-device streaming applications that require real-time responsiveness rely on UDP. Using either UDP or TCP has its advantages and disadvantages.

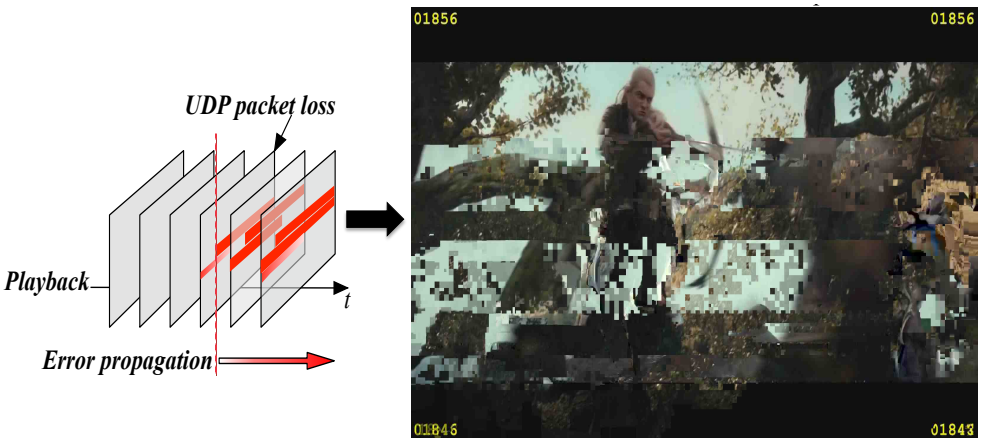


Figure A.1: Frame distortion due to UDP packet loss. Note that packet loss also causes frame distortion in subsequent frames due to error propagation.

UDP is better suited for applications that require low latency, e.g., live video streaming, interactive video, and screen mirroring. However, UDP suffers from *packet loss* due to its unreliable nature, which results in degraded video quality. On the other hand, TCP is a reliable protocol that guarantees packet delivery, which ensures perfect video quality. However, TCP suffers from packet delay that leads to *rebuffering*, and thus stalled video playback.

Figure A.1 illustrates the video quality degradation due to UDP packet loss, which affects not only the frame for which the packet loss occurred but also subsequent frames that use this frame as the reference frame (referred to as *error propagation*). Figure A.2 illustrates the effect of rebuffering caused by TCP packets arriving at the receiver after the playout deadline. This delay causes the received video to stall and wait for more packets to arrive before resuming playback.

Our prior work in [2] proposed a new hybrid H.264 video streaming technique called *Flexible Dual-UDP/TCP Streaming Protocol* (FDSP) that leverages the advantages of both UDP and TCP for HD video streaming between devices over a WLAN. FDSP uses TCP to guarantee delivery of the more important elements of a H.264-coded bitstream, such as Sequence Parameter Set (SPS), Picture Parameter Sets (PPSs), and slice headers, while the rest of the data is transported via UDP. This gives the H.264 decoder a better chance of decoding received video even when packet loss occurs. Overall, FDSP was shown to strike a balance between visual quality and delay by achieving higher video quality than pure-

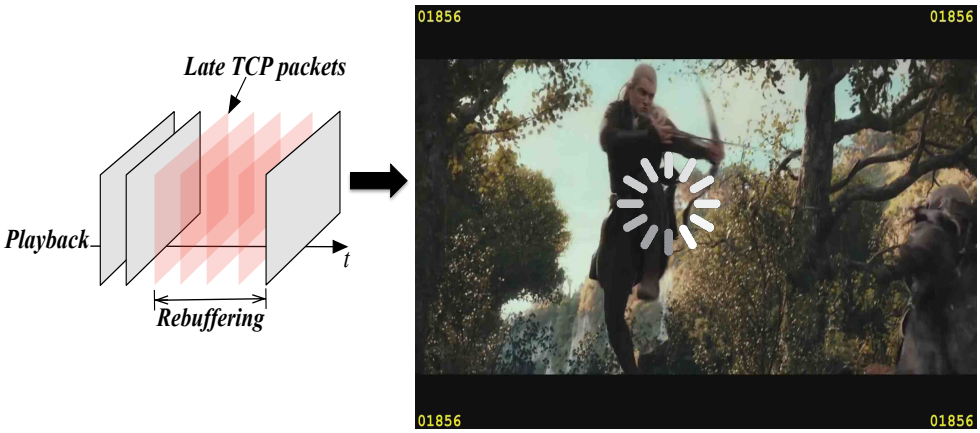


Figure A.2: Rebuffering due to late TCP packets.

UDP and less rebuffering than pure-TCP. FDSP was improved upon in [3] by introducing *Bitstream Prioritization* (BP) to reduce UDP packet loss and error propagation. BP is an adjustable metric that is used to select additional H.264 data to be transported over TCP. Increasing BP decreased UDP packet loss but also increased TCP rebuffering time (i.e., the total amount of time the video spends in the stalled state) and instances (i.e., the number of times the video stalls). FDSP-BP was further improved by using *Adaptive BP* that dynamically adjusts the BP parameter based on estimated TCP rebuffering time and UDP packet loss ratio (PLR) [4]. This further reduced both packet loss and rebuffering.

FDSP-based streaming is done in sequential video chunks called *substreams*. For each substream, the important syntax elements are sent first via TCP and then the rest of the data is sent via UDP. Therefore, the substream length determines the amount of TCP data that needs to be sent prior to the playback of that substream. In addition, the amount of TCP data affects rebuffering time (i.e., delay) between substreams. It also determines TCP transmission time, which in turn affects whether or not UDP packets meet the playout deadline at the receiver. The substream length also affects the total number of substreams, which is directly related to the number of possible rebuffering instances. In our prior work on FDSP, the substream length was chosen to be fixed at 10 seconds [5]. This paper analyzes how varying substream lengths affect UDP PLR and TCP rebuffering time and instances. Based on our results, better insight can be developed on how FDSP-based streaming can be furthered enhanced.

A.2 Related Work

Substream implementations for video streaming applications are commonly found in HTTP Adaptive Streaming (HAS) technology [6]. HAS aims to adjust video to the best possible quality (i.e., bitrate) under fluctuating network conditions. As such, HAS segments a video into multiple substreams, and dynamically switches between different quality levels (i.e., bitrates) as the video transitions from one substream to the next based on network conditions.

Existing proprietary HAS technologies use different substream lengths. For example, Microsoft Smooth Streaming (MSS) specifies 2-second substreams [7], Apple HTTP Live Streaming (HLS) typically targets a maximum of 10 seconds per substream [8], and Adobe HTTP Dynamic Streaming (HDS) substreams vary between 2 and 5 seconds [9]. These substream lengths are relatively short so that the streaming system can react fast enough to changes in network conditions while avoiding extra overhead [10].

In addition to these proprietary solutions, Dynamic Adaptive Streaming over HTTP (DASH) is an open standard for HAS introduced by MPEG in 2012 [11]. DASH provides no specification on substream lengths, and instead, this is left to the implementer. For example, in [12], Scalable Video Coding (SVC) is used to provide different bitrate versions of a video in a single media file rather than using multiple files with varying quality levels. Since all versions are encoded in the same file, there is flexibility in choosing not only the quality level but also the substream length based on media content.

Another HAS improvement based on substream length was proposed in [5]. A substream length of around 10 seconds is found to be sufficient for calculating how quickly a receiver fetches a substream. The network bandwidth can be determined solely from the application layer by observing the difference between fetch time and playout duration for a particular substream, and used by the adaptation algorithm.

The aforementioned substream length implementations are designed for HTTP streaming, which is TCP-based. Furthermore, they are aimed at efficient switching between different quality levels of the same video in order to achieve the best quality of experience (QoE) [6]. As a result, they are orthogonal to FDSP-based streaming since it can be combined with HAS and used within each quality level. However, to the best of our knowledge, there is no prior work on substream lengths for video streaming that is based on both TCP and UDP. Therefore, this paper looks at how different substream lengths affect FDSP

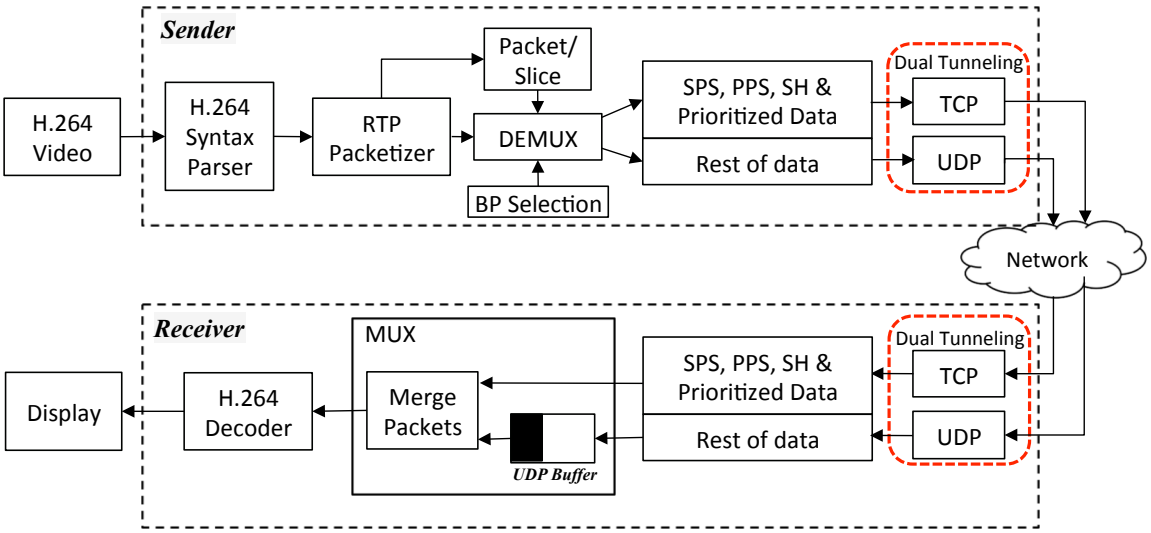


Figure A.3: Flexible Dual-UDP/TCP Streaming Protocol (FDSP) Architecture [2] augmented with modified MUX and DEMUX modules for FDSP-BP.

streaming. In addition, rather than optimizing bitrate switching, our analysis is focused on minimizing UDP packet loss and TCP rebuffering for a single version of high quality video.

A.3 Background on FDSP

This section provides an overview of FDSP, including its architectural features, video streaming using substreams, and factors affecting video quality. For more details, see [2], [3] and [4].

A.3.1 FDSP Architecture

FDSP is a new hybrid streaming protocol that combines the reliability of TCP with the low latency characteristics of UDP. Figure A.3 shows the FDSP architecture consisting of a sender and a receiver.

At the sender, H.264 video data is first processed by the *H.264 Syntax Parser* in order to detect critical NAL units, i.e., SPS, PPS, and slice headers. The rest of the NAL units are primarily slice data. The *RTP Packetizer* then encapsulates each NAL unit according

to the RTP Payload Format for H.264 video [13]. The *Demultiplexer* then directs the RTP packets containing critical NAL units to the TCP stream and the rest to the UDP stream. Meanwhile, *Dual Tunneling* keeps both TCP and UDP sessions active during video streaming. The *BP Selection* module sets the Bitstream Prioritization (BP) parameter, which represents a percentage of I-frame data that is also sent via TCP in addition to the original critical NAL units. BP is adjusted dynamically according to available network bandwidth.

In the receiver, the *Multiplexer* discards late UDP packets while rearranging TCP and UDP packets based on their RTP timestamps. This reordering is important for the *H.264 Decoder* to receive NAL units in the original decodable bitstream order.

A.3.2 FDSP Video Streaming using Substreams

During streaming, the FDSP sender subdivides the video into 10-second substreams. This minimizes initial buffering since all the TCP packets for a particular substream must be received before playback begins for that substream. To further minimize rebuffering, the TCP packets for the next substream are sent at the same time as the UDP packets for the current substream. This is called *substream overlapping* as illustrated in Figure A.4. However, rebuffering occurs whenever TCP packets for the current substream are not yet all available and the receiver has to wait. The rebuffering time then postpones the playout deadline for all subsequent packets.

Substream overlapping is initiated only when the network device's IP queue is below the TCP threshold as shown in Figure A.5. This prevents having too many TCP packets for the next substream in the queue, which would interfere with successful UDP packet transmission for the current substream (see Section A.3.3 for more details on UDP packet loss). In [2], it was determined that 30% of the queue limit was the most suitable threshold.

A.3.3 Factors Influencing Video Quality in FDSP Streaming

FDSP-streamed video quality is affected by several factors, i.e., queue size, TCP threshold, estimated available bandwidth, BP parameter, and substream length. Since the main focus of this paper is to examine how different substream lengths affect FDSP video streaming, this subsection briefly discusses the other four factors and their impact on FDSP. These four

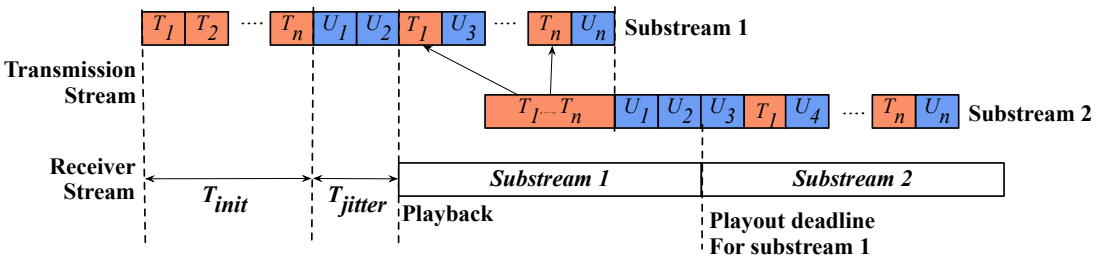


Figure A.4: Substream overlapping. Each packet is either UDP (U) or TCP (T), where the subscript represents the packet number within a substream.

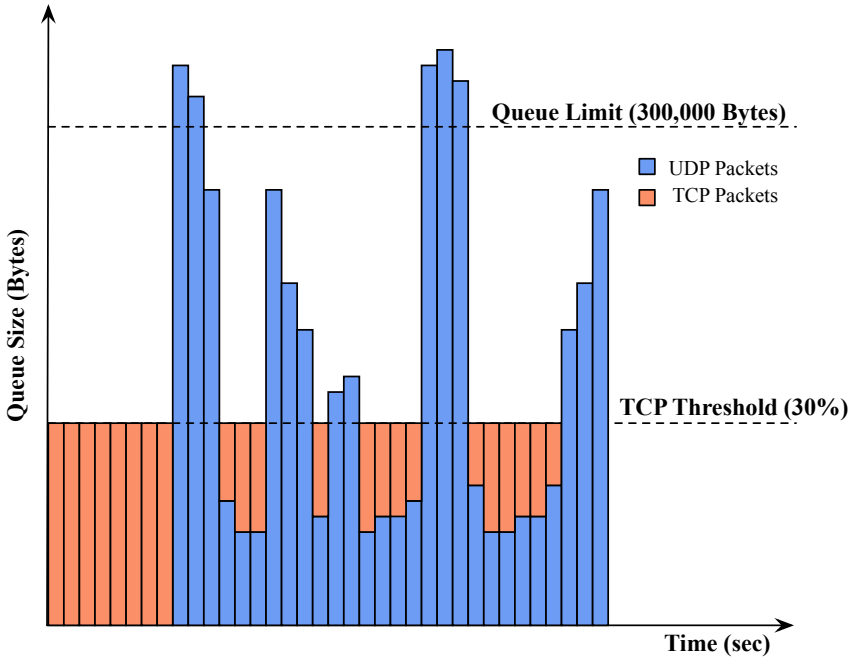


Figure A.5: IP queue containing both UDP and TCP packets. TCP packets are inserted when the queue size is below 30% of the limit (300 Kbytes). UDP packets inserted beyond the limit are considered lost.

factors are kept constant during our experiments (see Section B.5 for more details on the experiment setup).

A.3.3.1 Queue Size

This is the number of packets staged in the network device's IP queue prior to transmission. When the IP queue's limit is reached, any extra packets that are inserted are lost, which are referred to as queue drops. In FDSP, queue drops affect UDP packets, resulting in degraded video quality due to packet loss and error propagation.

A.3.3.2 TCP Threshold

This is the queue size threshold below which TCP packets can be inserted into the queue. The threshold is chosen such that TCP packets do not saturate the network due to retransmissions, which would lead to increased UDP packet loss and TCP rebuffering. Therefore, the TCP threshold helps mitigate these effects.

A.3.3.3 Estimated Available Bandwidth and BP

TCP round-trip time and UDP packet loss are used to estimate the available bandwidth, which determines how much data should be sent via TCP versus UDP based on the BP parameter. This further reduces UDP packet loss while keeping TCP rebuffering time and instances low.

A.4 Experimental Setup

For our experiments, two full HD (1920×1080 @30fps) 3-minute clips from a high-motion (animation) video *Bunny*, and a low-motion (documentary) video *Bears* are used. These videos are encoded using x264 with an average bit rate of 4 Mbps with four slices per frame. Our simulation environment is Open Evaluation Framework For Multimedia Over Networks (OEFMON) [14], which is composed of a multimedia framework, DirectShow, and a network simulator, QualNet 7.3 [15].

OEFMON, which originally worked with only single-slice video, was modified to allow for simulation of multi-slice H.264 videos. Within OEFMON, an 802.11g[1] ad-hoc network with a total bandwidth of 54 Mbps is set up as shown in Figure B.5. The distance between each source and destination pair is 5 m and the distance between pairs of nodes is 10 m. These distances were chosen to mimic the proximity of multiple pairs of neighboring

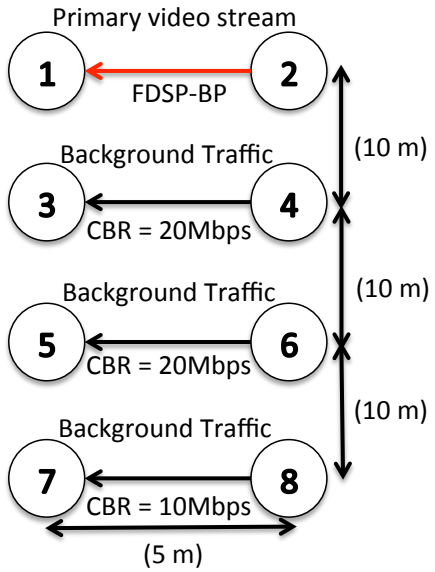


Figure A.6: Network configuration with 54 Mbps total bandwidth for simulating FDSP-based streaming with background traffic.

streaming devices in an apartment setting. The primary video is streamed between nodes 1 and 2. At the same time, the remaining three node pairs simulate either a partially or fully congested network by producing three constant bitrate (CBR) background streams that induce interference within Carrier Sense Multiple Access (CSMA) range of the primary 4 Mbps video stream. The partially congested configuration has a total 20 Mbps of background CBR traffic (i.e., 5, 10, and 10 Mbps streams), while the fully congested configuration has a total of 50 Mbps of background CBR traffic (i.e., 10, 20, and 20 Mbps streams).

The primary video is streamed using FDSP with BP of 0% and 100%. These two choices of BP values are based on our prior work [3], which showed that they represent the two extreme effects of FDSP-based streaming, i.e., UDP PLR and TCP rebuffering are maximized at BP of 0% and 100%, respectively, for 10-second substreams. Therefore, UDP PLR and TCP rebuffering are effectively separated via their corresponding BP values in order to study how substream length changes affect FDSP. For each BP value, 15 different substream lengths, ranging from 1 to 15 seconds, are evaluated for each video.

Finally, the IP queue size limit and the TCP threshold are set at 300 Kbytes and 30%, respectively, as shown in Figure A.5. Most video streaming applications today use a dynamic

queue size that is application specific [16]. For our simulations, a fixed queue size of 300 Kbytes is used.

A.5 Results

This section presents the results of our study on how substream length affects FDSP streaming in terms of TCP rebuffering and UDP PLR.

A.5.1 Impact of Substream Length on TCP Rebuffering

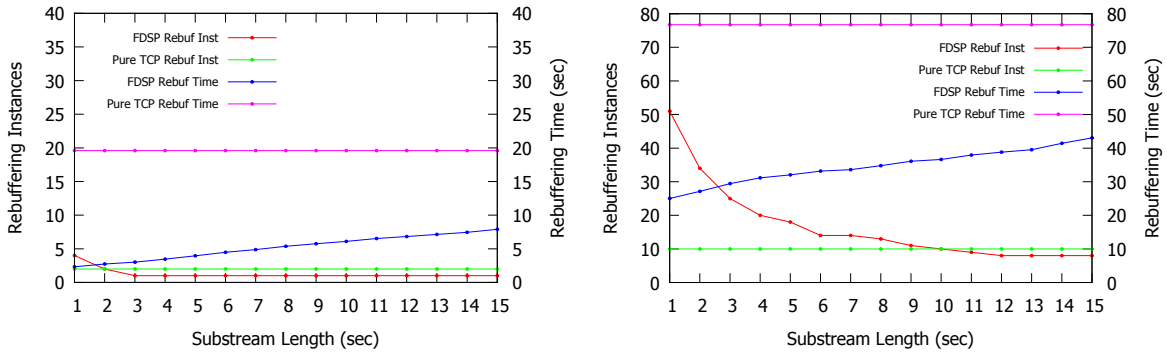
Figures A.7 and A.8 show the effects of substream length on the two test videos in both partially and fully congested networks, respectively, with BP of 100%.

For the partially congested network scenario, the number of FDSP rebuffering instances for both videos decreases slightly as the substream length increases, and then remains constant. For example, in the *Bunny* video (Figure A.7a), substream lengths of 1 and 2 seconds incur 6 and 2 rebuffering instances with a total rebuffering time of 3.26 and 3.34 seconds, respectively. In the *Bears* video (Figure A.8a), 1- and 2-second substreams incur 8 and 5 rebuffering instances with a total rebuffering times of 2.41 and 2.87 seconds, respectively. The results in Figures A.7a and A.8a also show that even with sufficient bandwidth, pure-TCP rebuffering still occurs and is greater than FDSP-TCP rebuffering. For example, the total pure-TCP rebuffering times for *Bunny* and *Bears* are 19.6 and 16 seconds, respectively.

In comparison to the partially congested network scenario, the rebuffering results for the fully congested network scenario are much more pronounced. For example, in the *Bunny* video (Figure A.7b), there are 51 instances of rebuffering for 1-second substreams, compared to 8 instances for 15-second substreams. This is because longer substreams result in fewer substreams for a given video duration, and thus fewer substream transitions. Therefore, there are correspondingly fewer opportunities for the video to stall. Meanwhile, 1-second substreams have a total buffering time of 26.52 seconds compared to 42 seconds for 15-second substreams.

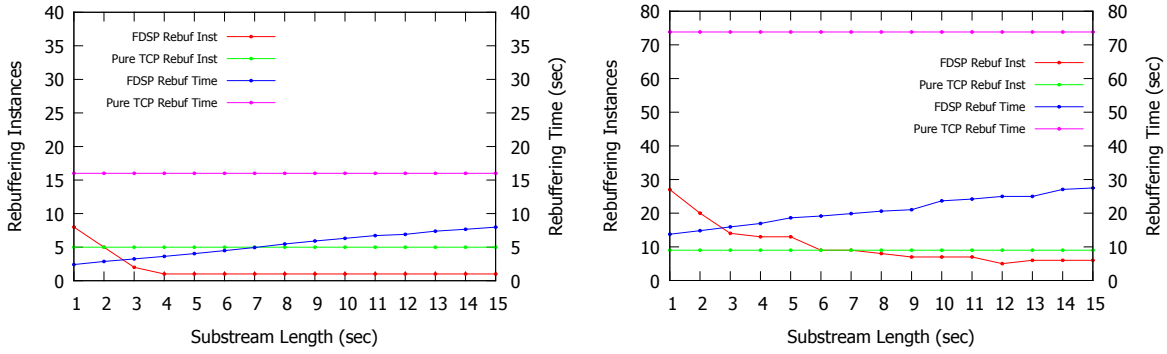
Figure A.8b shows that the rebuffering characteristics of *Bears* are similar to those of *Bunny* for the fully congested network scenario. However, rebuffering in *Bears* is less

[1]The version of the QualNet simulator used for our study only supports the IEEE 802.11g standard. However, the simulation study can be easily adopted to 802.11n and 802.11ac by having more background traffic to saturate the network.



(a) Partially congested network (b) Fully congested network

Figure A.7: Impact of substream length on TCP rebuffering time and instances in *Bunny* in partially and fully congested networks with BP of 100%.



(a) Partially congested network (b) Fully congested network

Figure A.8: Impact of substream length on TCP rebuffering time and instances in *Bears* in partially and fully congested networks with BP of 100%.

pronounced than that in *Bunny* due to lower level of motion. The *Bunny* video has a significantly higher level of motion than *Bears*, and thus includes more TCP packets for I-frames [17]. This is especially the case for BP of 100%, where all I-frame data is sent through TCP.

Note that the overall number of TCP packets for a video at a particular BP value remains constant in spite of changes in substream length. However, a longer substream will result in a larger number of UDP packets in the IP queue, which causes the queue size to be larger than

the TCP threshold for longer periods. Therefore, there are fewer opportunities to insert TCP packets into the IP queue. This means that less TCP packets are sent through substream overlapping and, instead, more are buffered in between substreams, thus increasing the total TCP rebuffering time. Furthermore, the network becomes more saturated as the IP queue fills up with more UDP packets. This increases the TCP average round-trip time and packet loss, and thus the number of retransmissions.

Nevertheless, the FDSP rebuffering times for both scenarios are less than pure-TCP rebuffering time, which is consistent with results from our previous work [3]. For this reason, the average time per instance of pure-TCP rebuffering is high despite the corresponding low number of rebuffering instances. For example, in the fully congested network scenario with *Bunny*, there are just 10 instances of pure-TCP rebuffering, but each one lasts an average of 7.8 seconds.

In comparison to BP of 100%, a BP value of 0% has minimal effect on TCP rebuffering, which is consistent with our prior work for fixed 10-second substreams [3, 4]. For example, in the fully congested network scenario with *Bunny*, only one instance of rebuffering occurs with a total rebuffering time of 1.54 seconds for 1-second substreams compared to 6.3 seconds for 15-second substreams.

A.5.2 Impact of Substream Length on UDP Packet Loss

Figure A.9 shows how substream length affects UDP packet loss with BP of 0% in partially and fully congested network scenarios for both videos. The partially congested network scenario has less PLR than the fully congested one due to higher bandwidth availability. Nonetheless, FDSP streaming in both scenarios exhibits erratic fluctuations in UDP PLR with respect to substream length changes.

Furthermore, there are multiple peaks in PLR at different substream lengths. This behavior can be attributed directly to peaks in TCP transmission time. For example, in Figure A.9b, there are spikes in UDP PLR for substream lengths of 8, 11 and 13 seconds in the fully congested network scenario. Table A.1 shows the relationship between the total TCP Transmission time and UDP PLR for these substream lengths (indicated in bold) as well as the adjacent ones. As can be seen, peaks in UDP PLR correspond to peaks in TCP transmission time. At the same time, TCP transmission time is affected by factors such as IP queue limit, average IP queue level, and TCP threshold. Analysis of these factors in the

context of UDP packet loss is left as a future work. Nevertheless, UDP packet loss for FDSP at different substream lengths is always less than pure-UDP packet loss, which is consistent with results in our previous work [2].

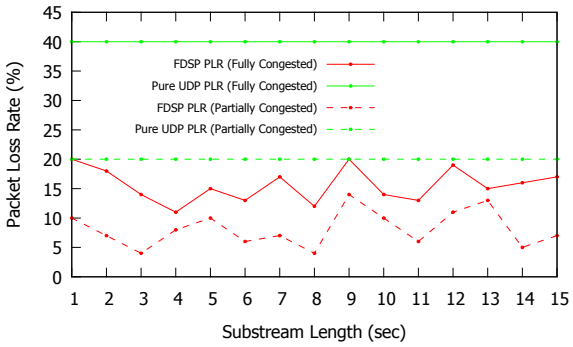
These results show that there is no significant correlation between substream length changes and UDP PLR. Our results also show that there is minimal UDP packet loss for BP of 100%, i.e., less than 1%. This is because the high ratio of TCP-to-UDP packets decreases the chances of UDP packet loss. This matches our results for fixed 10-second substreams [3, 4].

A.5.3 Recommendation on Substream Length Choice

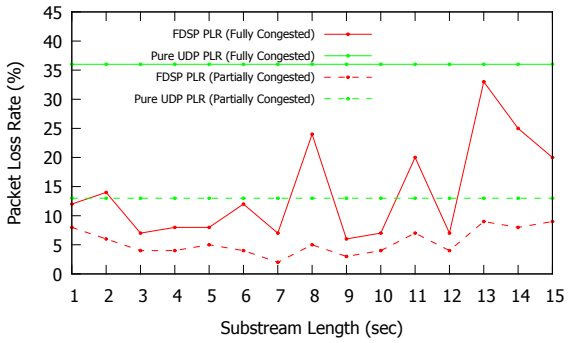
Our study shows that the most appropriate range of substream length is from 6 to 10 seconds. This is based on the fully congested network scenario because it affects TCP rebuffering and UDP PLR more adversely compared to the partially congested network scenario. The lower bound is chosen because the number of rebuffering instances increases almost exponentially for every 1-second decrease in substream length below 6 seconds (see Figures A.7b and A.8b). Furthermore, the number of rebuffering instances decreases up to the 12-second substream mark. Beyond this, there is no further improvement in rebuffering instances. However, the upper bound on substream length was chosen to be 10 seconds based on suitable initial buffering time. The 6-second and 10-second substream length videos have an initial buffering of about 5 seconds and 8 seconds, respectively, under the fully-congested network condition. However, in an ideal network condition, this initial delay is only 3 seconds and 5 seconds, respectively. These results show that initial delay incurred by FDSP streamed videos falls

Substream Length (sec)	Total TCP Tx Time (sec)	UDP PLR (%)
7	96.83	7
8	**99.64**	**25**
9	95.26	6
10	95.48	7
11	**98.96**	**20**
12	95.2	7
13	**100.12**	**33**
14	96.83	25

Table A.1: Relationship between the total TCP transmission time and UDP PLR in *Bears*. The bold entries correspond to spikes in UDP PLR.



(a) High-motion video (*Bunny* animation)



(b) Low-motion video (*Bears* documentary)

Figure A.9: Impact of substream length on UDP packet loss in partially and fully congested network scenarios with BP of 0%.

well within the user-acceptable range of 8 to 16 seconds [18].

Our recommendation for substream length choices does not take into account UDP PLR, which requires further analysis based on the results as discussed in Section A.5.2.

A.6 Conclusion and Future Work

This paper analyzed the effect that different substream lengths have on video streaming in the context of Flexible Dual-TCP/UDP Streaming Protocol (FDSP). Our study showed that substream lengths have a direct effect on TCP rebuffering time and instances. As substream length increases, TCP rebuffering time decreases while the number of TCP rebuffering instances increases. However, substream length changes do not have a clear correlation with

UDP packet loss. This lack of correlation is mainly attributed to its dependence on other parameters such as IP queue size, TCP threshold, etc. Our results show that a substream length of 6 to 10 seconds is best suited for FDSP video streaming.

As future work, we plan to analyze the effects of changing IP queue limit and TCP threshold together with varying substream length in order to further investigate FDSP-UDP packet loss. We also plan on modifying BP and substream length together towards developing a system with dynamic substream length modification.

Adaptive Queue Management Scheme for Flexible Dual TCP/UDP Streaming Protocol

Arul Dhamodaran, Kevin Gatimu, and Ben Lee

Proceedings of the 9th International Conferences on Advances in Multimedia (MMEDIA), Venice, Italy, April 23-27, 2017

Appendix B: Adaptive Queue Management Scheme for Flexible Dual TCP/UDP Streaming Protocol

Flexible Dual-TCP/UDP Streaming Protocol (FDSP) is a new method for streaming H.264-encoded High-definition (HD) video over wireless networks. FDSP streaming is done in sequential video segments or chunks called substreams. In FDSP, substream lengths are used to control the amount of Transmission Control Protocol (TCP) data that needs to be sent prior to the playback of that substream. To avoid frequent rebuffering, TCP packets of the next substream are overlapped with the User Datagram Protocol (UDP) packets of the current substream. The TCP threshold parameter determines when to overlap new TCP packets with the current UDP stream. This paper analyzes the TCP threshold parameter in the context of FDSP. Our results show that user Quality of Experience (QoE) can be enhanced by adaptive adjusting of the TCP threshold using the additive-increase/multiplicative-decrease (AIMD) algorithm based on the UDP packet loss rate and the TCP rebuffering.

B.1 Introduction

High-definition (HD) video streaming technologies have fundamentally changed the way multimedia content is consumed. These video streaming applications can be broadly classified into client-server and device-to-device streaming applications. Client-server based streaming applications, such as Netflix, Hulu, Amazon Video, etc., typically involve a streaming server to deliver multimedia content to the end user through the internet. On the other hand, device-to-device wireless HD video streaming is enabled by technologies such as Apple AirPlay®, Google Chromecast®, and Wi-Fi Alliance's Miracast® to facilitate various multi-screen (i.e., N-screen) applications [1]. However, the explosion of wireless enabled devices will strain the bandwidth limits due to the need to support multiple streams in the same network.

All the aforementioned video streaming services and applications rely on either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) protocol. The client-server streaming techniques rely primarily on HTTP-based streaming, which, in-turn, is based on

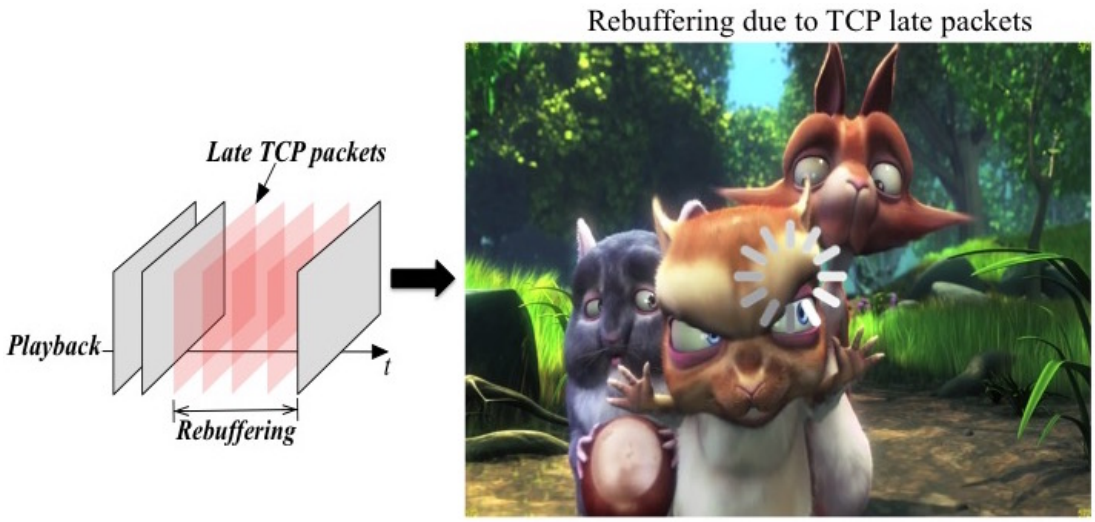


Figure B.1: Rebuffering due to late TCP packets.

TCP. On the other hand, device-to-device streaming applications Chromecast and Miracast rely on UDP while Airplay uses TCP for video streaming and screen mirroring applications. However, both TCP- and UDP-based streaming protocols have their own set of challenges. TCP guarantees packet delivery ensuring perfect video frame quality, but suffers from freeze frames during video playback due to packet delay caused by bandwidth bottleneck. Figure B.1 illustrates the effect of rebuffering caused by TCP packet delay, which occurs when TCP packets arrive at the receiver after the playout deadline due to network congestion. This delay causes the received video to freeze frame and stall for more TCP packets to arrive at the receiver before resuming playback. UDP, on the other hand, minimizes delay but suffers from packet loss. Figure B.2 illustrates the video quality degradation due to UDP packet loss, which affects not only the frame for which the packet loss occurred but also subsequent frames that use it as the reference frame (referred to as *error propagation*).

In our previous work, a new H.264 based video streaming technique called *Flexible Dual Streaming Protocol* (FDSP) was proposed [2]. FDSP sends packets containing important H.264 video syntax elements (i.e., Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and slice headers) via TCP for guaranteed delivery and the rest of the slice data packets via UDP giving an H.264 decoder a better chance of decoding received video even when packet losses occur. Therefore, FDSP exploits the combined benefits of TCP and UDP

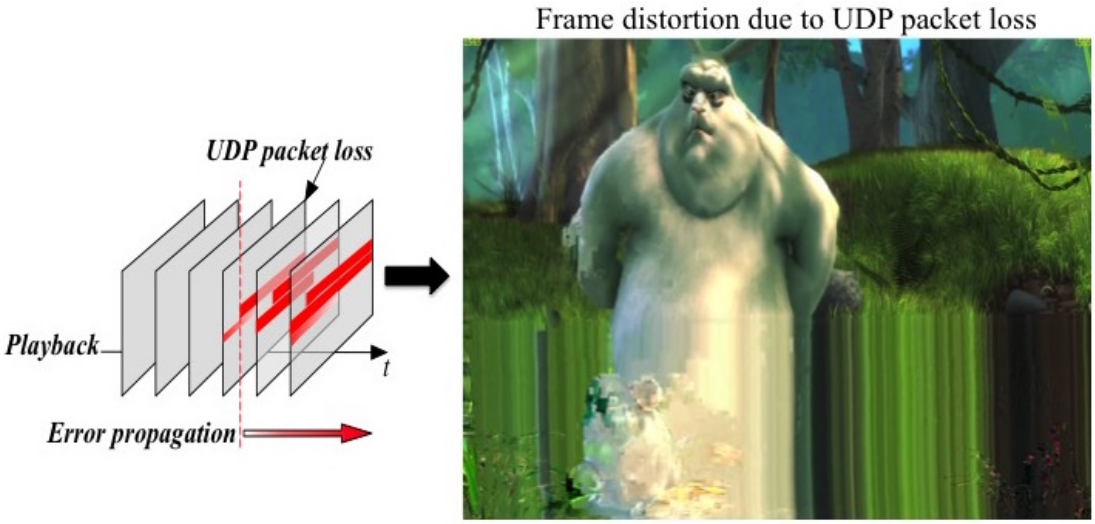


Figure B.2: Frame distortion due to UDP packet loss. Note that packet loss also causes frame distortion in subsequent frames due to error propagation.

by adding reliability to UDP while reducing the latency caused by TCP. This enables FDSP to strike a balance between visual quality and delay by achieving higher video quality than pure-UDP and less rebuffering than pure-TCP.

FDSP was enhanced in [3] using *Bitstream Prioritization* (BP) to reduce the impact of UDP packet loss. This method *statically* chooses the BP metric to classify a select percentage of originally UDP-designated packets from an H.264 bitstream as high priority, which are then transported over TCP for guaranteed delivery. FDSP-BP was further enhanced by introducing Adaptive-BP [4], where the percentage of packets sent over TCP versus UDP is dynamically adjusted based on the estimated rebuffering time for TCP packets and estimated packet loss ratio (PLR) for UDP packets. FDSP with Adaptive-BP further improved the performance by reducing both packet loss and rebuffering time.

FDSP-based streaming is done in sequential video chunks called *substreams*. For each substream, the important syntax elements are sent first via TCP, and then the rest of the data is sent via UDP. Therefore, the substream length determines the amount of TCP data that needs to be sent prior to the playback of that substream. To allow TCP packets to arrive on time, *substream overlapping* is performed where TCP packets for the next substream are sent at the same time as the UDP packets for the current substream. However, an important

issue with substream overlapping is the decision on when to insert new TCP packets into the outgoing IP queue of the sender, which is referred to as *TCP Threshold*. In our prior work on FDSP, the TCP threshold was chosen to be fixed at 35% of the maximum IP Queue size [3]. This paper analyzes how varying the TCP threshold affects UDP PLR and TCP rebuffering time and develops an Adaptive TCP Threshold technique to improve user Quality of Experience (QoE).

This paper is organized as follows. Section B.2 discusses other TCP and UDP streaming techniques. An overview of the FDSP streaming method is shown in Section B.3. Section B.4 discusses the effects of substream overlapping and TCP threshold. Sections B.5 goes over the experimental setup and Section B.6 discusses the results of our analysis on how the changes in the TCP threshold affect FDSP streaming. Finally, Section B.7 concludes the paper.

B.2 Related Work

Queue management techniques for video streaming applications have been well studied. The two basic approaches proposed for queue management are Random Early Detection (RED) [5] and Blue [6]. Both of these techniques use queue length as an indicator of congestion and use this information to regulate the packet drop rates. Xu *et al.* proposed an active queue management technique for wireless ad hoc networks, called Neighborhood RED (NERD) [7]. This technique uses channel utilization to estimate the queue length to help determine the packet drop probability.

However, all the above queue management techniques are designed for data communication in general, without any consideration for the unique characteristics of video streaming (i.e., multimedia communication). Chen *et al.* proposed an active queue management technique where packets that may potentially be late are actively dropped before they are transmitted to reduce the strain on the network resources and to effectively control the transmission queue length [8]. Shy *et al.* proposed another active queue management (AQM) system, which employs routers that deal with both best-effort traffic flows and multimedia traffic flows [9]. Round trip time (RTT) is used in the packet dropping probability calculations to assure rate reductions in both multimedia and best-effort flows before the queue becomes full.

The aforementioned queue management techniques are designed for video streaming

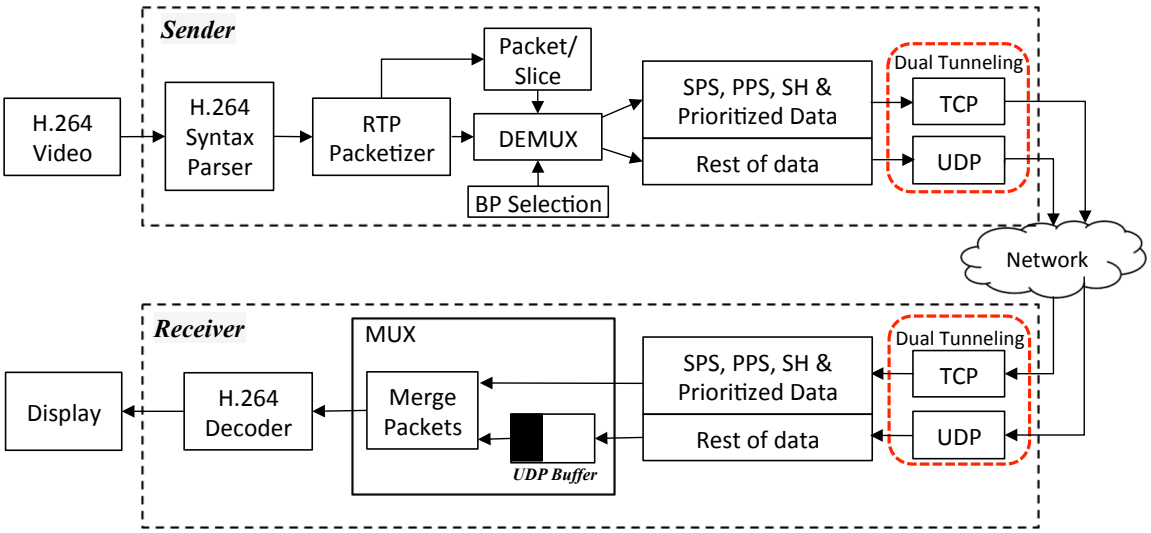


Figure B.3: The architecture of FDSP with Adaptive BP [4]

systems that are based either on TCP or UDP. Furthermore, all of these systems focus on techniques such as prioritized dropping based on queue length, fairness based scheduling algorithms for packet delay optimization, etc. On the other hand, FDSP is a hybrid streaming protocol that uses both TCP and UDP for video streaming, and thus it presents a unique set of challenges, such as the TCP threshold, that play a crucial role in packet delay optimization. Therefore, this paper expands on the scope of our prior research on FDSP with Adaptive-BP [4] by analyzing the TCP threshold parameter and its impact on UDP PLR and TCP rebuffering.

B.3 FDSP Overview

FDSP was proposed as a new device-to-device video streaming technique for H.264 content [2]. This section provides a brief overview of its various architectural features and factors affecting video quality (see [2]-[4] for details).

FDSP with Adaptive-BP architecture is shown in Figure B.3 [4], which consists of a sender and a receiver. The FDSP sender processes H.264 video data using the *H.264 Syntax Parser* to detect important Network Abstraction Layer (NAL) units, i.e., SPS, PPS, and slice headers (SH). The rest of the NAL units are primarily slice data packets. It also works

with the *RTP Packetizer* to encapsulate each NAL unit into the RTP payload format for H.264 video [10]. The *Demultiplexer* (DEMUX) then routes the important NAL units (SPS, PPS, SH, and prioritized I-frame data) through TCP and the rest of the NAL units through UDP. The *BP selection* module sets the BP parameter, which represents the percentage of the I-frame data to be prioritized and sent over TCP. In FDSP with Adaptive-BP, BP is adjusted dynamically based on the estimated available network bandwidth. Finally, *Dual Tunneling* is employed to keep both TCP and UDP sessions active during video streaming.

In the receiver, *Dual Tunneling* is employed to receive packets from both the TCP and UDP streams. The *Multiplexer* (MUX) then discards the late TCP packets and rearranges the TCP and UDP packets based on their RTP timestamps.

During FDSP streaming, the sender first segments the input video into 10 sec. *sub-streams*, as done in HTTP live Streaming (HLS) [11]. Then, all the TCP packets containing SPS, PPS, SH, and BP prioritized I-frame data for each substream are sent over TCP prior to sending UDP packets containing the slice data. Thus, the receiver must wait for its respective TCP data to arrive before playback. To avoid rebuffering caused by TCP packet delay, the transmission of UDP packets for the current substream is overlapped with the transmission of TCP packets for the next substream (i.e., substream overlapping).

BP is only applied to packets containing I-frame data because they serve as reference frames and any loss in I-frame data leads to error propagation to the entire Group Of Picture (GOP) sequence. If the BP parameter is set to zero, then it defaults to basic FDSP, where SPS, PPS, and slice headers are the only packets that will be sent via TCP. If BP is 25% then a quarter of all I-frame packets would be sent via TCP. Although it is possible to select any distribution of the I-frame to be sent via TCP, a sequential order of I-frame packets are selected to be sent via TCP to achieve better QoE. Increasing BP results in increasing the number of TCP packets, thus increasing the probability of TCP rebuffering, but it reduces UDP packet loss and error propagation due to the proportional reduction in the number of UDP packets.

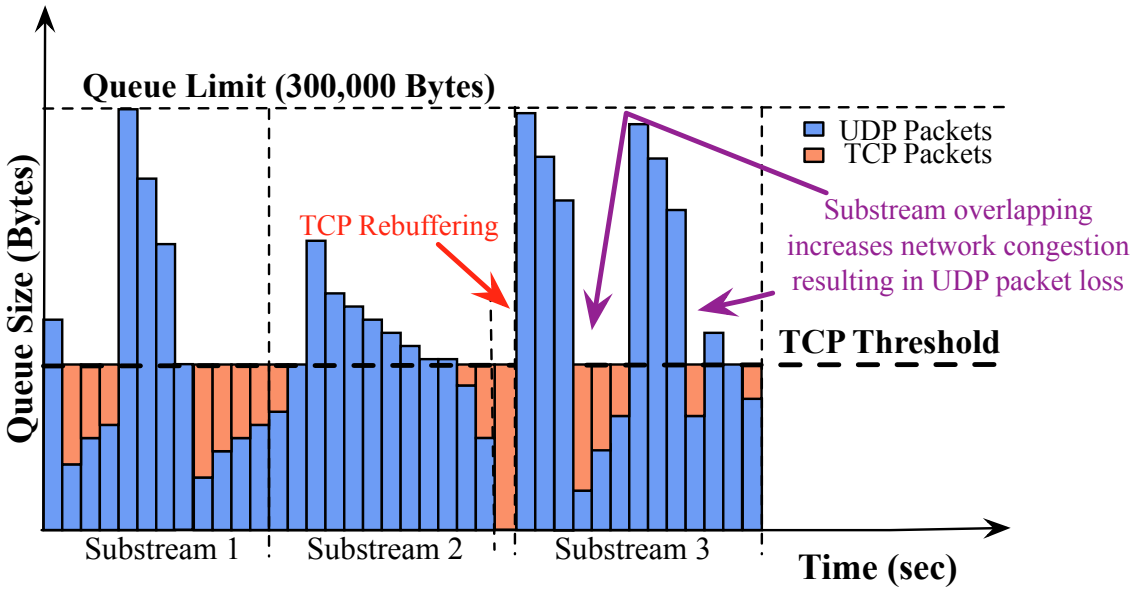
Since FDSP is a hybrid streaming technique that uses both TCP and UDP protocols, its performance is affected by both packet loss and rebuffering. The various factors that influence packet loss and rebuffering are the BP parameter, the substream length, and substream overlapping. The BP parameter is used to determine the percentage of I-frame packets that are to be sent through TCP. The BP parameter is computed based on TCP round-trip time and UDP packet loss rate, which in turn determines the percentage of TCP

versus UDP packets to be sent for each substream. Adaptively adjusting the BP parameter for each substream helps further reduce UDP packet loss while keeping TCP rebuffering time and instances low. The substream length trades off between the likelyhood of rebuffering and the frequency of adaptive BP selection process. Since the BP and substream length have been analyzed in our previous work, this paper focuses on examining how different TCP thresholds affects FDSP video streaming.

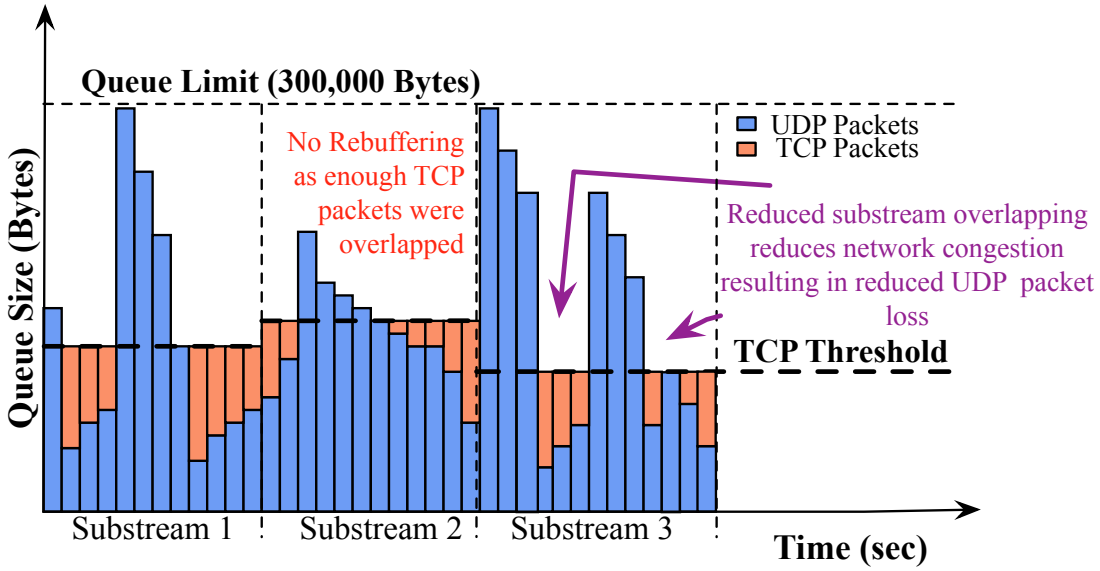
B.4 Substream Overlapping and TCP Threshold

In FDSP, the important syntax elements for each substream are sent first via TCP, and then the rest of the data is sent via UDP. Therefore, the substream length and the BP parameter determine the amount of TCP data that needs to be sent prior to the playback of that substream. However, rebuffering occurs whenever TCP packets for a substream are not yet all available at the time of playback of that substream. To avoid rebuffering, the TCP packets of the next substream (i.e., substream $i + 1$) are overlapped with the UDP packets of the current substream (i.e., substream i).

The important issue with substream overlapping is the decision on when to insert new TCP packets for the next substream into the outgoing IP queue of the sender. This is because inserting TCP packets for the next substream into the queue too soon would interfere with successful UDP packet transmission for the current substream. On the other hand, inserting TCP packets into the queue too late would result in rebuffering. Therefore, substream overlapping is initiated only when the number of packets in the network device's IP queue is below the *TCP Threshold*. Increasing the TCP threshold would result in increased UDP packet loss due to network saturation caused by flooding of TCP packets. On the other hand, decreasing the TCP threshold results in increased TCP rebuffering due to the reduction in the number of new TCP packets being inserted into the queue. The various factors that affect the TCP threshold are: (i) the number of UDP packets in the current substream; (ii) the number of TCP packets in the next substream; (iii) the average queue occupancy; and (iv) the average number of UDP packets per frame. The number of UDP packets for the current substream and the number of TCP packets for the next substream are used to calculate the estimated TCP rebuffering time and the estimated UDP PLR [4]. The average queue occupancy and the average number of UDP packets per frame are used to estimate the number of instances for substream overlapping.



(a) IP queue occupancy for static TCP threshold.



(b) IP queue occupancy for adaptive TCP threshold.

Figure B.4: IP queue occupancy example for static vs. adaptive TCP threshold.

Figure B.4a shows an example of the IP queue occupancy as a function of time for a static TCP threshold during FDSP streaming. In this example, the TCP threshold is set to 30% of the maximum queue size. For substream 1, the queue size decreases as UDP packets are streamed. When the queue size falls below the TCP threshold, the TCP packets for substream 2 are inserted into the queue. In substream 2, the average number of UDP packets per frame is higher than in substream 1 resulting in reduced opportunities to insert new TCP packets from substream 3, which in-turn increases the probability of TCP rebuffering (indicated by an extra time slot). In substream 3, both the queue occupancy and the average number of UDP packets per frame are high indicating a network congestion. In this scenario, inserting more TCP packets into the queue would exacerbate network congestion and result in UDP packet loss.

Figure B.4a clearly shows that having a static TCP threshold is not always optimal for FDSP streaming. Figure B.4b shows that adaptively adjusting the TCP threshold can reduce both UDP PLR and TCP rebuffering resulting in better QoE. In this example, the TCP threshold is also set to 30% by default for substream 1, and thus the queue behavior is identical to that of Figure B.4a. In substream 2, the average number of UDP packets per frame is higher than substream 1, this reduces the number of TCP packet insertions resulting in increased TCP rebuffering probability. However, since the queue occupancy for substream 2 is not very high, the TCP threshold is increased, which increases the number of TCP packets that can be inserted into the queue and in turn reduces the TCP rebuffering probability.

In substream 3, both the queue occupancy and the average number of UDP packets per frame are very high, indicating network congestion, which in-turn increases UDP packet loss. In Figure B.4b, the TCP threshold is decreased for substream 3 to reduce the number of TCP packets inserted into the queue. This in-turn increases the number of UDP packets transmitted, thus reducing UDP PLR. Note that although the TCP threshold is decreased there is no change in the queue level for the first three time slots of substream 3 because there is no substream overlapping. During time slots 4-6, the number of TCP packets inserted into the queue is reduced compared to the case in Figure B.4a. This reduction allows for more UDP packets to be transmitted. The increased UDP packet prioritization reduces the queue occupancy levels during time slots 7-9, which reduces the overall UDP packet loss rate for substream 3. This reduction in TCP threshold may increase the TCP rebuffering probability, but there are enough opportunities for TCP overlapping for substream 3 to

avoid rebuffering.

B.4.1 Adaptive TCP Threshold

The Additive-Increase/Multiplicative-Decrease (AIMD) algorithm is well suited to adaptively adjust the TCP threshold because the TCP threshold is progressively increased, which in-turn increases the number of TCP packets inserted into the queue reducing the likelihood of rebuffering. On the other hand, AIMD prioritizes UDP packets by exponentially reducing the TCP threshold at the first sign of network congestion (based on estimated UDP packet loss). For example, FDSP streaming begins with a default TCP threshold of 30% for the first two substreams. From the third substream on, the TCP threshold is progressively incremented based on the estimated TCP rebuffering probability until UDP packet loss is estimated at which point the TCP threshold is reduced in half. Note that the estimated TCP rebuffering probability and the estimated UDP PLR are computed using TCP round trip time and queue statistics, respectively (for more details please refer to FDSP with Adaptive-BP [4]).

B.5 Experimental Setup

This section discusses the experimental setup for the analysis of TCP threshold parameters and its impact on both UDP packet loss and TCP rebuffering for FDSP-based video streaming. For our experiments, two full HD (1920×1080 @30fps, 4300 frames) clips from a high-motion (animation) video *Bunny*, and a low-motion (documentary) video *Bears* are used. These clips are encoded using the x264 encoder with an average bit rate of 4 Mbps and four slices per frame.

Our simulation environment is *Open Evaluation Framework For Multimedia Over Networks* (OEFMON) [12], which is composed of a multimedia framework *DirectShow*, and a network simulator *QualNet* 7.3 [13]. OEFMON allows a raw video to be encoded and redirected to a simulated network to gather statistics on the received video. Within OEFMON, an 802.11g ad-hoc network with a bandwidth of 54 Mbps is setup. Note that the version of the Qualnet simulator used for our study only supports the IEEE 802.11g standard. However, the simulation study can easily be adapted to 802.11n by having more background traffic to saturate the network. The network scenario used is an 8-node configuration shown

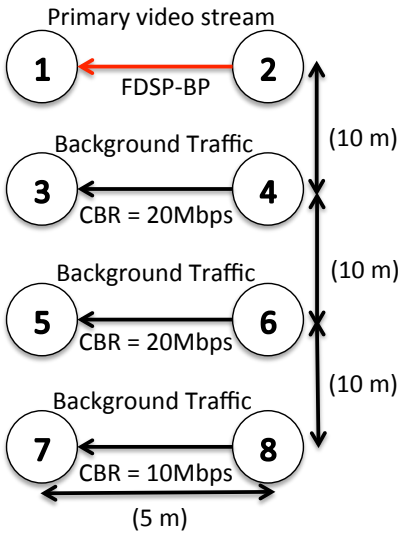
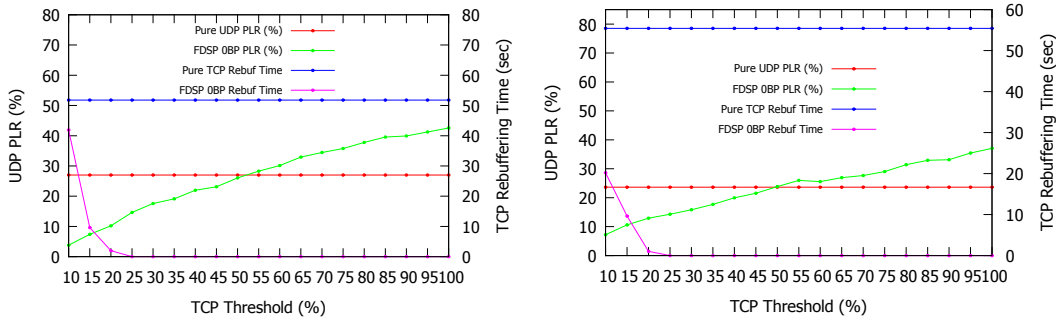


Figure B.5: Simulated network scenario.

in Figure B.5. The distance between each source and destination pair is 5 m and the distance between pairs of nodes is 10 m. These distances were chosen to mimic the proximity of multiple pairs of neighboring streaming devices in an apartment setting. The primary test video is being streamed between nodes 1 and 2, while the remaining three node pairs produces three streams of constant bit rate (CBR) background traffic of 50 Mbps to fully saturate the network.

For the TCP threshold analysis, the primary video is streamed using FDSP with BP of 0% and 100%. These two choices of BP values are based on our prior work [14], which showed that they represent the two extreme effects of FDSP-based streaming, i.e., UDP PLR and TCP rebuffering are maximized at BP of 0% and 100%, respectively, for 10-second substreams. Therefore, the effects of UDP PLR and TCP rebuffering are effectively isolated via their corresponding BP values in order to study how TCP threshold changes affect FDSP streams. For each BP value, 20 different TCP threshold values, ranging from 5% to 100%, are evaluated.



(a) *Bears* video. (b) *Bunny* video.

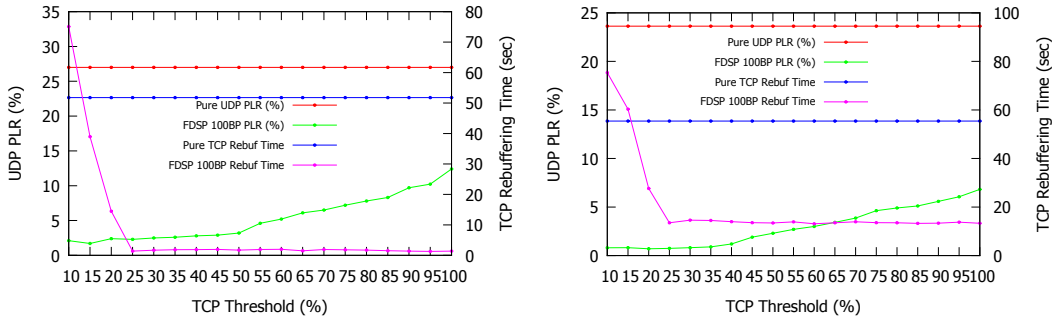
Figure B.6: Impact of TCP threshold changes on UDP packet loss and TCP rebuffering in a fully congested network scenario with BP of 0%.

B.6 Results

B.6.1 Impact of TCP threshold changes on UDP PLR and TCP Rebuffering

Figures B.6 and B.7 show the effects of the TCP threshold changes on both test videos in a fully congested network with BP of 0% and 100%, respectively. These figures also include the results for pure UDP and pure TCP as a comparison. In a fully congested network scenario, the total TCP rebuffering time incurred by both test videos that are streamed using FDSP with 0% BP decreases as the TCP threshold increases. Conversely, the UDP PLR incurred by both test videos increases as the TCP threshold increases. For example, in the *Bears* video (Figure B.6a), TCP thresholds of 10%, 15%, and 20% incur UDP PLRs of 3.8%, 7.4%, and 10.2% and TCP rebuffering times of 43 seconds, 9.8 second, and 1.98 seconds, respectively. Similarly, in the *Bunny* video (Figure B.6b), TCP thresholds of 10%, 15%, and 20% incur UDP PLRs of 7.2%, 10.62%, and 12.93% and TCP rebuffering times of 20.21 seconds, 9.64 seconds, and 1.05 seconds, respectively.

In comparison to FDSP with 0% BP streaming, UDP PLR and TCP rebuffering incurred by FDSP with 100% BP in a fully congested network scenario are much more pronounced. For example, in the *Bears* video (Figure B.7a), TCP thresholds of 10%, 15%, and 20% incur UDP PLRs of 2.1%, 1.7%, and 2.4% and TCP rebuffering times of 75.1 seconds, 38.95 seconds, and 14.47 seconds, respectively. Similarly, in the *Bunny* video (Figure B.7b), TCP

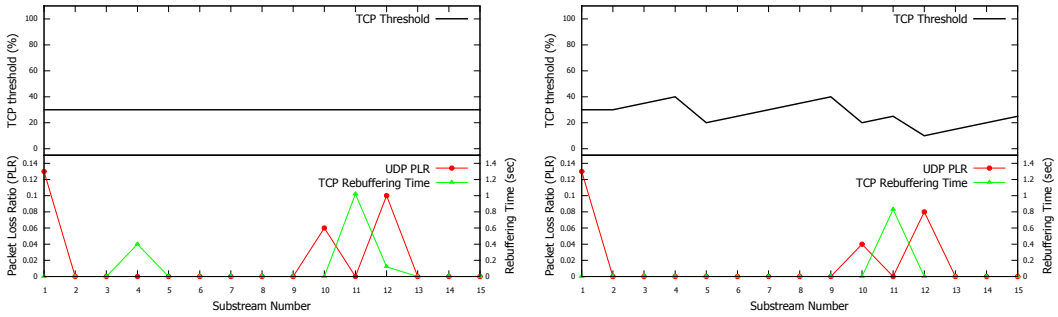


(a) *Bears* video. (b) *Bunny* video.

Figure B.7: Impact of TCP threshold changes on UDP packet loss and TCP rebuffering in a fully congested network scenario with BP of 100%.

thresholds of 10%, 15%, and 20% incur UDP PLRs of 0.8%, 0.81%, and 0.74% and TCP rebuffering times of 75.38 seconds, 60.34 seconds, and 27.63 seconds, respectively. These results show that having a large TCP threshold results in greater opportunities to insert new TCP packets into the IP queue reducing TCP rebuffering time. This, in turn, will cause UDP PLR to increase due to the increase in UDP packet delay resulting in late packets. On the other hand, a smaller TCP threshold results in fewer opportunities to insert TCP packets into the IP queue. This means that fewer TCP packets are sent through substream overlapping and, instead they are buffered in between substreams, thus increasing the total TCP rebuffering time.

The ideal TCP threshold region is the one that minimizes both UDP PLR and TCP rebuffering. For the *Bears* video using FDSP with 0% BP (Figure B.6a), the ideal TCP threshold region lies between 15% to 30%. For the *Bunny* video using FDSP with 0% BP (Figure B.6b), the ideal TCP threshold region lies between 15% to 25%. Similarly, for the *Bears* video using FDSP with 100% BP (Figure B.7a), the ideal TCP threshold region lies between 25 to 50%. For the *Bunny* video with FDSP 100% BP (Figure B.7b), the ideal TCP threshold region lies between 25 to 55%. The optimal threshold range for both videos increases as BP increases to accommodate the increase in the number of TCP packets. These results show that the ideal TCP threshold is not constant for all types of videos. Furthermore, the ideal TCP threshold region is affected by the changes in the BP parameter. Hence, the TCP threshold has to be adaptively adjusted for each substream to



(a) Static TCP threshold results.

(b) Adaptive TCP threshold results.

Figure B.8: Comparison of static vs. adaptive TCP threshold performance of *Bunny* video with Adaptive-BP.

optimize the substream overlapping and improve FDSP's performance.

B.6.2 Performance of Adaptive TCP Threshold

Figures B.8a and B.8b show UDP PLR and TCP rebuffering time for the *Bunny* video streamed based on FDSP with Adaptive-BP using static and adaptive TCP threshold techniques, respectively. Note that the figures are also stacked with plots of the TCP threshold values. These results show that adaptively adjusting the TCP threshold for each substream reduces both UDP PLR and TCP rebuffering time as compared to having a static TCP threshold. For example, using the adaptive TCP threshold scheme incurs only one instance of rebuffering that lasts for 0.83 seconds as compared to the static TCP threshold scheme, which incurs three instances of rebuffering with 0.4 seconds, 1.02 seconds, and 0.12 seconds of rebuffering times. Similarly, the adaptive TCP threshold scheme incurs three instances of UDP packet loss with PLRs of 0.13, 0.04, and 0.08, where as the static TCP threshold scheme also incurs three instances of UDP packet loss but with slightly higher PLRs of 0.13, 0.06 and 0.1. Thus, adaptively adjusting the TCP threshold reduces the TCP rebuffering incurred by 50% and also slightly reduces UDP PLR for the *Bunny* video.

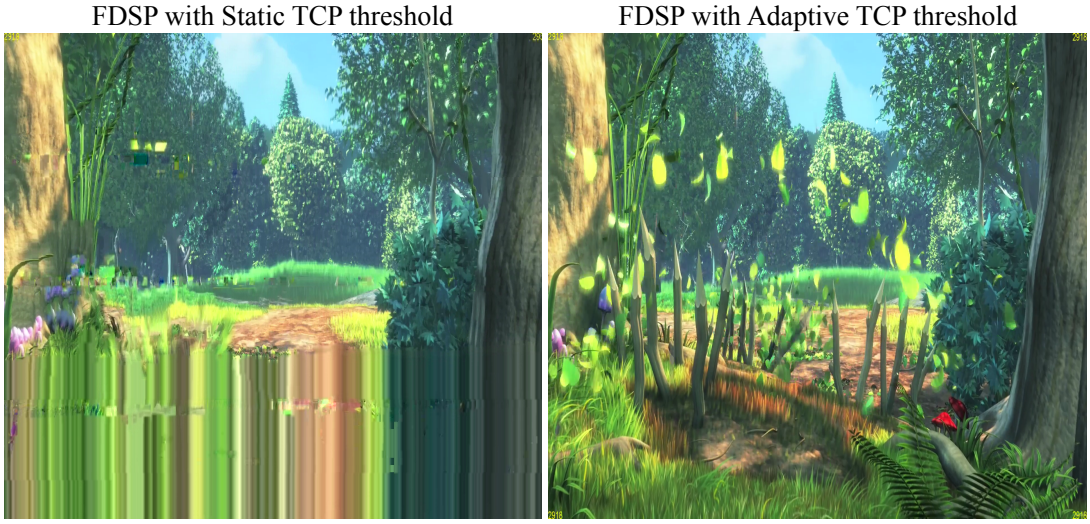
Figures B.9a and B.9b compare the visual quality of the static and adaptive TCP threshold schemes for frames 2918 and 3391, respectively. These figures clearly show that the adaptive TCP threshold scheme achieves better visual quality than the static TCP threshold scheme by reducing UDP PLR as well as TCP rebuffering time. For frame 2918, the

adaptive TCP threshold scheme incurs no packet loss resulting in perfect frame quality. For frame 3391, adaptively adjusting the TCP threshold results in slightly lower packet loss as compared to using a static TCP threshold value even though the difference in visual quality is marginal.

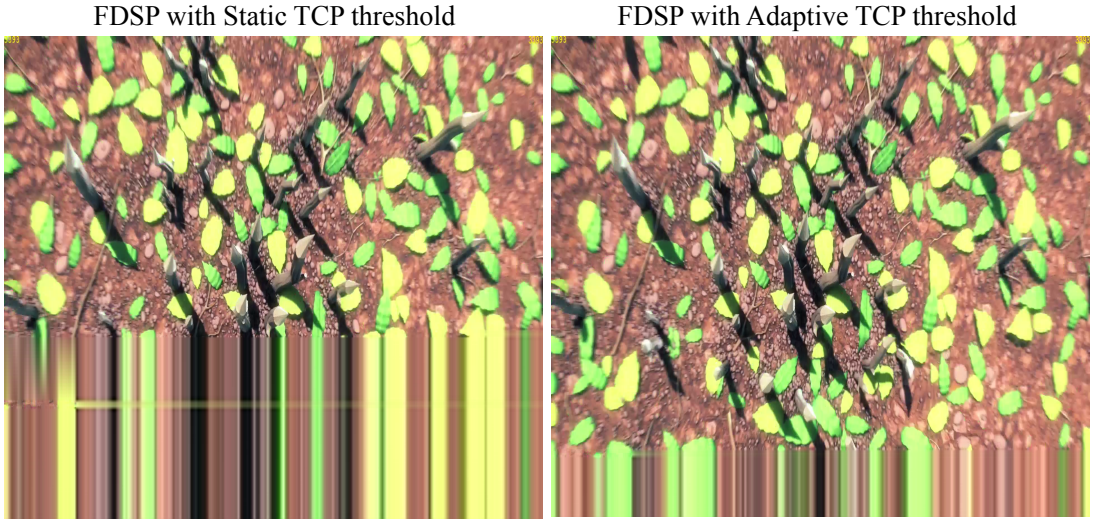
These results clearly show that the adaptive TCP threshold scheme reduces both UDP PLR and TCP rebuffering time as compared to the static TCP threshold scheme resulting in better end user QoE.

B.7 Conclusion and Future Work

This paper studied the effects that different TCP threshold values have on video streaming in the context of Flexible Dual-TCP/UDP Streaming Protocol (FDSP). Our analysis showed that TCP threshold has a direct effect on both TCP rebuffering and UDP PLR. Our results showed that adaptively adjusting the TCP threshold using an AIMD algorithm reduces both packet loss ratio and rebuffering time, and leads to a better overall video streaming experience. As future work, we plan to study the impact of UDP packet loss and TCP rebuffering on end user Quality of Experience for FDSP streaming.



(a) Video quality comparison of Frame 2918.



(b) Video quality comparison of Frame 3391.

Figure B.9: Visual quality comparison of static vs. adaptive TCP threshold for frames 2918 and 3391.

Bibliography

Chapter 1: General Introduction

[1] Zenith Media. *Online video viewing to exceed an hour a day in 2018.*

[2] *VNI Global Fixed and Mobile Internet Traffic Forecasts.* URL: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html (visited on 06/18/2017).

Chapter 2: Evaluation of Wireless High Definition Video Transmission using H.264 over WLANs

[1] URL: http://www.whdi.org/.

[2] URL: http://www.wirelesshd.org/.

[3] URL: http://www.ieee802.org/11/Reports/tgac_update.htm.

[4] URL: http://wirelessgigabitalliance.org/.

[5] C.T. Calafate, M.P. Malumbres, and P. Manzoni. "Performance of H.264 compressed video streams over 802.11b based MANETs". In: *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on.* Mar. 2004, pp. 776–781. DOI: 10.1109/ICDCSW.2004.1284121.

[6] P. Bucciol et al. "Performance Evaluation of H. 264 Video Streaming over Inter-Vehicular 802.11 Ad Hoc Networks". In: *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on.* Vol. 3. Sept. 2005, pp. 1936–1940. DOI: 10.1109/PIMRC.2005.1651778.

[7] E. Masala et al. "Real-Time Transmission of H.264 Video over 802.11B-Based Wireless ad hoc Networks". In: *DSP for In-Vehicle and Mobile Systems.* Ed. by Hüseyin Abut, John H.L. Hansen, and Kazuya Takeda. Springer US, 2005, pp. 193–207. ISBN: 978-0-387-22979-9.

[8] A. Moid and A.O. Fapojuwo. "Application layer optimization for efficient video streaming over IEEE 802.11 based wireless networks". In: *Electrical and Computer Engineering, 2009. CCECE '09. Canadian Conference on.* May 2009, pp. 789–793. DOI: 10.1109/CCECE.2009.5090236.

[9] Andrea Detti et al. "Streaming H.264 scalable video over data distribution service in a wireless environment". In: *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*. June 2010, pp. 1–3. DOI: 10.1109/WOWMOM.2010.5534937.

[10] Deer Li and Jianping Pan. "Performance evaluation of video streaming over multi-hop wireless local area networks". In: *Wireless Communications, IEEE Transactions on* 9.1 (Jan. 2010), pp. 338–347. ISSN: 1536-1276. DOI: 10.1109/TWC.2010.01.090556.

[11] R. Stapenhurst et al. "Adaptive HRD parameter selection for fixed delay live wireless video streaming". In: *Packet Video Workshop (PV), 2010 18th International*. Dec. 2010, pp. 142–149. DOI: 10.1109/PV.2010.5706831.

[12] S. Wenger. "H.264/AVC over IP". In: *Circuits and Systems for Video Technology, IEEE Transactions on* 13.7 (July 2003), pp. 645–656. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2003.814966.

[13] Sunil Kumar et al. "Error resiliency schemes in H.264/AVC standard". In: *Journal of Visual Communication and Image Representation* 17.2 (2006). Introduction: Special Issue on emerging H.264/AVC video coding standard, pp. 425–450. ISSN: 1047-3203. DOI: DOI:10.1016/j.jvcir.2005.04.006.

[14] Xingjun Zhang and Xiaohong Peng. "An unequal packet loss protection scheme for H.264/AVC video transmission". In: *Information Networking, 2009. ICOIN 2009. International Conference on*. Jan. 2009, pp. 1–5.

[15] Xiaolong Wang, Kun Tang, and Huijuan Cui. "A Novel Unequal Error Protection Scheme for ROI Based Video Coding in H.264/AVC". In: *Computer Modeling and Simulation, 2010. ICCMS '10. Second International Conference on*. Vol. 3. Jan. 2010, pp. 175–178. DOI: 10.1109/ICCMS.2010.408.

[16] A. Ksentini, M. Naimi, and A. Gueroui. "Toward an improvement of H.264 video transmission over IEEE 802.11e through a cross-layer architecture". In: *Communications Magazine, IEEE* 44.1 (Jan. 2006), pp. 107–114. ISSN: 0163-6804. DOI: 10.1109/MCOM.2006.1580940.

[17] Byung Joon Oh and Chang Wen Chen. "Performance evaluation of H.264 video over ad hoc networks based on dual mode IEEE 802.11B/G and EDCA MAC architecture". In: *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*. May 2008, pp. 3510–3513. DOI: 10.1109/ISCAS.2008.4542216.

[18] R. MacKenzie, D. Hands, and T. O'Farrell. "Effectiveness of H.264 Error Resilience Techniques in 802.11e WLANs". In: *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*. Apr. 2009, pp. 1–6. DOI: 10.1109/WCNC.2009.4917696.

[19] URL: http://www.apple.com/appletv/.

[20] URL: www.intel.com/WirelessDisplay.

[21] URL: http://www.caviumnetworks.com/PureVu_WiVu-Solution.html.

[22] Chunho Lee et al. "OEFMON: An Open Evaluation Framework for Multimedia Over Networks". In: *IEEE Communications Magazine* (Sept. 2011), pp. 153–161.

[23] URL: http://www.videolan.org/.

[24] URL: http://www.wireshark.org/.

[25] *QualNet 5.0.2 User's Guide*. Scalable Network Technologies, Inc. 2010.

[26] URL: http://blog.monogram.sk/janos/2009/05/20/monogram-x264-encoder-1050/.

[27] URL: http://corecodec.com/products/coreavc.

Chapter 3: Weighted Nearest Valid Motion Vector Averaging for Spatial Motion Vector Recovery in Wireless HD Video Transmission using H.264 over WLANs

[1] T. Wiegand et al. "Overview of the H.264/AVC video coding standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (July 2003), pp. 560–576. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2003.815165.

[2] "IEEE Std: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput". In: *IEEE Std 802.11n-2009 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, and IEEE Std 802.11w-2009)* (29 2009), pp. c1–502. DOI: 10.1109/IEEESTD.2009.5307322.

[3] URL: http://www.ieee802.org/11/Reports/tgac_update.htm.

[4] URL: http://wirelessgigabitalliance.org/.

[5] 2011. URL: ffmpeg.org.

[6] K. Gatimu et al. "Evaluation of wireless high definition video transmission using H.264 over WLANs". In: *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. Jan. 2012, pp. 204–208. DOI: 10.1109/CCNC.2012.6181087.

[7] M.E. Al-Mualla, C.N. Canagarajah, and D.R. Bull. "Motion field interpolation for temporal error concealment". In: *Vision, Image and Signal Processing, IEE Proceedings* - 147.5 (2000), pp. 445–453. ISSN: 1350-245X. DOI: 10.1049/ip-vis:20000385.

[8] Mei-Juan Chen, Che-Shing Chen, and Ming-Chieh Chi. "Temporal error concealment algorithm by recursive block-matching principle". In: *IEEE Transactions on Circuits and Systems for Video Technology* 15.11 (2005), pp. 1385–1393. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2005.857301.

[9] Donghyung Kim, Siyoung Yang, and Jechang Jeong. "A new temporal error concealment method for H.264 using adaptive block sizes". In: *IEEE International Conference on Image Processing, 2005. ICIP 2005*. Vol. 3. 2005, pp. III–928-31. DOI: 10.1109/ICIP.2005.1530545.

[10] Yanling Xu and Yuanhua Zhou. "Adaptive temporal error concealment scheme for H.264/AVC video decoder". In: *IEEE Transactions on Consumer Electronics* 54.4 (2008), pp. 1846–1851. ISSN: 0098-3063. DOI: 10.1109/TCE.2008.4711244.

[11] Arvind Raman and Murali Basu. *A Low Complexity Error Concealment Scheme for MPEG-4 Coded Video Sequences*. 2001.

[12] Myounghoon Kim, Hoonjae Lee, and Sanghoon Sull. "Spatial error concealment for H.264 using sequential directional interpolation". In: *IEEE Transactions on Consumer Electronics* 54.4 (2008), pp. 1811–1818. ISSN: 0098-3063. DOI: 10.1109/TCE.2008.4711239.

[13] O. Nemethova, A. Al-Moghrabi, and M. Rupp. "Flexible Error Concealment for H.264 Based on Directional Interpolation". In: *2005 International Conference on Wireless Networks, Communications and Mobile Computing*. Vol. 2. 2005, pp. 1255–1260. DOI: 10.1109/WIRLES.2005.1549592.

[14] Yi Jin-wang, Cheng En, and Yuan Fei. "An Improved Spatial Error Concealment Algorithm Based on H.264". In: *Third International Symposium on Intelligent Information Technology Application, 2009. IITA 2009*. Vol. 3. 2009, pp. 455–458. DOI: 10.1109/IITA.2009.451.

[15] S. Beesley, A. Armstrong, and C. Grecos. "An Edge Preserving Spatial Error Concealment Technique for the H.264 Video Coding Standard". In: *Research in Microelectronics and Electronics 2006, Ph. D.* 2006, pp. 113–116. DOI: 10.1109/RME.2006.1689909.

[16] P. A. P. Moran. "Notes on Continuous Stochastic Phenomena". English. In: *Biometrika* 37.1/2 (1950), pp. 17–23. ISSN: 00063444. URL: http://www.jstor.org/stable/2332142.

[17] URL: http://www.pysal.org/users/tutorials/weights.html#a-closer-look-at-w.

Chapter 4: Spatial Motion Vector Recovery for Wireless HD Video Transmission over WLANs using Weighted Nearest Valid Motion Vector Averaging

[1] *Home Entertainment Inc.* Aug. 2014. URL: http://www.homeentertainmentinc.com/home-networking/.

[2] *N-Screen Connected Devices 2014*. Aug. 2014. URL: http://www.bogotobogo.com/WebTechnologies/n-screen_connected_devices.php.

[3] *nScreen Deluxe*. 2014 August. URL: http://www.honestech.com/main/nScreenDeluxe.asp.

[4] Google. *Chromecast*. URL: http://www.google.com/intl/en/chrome/devices/chromecast/.

[5] *Screen Mirroring with Chromecast*. Aug. 2014. URL: http://allaboutchromecast.com/screen-mirroring-with-chromecast/.

[6] *Share Your Screen With Intel® Wireless Display (Intel® WiDi)*. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/intel-wireless-display.html.

[7] *AirPlay*. URL: http://www.apple.com/airplay/?cid=wwa-us-kwg-features-com.

[8] T. Wiegand et al. "Overview of the H.264/AVC video coding standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (July 2003), pp. 560–576. ISSN: 1051-8215. DOI: 10.1109/TCSVT.2003.815165.

[9] *YouTube Advanced Encoding Settings*. URL: https://support.google.com/youtube/answer/1722171?hl=en.

[10] URL: http://www.ieee802.org/11/Reports/tgac_update.htm.

[11] URL: http://wirelessgigabitalliance.org/.

[12] K. Gatimu et al. "Weighted nearest valid motion vector averaging for spatial motion vector recovery in wireless HD video transmission using H.264 over WLANs". In: *Network of the Future (NOF), 2013 Fourth International Conference on the*. Oct. 2013, pp. 1–6. DOI: 10.1109/NOF.2013.6724519.

[13] O. Nemethova, A. Al-Moghrabi, and M. Rupp. "Flexible Error Concealment for H.264 Based on Directional Interpolation". In: *2005 International Conference on Wireless Networks, Communications and Mobile Computing*. Vol. 2. 2005, pp. 1255–1260. DOI: 10.1109/WIRLES.2005.1549592.

[14] Jinghong Zheng and L-P Chau. "Efficient motion vector recovery algorithm for H.264 based on a polynomial model". In: *IEEE Transactions on Multimedia* 7.3 (June 2005), pp. 507–513. ISSN: 1520-9210. DOI: 10.1109/TMM.2005.843343.

[15] S. Wenger. *RTP Payload Format for H..264 Video*.

[16] K. Gatimu et al. "Evaluation of wireless high definition video transmission using H.264 over WLANs". In: *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*. Jan. 2012, pp. 204–208. DOI: 10.1109/CCNC.2012.6181087.

[17] 2011. URL: ffmpeg.org.

[18] Saurav K Bandyopadhyay et al. "An Error Concealment Scheme for Entire Frame Losses for H.264/AVC". In: *2006 IEEE Sarnoff Symposium*. IEEE, Mar. 2006, pp. 1–4. ISBN: 978-1-4244-0002-7. DOI: 10.1109/SARNOF.2006.4534755.

[19] Jui-Ting Chien, Gwo-Long Li, and Mei-Juan Chen. "Effective error concealment algorithm of whole frame loss for H.264 video coding standard by recursive motion vector refinement". In: *IEEE Transactions on Consumer Electronics* 56.3 (Aug. 2010), pp. 1689–1695. ISSN: 0098-3063. DOI: 10.1109/TCE.2010.5606314.

[20] Bo Yan and H. Gharavi. "A Hybrid Frame Concealment Algorithm for H.264/AVC". In: *IEEE Transactions on Image Processing* 19.1 (Jan. 2010), pp. 98–107. ISSN: 1057-7149. DOI: 10.1109/TIP.2009.2032311.

[21] Min Li and Bo Wang. "An Entire Frame Loss Recovery Algorithm for H.264/AVC over Wireless Networks". In: *5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09.* Sept. 2009, pp. 1–4. DOI: 10.1109/WICOM.2009.5303179.

[22] Arvind Raman and Murali Basu. *A Low Complexity Error Concealment Scheme for MPEG-4 Coded Video Sequences.* 2001.

[23] Yi Jin-wang, Cheng En, and Yuan Fei. "An Improved Spatial Error Concealment Algorithm Based on H.264". In: *Third International Symposium on Intelligent Information Technology Application, 2009. IITA 2009.* Vol. 3. 2009, pp. 455–458. DOI: 10.1109/IITA.2009.451.

[24] S. Beesley, A. Armstrong, and C. Grecos. "An Edge Preserving Spatial Error Concealment Technique for the H.264 Video Coding Standard". In: *Research in Microelectronics and Electronics 2006, Ph. D.* 2006, pp. 113–116. DOI: 10.1109/RME.2006.1689909.

[25] Jinghong Zheng and L-P Chau. "A motion vector recovery algorithm for digital video using Lagrange interpolation". In: *IEEE Transactions on Broadcasting* 49.4 (Dec. 2003), pp. 383–389. ISSN: 0018-9316. DOI: 10.1109/TBC.2003.819050.

[26] K. Seth, V. Kamakoti, and S. Srinivasan. "Efficient Motion Vector Recovery Algorithm for H.264 Using B-Spline Approximation". English. In: *IEEE Transactions on Broadcasting* 56.4 (Dec. 2010), pp. 467–480. ISSN: 0018-9316. DOI: 10.1109/TBC.2010.2058030.

[27] P. A. P. Moran. "Notes on Continuous Stochastic Phenomena". English. In: *Biometrika* 37.1/2 (1950), pp. 17–23. ISSN: 00063444. URL: http://www.jstor.org/stable/2332142.

Chapter 5: Experimental Study of QoE Improvements Towards Adaptive HD Video Streaming using Flexible Dual TCP-UDP Streaming Protocol

[1] *VNI Global Fixed and Mobile Internet Traffic Forecasts.* URL: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html (visited on 06/18/2017).

[2] *4K Internet TV & Video to be Viewed by 1 in 10 US Residents*. Aug. 2016. URL: https://www.juniperresearch.com/press/press-releases/4k-internet-tv-video-content-to-be-viewed-by-1-i (visited on 06/19/2017).

[3] *Ericsson Mobility Report – Ericsson*. SectionStartPage. Nov. 2016. URL: https://www.ericsson.com/en/mobility-report/reports/november-2017/key-figures (visited on 06/19/2017).

[4] Alex Zambelli. *Smooth Streaming Technical Overview*. URL: http://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview.

[5] Adobe Systems. *HTTP Dynamic Streaming*. URL: http://www.adobe.com/products/hds-dynamic-streaming.html.

[6] Apple Inc. *HTTP Live Streaming Internet—Draft*. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[7] *ISO/IEC 23009-1:2012 - Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623 (visited on 06/22/2016).

[8] *What YOU Need to Know About HLS: Pros and Cons*. Jan. 2016. URL: http://blog.red5pro.com/what-you-need-to-know-about-hls-pros-and-cons/ (visited on 06/19/2017).

[9] Tom Bova and Ted Krivoruchka. *Reliable UDP Protocol*. URL: https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00 (visited on 02/24/2017).

[10] Alyssa Wilk et al. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. URL: https://tools.ietf.org/html/draft-hamilton-early-deployment-quic-00 (visited on 02/24/2017).

[11] Christian Timmerer and Alan Bertoni. "Advanced Transport Options for the Dynamic Adaptive Streaming over HTTP". In: *arXiv preprint arXiv:1606.00264* (2016). URL: https://arxiv.org/abs/1606.00264 (visited on 06/19/2017).

[12] X. Liu, H. Yin, and C. Lin. "A Novel and High-Quality Measurement Study of Commercial CDN-P2P Live Streaming". In: *2009 WRI International Conference on Communications and Mobile Computing*. Vol. 3. Jan. 2009, pp. 325–329. DOI: 10.1109/CMC.2009.152.

[13] ZhiHui Lu, Ye Wang, and Yang Richard Yang. "An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model". In: *Journal of Communications* 7.3 (Mar. 2012). ISSN: 1796-2021. DOI: 10.4304/jcm.7.3.232-245. URL: http://www.jocm.us/index.php?m=content&c=index&a=show&catid=39&id=90 (visited on 06/21/2017).

[14] Jing Zhao et al. “Flexible Dual TCP/UDP Streaming for H.264 HD Video over WLANs”. In: *Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC 2013)*. Kota Kinabalu, Malaysia, 2013, 34:1–34:9. ISBN: 978-1-4503-1958-4. DOI: 10.1145/2448556.2448590.

[15] Mohammed Sinky et al. “Analysis of H.264 Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video Over WLANs”. In: *IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*. Las Vegas, USA, Jan. 2015, pp. 576–581.

[16] Arul Dhamodaran, Mohammed Sinky, and Ben Lee. “Adaptive Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video”. In: *The Tenth International Conference on Systems and Networks Communications (ICSNC 2015)*. Barcelona, Spain, Nov. 2015, pp. 35–40.

[17] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. “Rate Adaptation for Adaptive HTTP Streaming”. In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. MMSys ’11. New York, NY, USA: ACM, 2011, pp. 169–174. ISBN: 978-1-4503-0518-1. DOI: 10.1145/1943552.1943575. URL: http://doi.acm.org/10.1145/1943552.1943575 (visited on 06/25/2016).

[18] V. Swaminathan and S. Wei. “Low latency live video streaming using HTTP chunked encoding”. In: *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. Oct. 2011, pp. 1–6. DOI: 10.1109/MMSP.2011.6093825.

[19] Greg Wilkins et al. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. URL: https://tools.ietf.org/html/rfc6202#page-16 (visited on 07/19/2017).

[20] Sheng Wei and Viswanathan Swaminathan. “Low Latency Live Video Streaming over HTTP 2.0”. In: *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. NOSSDAV ’14. New York, NY, USA: ACM, 2014, 37:37–37:42. ISBN: 978-1-4503-2706-0. DOI: 10.1145/2578260.2578277. URL: http://doi.acm.org/10.1145/2578260.2578277 (visited on 06/20/2017).

[21] Wael Cherif et al. “DASH Fast Start Using HTTP/2”. In: *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV ’15. New York, NY, USA: ACM, 2015, pp. 25–30. ISBN: 978-1-4503-3352-8. DOI: 10.1145/2736084.2736088. URL: http://doi.acm.org/10.1145/2736084.2736088 (visited on 06/21/2017).

[22] Rafael Huysegems et al. “HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming”. In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM ’15. New York, NY, USA: ACM, 2015, pp. 541–550. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806264. URL: http://doi.acm.org/10.1145/2733373.2806264 (visited on 06/21/2017).

[23] Alex Theedom. *Tracking HTTP/2 Adoption: Stagnation - DZone Web Dev*. 2016. URL: https://dzone.com/articles/tracking-http2-adoption-stagnation (visited on 06/21/2017).

[24] Jacob Chakareski et al. "System and method for low delay, interactive communication using multiple TCP connections and scalable coding". US8699522 B2. U.S. Classification 370/474, 370/536, 375/240.05, 709/231; International Classification H04J3/24; Cooperative Classification H04L65/607, H04L47/32, H04L47/10, H04L47/193, H04L47/2416, H04L65/4015, H04L47/283, H04L65/80. Apr. 2014. URL: http://www.google.com/patents/US8699522 (visited on 06/20/2017).

[25] P. Houzé et al. "Applicative-layer multipath for low-latency adaptive live streaming". In: *2016 IEEE International Conference on Communications (ICC)*. May 2016, pp. 1–7. DOI: 10.1109/ICC.2016.7511550.

[26] Dongyan Xu et al. "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution". en. In: *Multimedia Systems* 11.4 (Apr. 2006), pp. 383–399. ISSN: 0942-4962, 1432-1882. DOI: 10.1007/s00530-006-0015-3. URL: https://link.springer.com/article/10.1007/s00530-006-0015-3 (visited on 06/21/2017).

[27] S. M. Y. Seyyedi and B. Akbari. "Hybrid CDN-P2P architectures for live video streaming: Comparative study of connected and unconnected meshes". In: *2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*. Feb. 2011, pp. 175–180. DOI: 10.1109/CNDS.2011.5764567.

[28] Tran Thi Thu Ha, Jinsul Kim, and Jiseung Nam. "Design and Deployment of Low-Delay Hybrid CDN–P2P Architecture for Live Video Streaming Over the Web". en. In: *Wireless Personal Communications* 94.3 (June 2017), pp. 513–525. ISSN: 0929-6212, 1572-834X. DOI: 10.1007/s11277-015-3144-1. URL: https://link.springer.com/article/10.1007/s11277-015-3144-1 (visited on 06/21/2017).

[29] Chris Michaels. *HLS Latency Sucks, But Here's How to Fix It | Wowza*. Feb. 2017. URL: https://www.wowza.com/blog/hls-latency-sucks-but-heres-how-to-fix-it (visited on 06/21/2017).

[30] *WebRTC 1.0: Real-time Communication Between Browsers*. en. 2017. URL: https://www.w3.org/TR/webrtc/ (visited on 06/21/2017).

[31] 2011. URL: http://www.videolan.org/.

[32] *Network Latency and Packet Loss Emulation @ Calomel.org*. URL: https://calomel.org/network_loss_emulation.html (visited on 07/28/2017).

[33] *IP Latency Statistics*. May 2017. URL: http://www.verizonenterprise.com/about/network/latency/ (visited on 06/29/2017).

Chapter 6: A QoE Study on the Impact of Flexible Dual-TCP/UDP Streaming Protocol on Wireless Video Streaming

[1] *VNI Global Fixed and Mobile Internet Traffic Forecasts*. URL: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html (visited on 06/18/2017).

[2] *4K Internet TV & Video to be Viewed by 1 in 10 US Residents*. Aug. 2016. URL: https://www.juniperresearch.com/press/press-releases/4k-internet-tv-video-content-to-be-viewed-by-1-i (visited on 06/19/2017).

[3] Junchen Jiang, Vyas Sekar, and Hui Zhang. "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive". In: *IEEE/ACM Trans. Netw.* 22.1 (Feb. 2014), pp. 326–340. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2291681. URL: http://dx.doi.org/10.1109/TNET.2013.2291681.

[4] S. S. Krishnan and R. K. Sitaraman. "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs". In: *IEEE/ACM Transactions on Networking* 21.6 (Dec. 2013), pp. 2001–2014. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2281542.

[5] *What YOU Need to Know About HLS: Pros and Cons*. Jan. 2016. URL: http://blog.red5pro.com/what-you-need-to-know-about-hls-pros-and-cons/ (visited on 06/19/2017).

[6] Jing Zhao et al. "Flexible Dual TCP/UDP Streaming for H.264 HD Video over WLANs". In: *Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC 2013)*. Kota Kinabalu, Malaysia, 2013, 34:1–34:9. ISBN: 978-1-4503-1958-4. DOI: 10.1145/2448556.2448590.

[7] Mohammed Sinky et al. "Analysis of H.264 Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video Over WLANs". In: *IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*. Las Vegas, USA, Jan. 2015, pp. 576–581.

[8] Arul Dhamodaran, Mohammed Sinky, and Ben Lee. "Adaptive Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video". In: *The Tenth International Conference on Systems and Networks Communications (ICSNC 2015)*. Barcelona, Spain, Nov. 2015, pp. 35–40.

[9] K. Gatimu et al. "Experimental study of low-latency HD VoD streaming using flexible dual TCP-UDP streaming protocol". In: *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. Jan. 2018, pp. 1–6. DOI: 10.1109/CCNC.2018.8319234.

[10] Te-Yuan Huang et al. “A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service”. In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM ’14. Chicago, Illinois, USA: ACM, 2014, pp. 187–198. ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626296. URL: http://doi.acm.org/10.1145/2619239.2626296.

[11] *ISO/IEC TR 23009-3:2015 - Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 3: Implementation guidelines*. URL: https://www.iso.org/standard/63562.html (visited on 04/03/2018).

[12] Alex Zambelli. *Smooth Streaming Technical Overview*. URL: http://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview.

[13] Adobe Systems. *HTTP Dynamic Streaming*. URL: http://www.adobe.com/products/hds-dynamic-streaming.html.

[14] Apple Inc. *HTTP Live Streaming Internet—Draft*. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[15] *ISO/IEC 23009-1:2012 - Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623 (visited on 06/22/2016).

[16] J. Kua, G. Armitage, and P. Branch. “A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP”. In: *IEEE Communications Surveys Tutorials* 19.3 (thirdquarter 2017), pp. 1842–1866. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2685630.

[17] Te-Yuan Huang et al. “Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard”. In: *Proceedings of the 2012 Internet Measurement Conference*. IMC ’12. Boston, Massachusetts, USA: ACM, 2012, pp. 225–238. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398800. URL: http://doi.acm.org/10.1145/2398776.2398800.

[18] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. “BOLA: Near-optimal bitrate adaptation for online videos”. In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524428.

[19] Xiaoqi Yin et al. “A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP”. In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 325–338. ISSN: 0146-4833. DOI: 10.1145/2829988.2787486. URL: http://doi.acm.org/10.1145/2829988.2787486.

[20] Lucian Popa, Ali Ghodsi, and Ion Stoica. "HTTP As the Narrow Waist of the Future Internet". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 6:1–6:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868453. URL: http://doi.acm.org/10.1145/1868447.1868453.

[21] *Peer5*. en. 2018. URL: https://www.peer5.com/ (visited on 04/25/2018).

[22] Sam Dutton. *WebRTC in the real world: STUN, TURN and signaling - HTML5 Rocks*. URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/ (visited on 06/21/2017).

[23] Bing Wang et al. "Multimedia Streaming via TCP: An Analytic Performance Study". In: *ACM Trans. Multimedia Comput. Commun. Appl.* 4.2 (May 2008), 16:1–16:22. ISSN: 1551-6857. DOI: 10.1145/1352012.1352020. URL: http://doi.acm.org/10.1145/1352012.1352020.

[24] A. Aggarwal, S. Savage, and T. Anderson. "Understanding the performance of TCP pacing". In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 3. Mar. 2000, 1157–1165 vol.3. DOI: 10.1109/INFCOM.2000.832483.

[25] Jacob Chakareski et al. "System and method for low delay, interactive communication using multiple TCP connections and scalable coding". US8699522 B2. U.S. Classification 370/474, 370/536, 375/240.05, 709/231; International Classification H04J3/24; Cooperative Classification H04L65/607, H04L47/32, H04L47/10, H04L47/193, H04L47/2416, H04L65/4015, H04L47/283, H04L65/80. Apr. 2014. URL: http://www.google.com/patents/US8699522 (visited on 06/20/2017).

[26] V. Swaminathan and S. Wei. "Low latency live video streaming using HTTP chunked encoding". In: *2011 IEEE 13th International Workshop on Multimedia Signal Processing*. Oct. 2011, pp. 1–6. DOI: 10.1109/MMSP.2011.6093825.

[27] P. Houzé et al. "Applicative-layer multipath for low-latency adaptive live streaming". In: *2016 IEEE International Conference on Communications (ICC)*. May 2016, pp. 1–7. DOI: 10.1109/ICC.2016.7511550.

[28] Sheng Wei and Viswanathan Swaminathan. "Low Latency Live Video Streaming over HTTP 2.0". In: *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. NOSSDAV '14. New York, NY, USA: ACM, 2014, 37:37–37:42. ISBN: 978-1-4503-2706-0. DOI: 10.1145/2578260.2578277. URL: http://doi.acm.org/10.1145/2578260.2578277 (visited on 06/20/2017).

[29] Wael Cherif et al. "DASH Fast Start Using HTTP/2". In: *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV '15. New York, NY, USA: ACM, 2015, pp. 25–30. ISBN: 978-1-4503-3352-8. DOI: 10.1145/2736084.2736088. URL: http://doi.acm.org/10.1145/2736084.2736088 (visited on 06/21/2017).

[30] Rafael Huysegems et al. "HTTP/2-Based Methods to Improve the Live Experience of Adaptive Streaming". In: *Proceedings of the 23rd ACM International Conference on Multimedia*. MM '15. New York, NY, USA: ACM, 2015, pp. 541–550. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806264. URL: http://doi.acm.org/10.1145/2733373.2806264 (visited on 06/21/2017).

[31] Alex Theedom. *Tracking HTTP/2 Adoption: Stagnation - DZone Web Dev*. 2016. URL: https://dzone.com/articles/tracking-http2-adoption-stagnation (visited on 06/21/2017).

[32] X. Liu, H. Yin, and C. Lin. "A Novel and High-Quality Measurement Study of Commercial CDN-P2P Live Streaming". In: *2009 WRI International Conference on Communications and Mobile Computing*. Vol. 3. Jan. 2009, pp. 325–329. DOI: 10.1109/CMC.2009.152.

[33] ZhiHui Lu, Ye Wang, and Yang Richard Yang. "An Analysis and Comparison of CDN-P2P-hybrid Content Delivery System and Model". In: *Journal of Communications* 7.3 (Mar. 2012). ISSN: 1796-2021. DOI: 10.4304/jcm.7.3.232-245. URL: http://www.jocm.us/index.php?m=content&c=index&a=show&catid=39&id=90 (visited on 06/21/2017).

[34] Dongyan Xu et al. "Analysis of a CDN-P2P hybrid architecture for cost-effective streaming media distribution". en. In: *Multimedia Systems* 11.4 (Apr. 2006), pp. 383–399. ISSN: 0942-4962, 1432-1882. DOI: 10.1007/s00530-006-0015-3. URL: https://link.springer.com/article/10.1007/s00530-006-0015-3 (visited on 06/21/2017).

[35] S. M. Y. Seyyedi and B. Akbari. "Hybrid CDN-P2P architectures for live video streaming: Comparative study of connected and unconnected meshes". In: *2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*. Feb. 2011, pp. 175–180. DOI: 10.1109/CNDS.2011.5764567.

[36] Tran Thi Thu Ha, Jinsul Kim, and Jiseung Nam. "Design and Deployment of Low-Delay Hybrid CDN–P2P Architecture for Live Video Streaming Over the Web". en. In: *Wireless Personal Communications* 94.3 (June 2017), pp. 513–525. ISSN: 0929-6212, 1572-834X. DOI: 10.1007/s11277-015-3144-1. URL: https://link.springer.com/article/10.1007/s11277-015-3144-1 (visited on 06/21/2017).

[37] Chris Michaels. *HLS Latency Sucks, But Here's How to Fix It | Wowza*. Feb. 2017. URL: https://www.wowza.com/blog/hls-latency-sucks-but-heres-how-to-fix-it (visited on 06/21/2017).

[38] Jack Sax Tom Velten Robert Hinden. *Reliable Data Protocol*. July 1984. URL: https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00 (visited on 02/24/2017).

[39] Tom Bova and Ted Krivoruchka. *Reliable UDP Protocol*. URL: https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00 (visited on 02/24/2017).

[40] John Williams. *Microsoft TV Test*. May 2011. URL: https://www.viavisolutions.com/en-us/literature/microsoft-tv-test-application-notes-en.pdf.

[41] S. Floyd, M. Handley, and E. Kohler. *Datagram Congestion Control Protocol (DCCP)*. URL: https://tools.ietf.org/html/rfc4340 (visited on 02/24/2017).

[42] Alyssa Wilk et al. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2*. URL: https://tools.ietf.org/html/draft-hamilton-early-deployment-quic-00 (visited on 02/24/2017).

[43] Christian Timmerer and Alan Bertoni. "Advanced Transport Options for the Dynamic Adaptive Streaming over HTTP". In: *arXiv preprint arXiv:1606.00264* (2016). URL: https://arxiv.org/abs/1606.00264 (visited on 06/19/2017).

[44] 2011. URL: http://www.videolan.org/.

[45] *Network Latency and Packet Loss Emulation @ Calomel.org*. URL: https://calomel.org/network_loss_emulation.html (visited on 07/28/2017).

[46] *IP Latency Statistics*. May 2017. URL: http://www.verizonenterprise.com/about/network/latency/ (visited on 06/29/2017).

[47] DASH Industry Forum. *Guidelines for Implementation: DASH-AVC/264 Test cases and Vectors*. Mar. 2014. URL: https://dashif.org/wp-content/uploads/2016/06/DASH-AVC-264-Test-Vectors-v1.0.pdf (visited on 2014).

Chapter 7: qMDP: DASH Adaptation using Queueing Theory within a Markov Decision Process

[1] Zenith Media. *Online video viewing to exceed an hour a day in 2018*.

[2] *VNI Global Fixed and Mobile Internet Traffic Forecasts*. URL: http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html (visited on 06/18/2017).

[3] Alex Zambelli. *Smooth Streaming Technical Overview*. URL: http://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview.

[4] Adobe Systems. *HTTP Dynamic Streaming*. URL: http://www.adobe.com/products/hds-dynamic-streaming.html.

[5] Apple Inc. *HTTP Live Streaming Internet—Draft*. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[6] *ISO/IEC 23009-1:2012 - Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623 (visited on 06/22/2016).

[7] S. S. Krishnan and R. K. Sitaraman. "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs". In: *IEEE/ACM Transactions on Networking* 21.6 (Dec. 2013), pp. 2001–2014. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2281542.

[8] J. Kua, G. Armitage, and P. Branch. "A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP". In: *IEEE Communications Surveys Tutorials* 19.3 (thirdquarter 2017), pp. 1842–1866. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2685630.

[9] Te-Yuan Huang et al. "Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard". In: *Proceedings of the 2012 Internet Measurement Conference*. IMC '12. Boston, Massachusetts, USA: ACM, 2012, pp. 225–238. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398800. URL: http://doi.acm.org/10.1145/2398776.2398800.

[10] Te-Yuan Huang et al. "A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service". In: *Proceedings of the 2014 ACM Conference on SIGCOMM*. SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, pp. 187–198. ISBN: 978-1-4503-2836-4. DOI: 10.1145/2619239.2626296. URL: http://doi.acm.org/10.1145/2619239.2626296.

[11] Xiaoqi Yin et al. "A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP". In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 325–338. ISSN: 0146-4833. DOI: 10.1145/2829988.2787486. URL: http://doi.acm.org/10.1145/2829988.2787486.

[12] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. "BOLA: Near-optimal bitrate adaptation for online videos". In: *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. Apr. 2016, pp. 1–9. DOI: 10.1109/INFOCOM.2016.7524428.

[13] Praveen Kumar Yadav, Arash Shafiei, and Wei Tsang Ooi. "QUETRA: A Queuing Theory Approach to DASH Rate Adaptation". In: *Proceedings of the 2017 ACM on Multimedia Conference*. MM '17. Mountain View, California, USA: ACM, 2017, pp. 1130–1138. ISBN: 978-1-4503-4906-2. DOI: 10.1145/3123266.3123390. URL: http://doi.acm.org/10.1145/3123266.3123390.

[14] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[15] Federico Chiariotti et al. "Online Learning Adaptation Strategy for DASH Clients". In: *Proceedings of the 7th International Conference on Multimedia Systems*. MMSys '16. Klagenfurt, Austria: ACM, 2016, 8:1–8:12. ISBN: 978-1-4503-4297-1. DOI: 10.1145/2910017.2910603. URL: http://doi.acm.org/10.1145/2910017.2910603.

[16] Te-Yuan Huang, Ramesh Johari, and Nick McKeown. "Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming". In: *Proceedings of the 2013 ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*. FhMN '13. Hong Kong, China: ACM, 2013, pp. 9–14. ISBN: 978-1-4503-2183-9. DOI: 10.1145/2491172.2491179. URL: http://doi.acm.org/10.1145/2491172.2491179.

[17] Junchen Jiang, Vyas Sekar, and Hui Zhang. "Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive". In: *IEEE/ACM Trans. Netw.* 22.1 (Feb. 2014), pp. 326–340. ISSN: 1063-6692. DOI: 10.1109/TNET.2013.2291681. URL: http://dx.doi.org/10.1109/TNET.2013.2291681.

[18] Z. Li et al. "Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale". In: *IEEE Journal on Selected Areas in Communications* 32.4 (Apr. 2014), pp. 719–733. ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.140405.

[19] Maxim Claeys et al. "Design of a Q-learning-based client quality selection algorithm for HTTP adaptive video streaming". In: *Proceedings of the 2013 Workshop on Adaptive and Learning Agents (ALA), Saint Paul (Minn.), USA*. 2013, pp. 30–37.

[20] Maxim Claeys et al. "Design and optimisation of a (FA) Q-learning-based HTTP adaptive streaming client". In: *Connection Science* 26.1 (2014), pp. 25–43.

[21] C. Zhou, C. Lin, and Z. Guo. "mDASH: A Markov Decision-Based Rate Adaptation Approach for Dynamic HTTP Streaming". In: *IEEE Transactions on Multimedia* 18.4 (Apr. 2016), pp. 738–751. ISSN: 1520-9210. DOI: 10.1109/TMM.2016.2522650.

[22] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. "Neural Adaptive Video Streaming with Pensieve". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. Los Angeles, CA, USA: ACM, 2017, pp. 197–210. ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098843. URL: http://doi.acm.org/10.1145/3098822.3098843.

[23] A. K. Erlang. "The theory of probabilities and telephone conversations". In: *Nyt Tidsskrift for Matematik B.* 20 (1909), pp. 33–39.

[24] Olivier Brun and Jean-Marie Garcia. "Analytical Solution of Finite Capacity M/D/1 Queues". In: *Journal of Applied Probability* 37.4 (2000), pp. 1092–1098. ISSN: 00219002. URL: http://www.jstor.org/stable/3215497.

[25] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[26] Christopher Watkins. "Learning From Delayed Rewards". In: (Jan. 1989).

[27] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: https://www.nature.com/articles/nature14236.

[28] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1928–1937. URL: http://dl.acm.org/citation.cfm?id=3045390.3045594.

[29] *Home · Dash-Industry-Forum/dash.js Wiki · GitHub*. URL: https://github.com/Dash-Industry-Forum/dash.js/wiki (visited on 02/28/2017).

[30] J. van der Hooft et al. "HTTP/2-Based Adaptive Streaming of HEVC Video Over 4G/LTE Networks". In: *IEEE Communications Letters* 20.11 (Nov. 2016), pp. 2177–2180. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2016.2601087.

[31] Martin Abadi et al. "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026899.

[32] Lajos Takács. *Introduction to the Theory of Queues*. Oxford University Press, 1962.

[33] J.W. Cohen. *The Single Server Queue*. North Holland, Amsterdam, 1969.

Appendix A: Analysis of Optimum Substream Lengths for Dual TCP/UDP Streaming of HD H.264 Video Over Congested WLANs

[1] Changwoo Yoon, Taiwon Um, and Hyunwoo Lee. "Classification of N-Screen Services and its standardization". In: *2012 14th International Conference on Advanced Communication Technology (ICACT)*. Feb. 2012, pp. 597–602.

[2] Jing Zhao et al. "Flexible Dual TCP/UDP Streaming for H.264 HD Video over WLANs". In: *Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC 2013)*. Kota Kinabalu, Malaysia, 2013, 34:1–34:9. ISBN: 978-1-4503-1958-4. DOI: 10.1145/2448556.2448590.

[3] Mohammed Sinky et al. "Analysis of H.264 Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video Over WLANs". In: *IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*. Las Vegas, USA, Jan. 2015, pp. 576–581.

[4] Arul Dhamodaran, Mohammed Sinky, and Ben Lee. "Adaptive Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video". In: *The Tenth International Conference on Systems and Networks Communications (ICSNC 2015)*. Barcelona, Spain, Nov. 2015, pp. 35–40.

[5] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. "Rate Adaptation for Adaptive HTTP Streaming". In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. MMSys '11. New York, NY, USA: ACM, 2011, pp. 169–174. ISBN: 978-1-4503-0518-1. DOI: 10.1145/1943552.1943575. URL: http://doi.acm.org/10.1145/1943552.1943575 (visited on 06/25/2016).

[6] M. Seufert et al. "A Survey on Quality of Experience of HTTP Adaptive Streaming". In: *IEEE Communications Surveys Tutorials* 17.1 (2015), pp. 469–492. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2360940.

[7] Alex Zambelli. *Smooth Streaming Technical Overview*. URL: http://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview.

[8] Apple Inc. *HTTP Live Streaming Internet—Draft*. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-19.

[9] Adobe Systems. *HTTP Dynamic Streaming*. URL: http://www.adobe.com/products/hds-dynamic-streaming.html.

[10] K. Ramkishor et al. "Adaptation of video encoders for improvement in quality". In: *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03*. Vol. 2. May 2003, II–692–II–695 vol.2. DOI: 10.1109/ISCAS.2003.1206068.

[11] *ISO/IEC 23009-1:2012 - Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. URL: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623 (visited on 06/22/2016).

[12] Thomas Schierl et al. "Priority-based Media Delivery using SVC with RTP and HTTP streaming". en. In: *Multimedia Tools and Applications* 55.2 (Sept. 2010), pp. 227–246. ISSN: 1380-7501, 1573-7721. DOI: 10.1007/s11042-010-0572-5. URL: http://link.springer.com.ezproxy.proxy.library.oregonstate.edu/article/10.1007/s11042-010-0572-5 (visited on 07/02/2016).

[13] Y. K. Wang et al. *RTP Payload Format for H.264 Video*. RFC 6184 (Proposed Standard). Internet Engineering Task Force, May 2011. URL: http://www.ietf.org/rfc/rfc6184.txt.

[14] Chunho Lee et al. "OEFMON: An open evaluation framework for multimedia over networks". In: *IEEE Communications Magazine* 49.9 (Sept. 2011), pp. 153–161.

[15] *QualNet 7.3 User's Guide*. Scalable Network Technologies, Inc. 2016.

[16] S. Wei, G. Bai, and H. Shen. "An Adaptive Queue Management Mechanism for Video Streaming over Mobile Ad Hoc Networks". In: *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*. Sept. 2009, pp. 1–4. DOI: 10.1109/WICOM.2009.5301909.

[17] A. Divakaran, R. Radhakrishnan, and K. A. Peker. "Motion activity-based extraction of key-frames from video shots". In: *2002 International Conference on Image Processing. 2002. Proceedings*. Vol. 1. 2002, I–932–I–935 vol.1. DOI: 10.1109/ICIP.2002.1038180.

[18] T. Hossfeld et al. "Initial delay vs. interruptions: Between the devil and the deep blue sea". In: *2012 Fourth International Workshop on Quality of Multimedia Experience (QoMEX)*. July 2012, pp. 1–6. DOI: 10.1109/QoMEX.2012.6263849.

Appendix B:Adaptive Queue Management Scheme for Flexible Dual TCP/UDP Streaming Protocol

[1] Changwoo Yoon, Taiwon Um, and Hyunwoo Lee. "Classification of N-Screen Services and its standardization". In: *14th International Conference on Advanced Communication Technology (ICACT)*. 2012, pp. 597–602.

[2] Jing Zhao et al. "Flexible Dual TCP/UDP Streaming for H.264 HD Video over WLANs". In: *Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC '13)*. Kota Kinabalu, Malaysia: ACM, 2013, 34:1–34:9. ISBN: 978-1-4503-1958-4. DOI: 10.1145/2448556.2448590.

[3] Mohammed Sinky et al. "Analysis of H.264 Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video Over WLANs". In: *2015 IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*. Las Vegas, USA, Jan. 2015, pp. 576–581.

[4] Arul Dhamodaran, Mohammed Sinky, and Ben Lee. "Adaptive Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video". In: *The Tenth International Conference on Systems and Networks Communications (ICSNC 2015)*. Barcelona, Spain, Nov. 2015, pp. 35–40.

[5] S. Floyd and V. Jacobson. "Random early detection gateways for congestion avoidance". In: *IEEE/ACM Transactions on Networking* 1.4 (Aug. 1993), pp. 397–413. ISSN: 1063-6692. DOI: 10.1109/90.251892.

[6] Wu-Chang Feng et al. "The BLUE active queue management algorithms". In: *IEEE/ACM Transactions on Networking* 10.4 (Aug. 2002), pp. 513–528. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.801399.

[7] K. Xu et al. "Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED". In: *Proceedings of MOBICOM*. San Diego, CA USA, Sept. 2003, pp. 16–28.

[8] Jian Chen and V. C. M. Leung. "Applying active queue management to link layer buffers for real-time traffic over third generation wireless networks". In: *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*. Vol. 3. Mar. 2003, 1657–1662 vol.3. DOI: 10.1109/WCNC.2003.1200635.

[9] Mei-Ling Shy, Shu-Ching Chen, and C. Ranasingha. "Router active queue management for both multimedia and best-effort traffic flows". In: *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*. Vol. 1. June 2004, 451–454 Vol.1. DOI: 10.1109/ICME.2004.1394226.

[10] Y. K. Wang et al. *RTP Payload Format for H.264 Video*. RFC 6184 (Proposed Standard). Internet Engineering Task Force, May 2011. URL: http://www.ietf.org/rfc/rfc6184.txt.

[11] Roger Pantos and William May. *HTTP Live Streaming*. IETF Draft, URL: https://developer.apple.com/streaming/ [Accessed: 2017-03-03]. Apr. 2014.

[12] Chunho Lee et al. "OEFMON: An Open Evaluation Framework for Multimedia Over Networks". In: *IEEE Communications Magazine* (Sept. 2011), pp. 153–161.

[13] *QualNet 7.3 User's Guide*. Scalable Network Technologies, Inc. 2016.

[14] Mohammed Sinky et al. "Analysis of H.264 Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video Over WLANs". In: *IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*. Las Vegas, USA, Jan. 2015, pp. 576–581.