AN ABSTRACT OF THE THESIS OF

Jing Zhao for the degree of Master of Science in

Electrical and Computer Engineering presented on May 30, 2013

Title: Flexible Dual TCP/UDP Streaming for H.264 HD Video Over WLANs

Abstract approved: ______________________________

Ben Lee

High Definition video streaming over WLANs faces many challenges because video data requires not only data integrity but also frames have strict playout deadline. Traditional streaming methods that rely solely on either UDP or TCP have difficulties meeting both requirements because UDP incurs packet loss while TCP incurs delay. This thesis proposed a new streaming method called *Flexible Dual-TCP/UDP Streaming Protocol* (FDSP) that utilizes the benefit of both UDP and TCP. The FDSP takes advantage of the hierarchical structure of the H.264/AVC syntax and uses TCP to transmit important syntax elements of H.264/AVC video and UDP to transmit non-important elements. Moreover, if desired, FDSP is flexible enough to send any H.264 syntax element via TCP. The proposed FDSP is implemented and validated under different wireless network conditions. Both visual quality and delay results are compared against pure-UDP and pure-TCP streaming methods. Our results show that FDSP effectively achieves a balance between delay and visual quality, thus it has advantage over traditional pure-UDP and pure-TCP methods.

©Copyright by Jing Zhao

May 30, 2013

All Rights Reserved

Flexible Dual TCP/UDP Streaming for H.264 HD Video Over WLANs

by

Jing Zhao

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 30, 2013
Commencement June 2013

Master of Science thesis of Jing Zhao presented on May 30, 2013

APPROVED:

Major Professor, representing Electrical and Computer Engineering

Director of School of Electrical Engineering and Computer Science

Dean of the Graduate School

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

Jing Zhao, Author

ACKNOWLEDGEMENTS

First and foremost, I sincerely thank my advisor Dr. Ben Lee for his expert guidance during my study at Oregon State University. His strong academic background and insightful advices make my academic journey much smoother than I expect. I also appreciate his persistent encouragement, complete trust in me, and care about my personal life. It was my pleasure to work with him.

I would like to express my gratitude to my research teammates for their excellent support. Special thanks to Mohammed Sinky and Kevin Gatimu for their efforts in developing many tools that are used for my research.

I particularly thank my wife, Yi, who gave birth to our daughter Jinmo in Spring 2013. I appreciate her patience and loving support during the busy time in our lives.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

FLEXIBLE DUAL TCP/UDP STREAMING FOR H.264 HD VIDEO OVER WLANS

1. INTRODUCTION

High Definition (HD) video streaming over WLANs has become a viable and important technology as network bandwidth continues to improve and the use of smartphones, Mobile Internet Devices, and wireless display devices increases. Some notable HD wireless streaming technologies include Apple AirPlay® [2], Intel WiDi® [3], and Cavium WiVu® [4]. These technologies are deployed in an ad hoc mode, and the state-of-the-art video compression standard H.264 facilitates wireless video streaming by providing more efficient compression algorithm and thus less data needs to be transmitted through the network. Moreover, H.264 provides many error-resilience and network-friendly features, such as Data Partitioning (DP), Flexible Macroblock Ordering (FMO), and Network Adaption Layer (NAL) structure [5]. However, wireless HD video streaming still faces many challenges. This is because, unlike transmitting traditional data, video streaming requires not only data integrity but also frames have strict playout deadline in the presence of packet delay and loss. Both of these factors are also closely related to streaming protocols.

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the two fundamental Transport Layer protocols used to transmit video data through the network. TCP is a reliable protocol but delay and bandwidth consumption increase due to re-transmissions of lost packets, which further increase the likelihood of packet loss. For example, HTTP-based streaming video relies on TCP. Much work has been done to hide or reduce delay caused by TCP [6–8], but this remains a major problem for real-time video streaming. By contrast, UDP offers minimal delay but does not guarantee delivery of packets. These lost packets cause errors that propagate to subsequent frames.

Although considerable amount of research has been done on both TCP and UDP to improve video streaming in general, little attention has been paid to utilize the advantages of using both TCP and UDP for wireless video streaming. Recently, Porter and Peng proposed a Hybrid TCP/UDP streaming method, which relies on TCP to transmit higher priority data and UDP to transmit lower priority data [9]. However, they did not actually implement their method in a realistic network environment and instead used a tool to randomly remove data from an encoded video locally in order to simulate packet loss caused by UDP. This evaluation process lacks rigor and prevents them from providing meaningful results, such as visual quality and buffering time caused by using both TCP and UDP.

In contrast to previous research that rely on either UDP or TCP, this thesis presents a new video streaming method called *Flexible Dual-TCP/UDP Streaming Protocol* (FDSP) that utilizes the benefits of combining TCP and UDP. FDSP achieves a balance between visual quality and buffering time by sending important data via TCP and less important data via UDP. In order to validate our idea, FDSP was implemented and tested on a complete video streaming simulator using HD videos. Our results show that the proposed method has advantage over traditional pure-UDP and pure-TCP streaming methods. Although this thesis focuses on streaming real-time HD video in a wireless ad-hoc network environment, FDSP can also be applied to sub-HD videos and other types of networks.

The rest of the thesis is organized as follows: Chapter 2. presents a background on H.264 video compression standard, streaming protocols, and packetization methods. Chapter 3. discusses the related work. The proposed streaming method is presented in Chapter 4. Chapter 5. presents a technique called *Sub-stream Overlapping* that enhances basic FDSP method to minimize initial buffering for long videos. Chapter 6. introduces a new module called "Percentage Based Prioritization" to FDSP that further improves its performance. Finally, Chapter 7. concludes the thesis.

2. BACKGROUND

This chapter provides the background information necessary to understand the proposed FDSP, and the relationship between how H.264 video is encoded and streamed and the effect of packet delay and loss on its visual quality.

2.1. I, P, and B-Frame

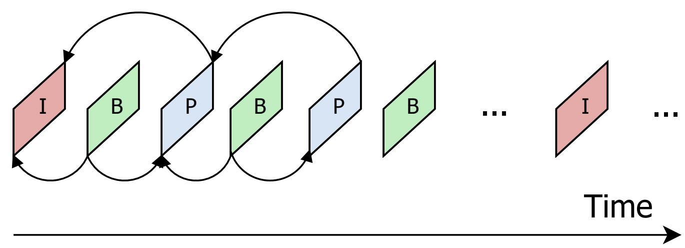


FIGURE 2.1: Typical GOP structure. Each arrow indicates the relationship between current (i.e., predicted) frame and reference frame(s).

An encoded video stream consists of a sequence of Group of Pictures (GOPs) as shown in Fig. 2.1. Each GOP consists of an intra-frame (I-frame), predicted-frames (P-frames), and bi-predicted frames (B-frames). An I-frame contains all the data required to reconstruct a complete frame and does not refer to other frames. Conversely, P- and B-frames require reference frame(s) during decoding. If a reference frame contains errors, these errors will propagate through subsequent frames that refer to this frame. Since an I-frame does not depend on any other frame, error propagation will cease when a new I-frame arrives. Consequently, I-frames should be given higher priority if possible during video streaming.

2.2. Features of H.264

H.264 is the state-of-the-art video compression standard. Compared to its predecessors, H.264 provides more aggressive compression ratio and has network-friendly features that make it more favorable for video streaming.

There are several characteristics of H.264, and video compression in general, that are important for efficient wireless video streaming. The two most important characteristics are the syntax for encoding video data to bitstream data and how the bitstream is packetized and transmitted. These issues are discussed below.

2.2.1 H.264 Bitstream Syntax and the Effect of Data Loss on Video

The hierarchical structure of the H.264 bitstream syntax is shown in Fig. 2.2. Network Adaption Layer (NAL) consists of a series of NAL units. Three common NAL units are Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and slice.

SPS contains parameters common to an entire video, such as profile and level the coded video conforms to, the size of a video frame and certain decoder constraints such as the maximum number of reference frames. Therefore, if SPS is lost, then the entire video cannot be decoded.

PPS contains common parameters that are applied to a sequence of frames, such as entropy coding type, number of active reference pictures and initialization parameters. If PPS for a sequence of frames is lost, then these frames cannot be decoded.

A slice is a unit for constructing a frame, and a frame can have either a single slice or multiple slices. A slice can be I-slice, P-slice, B-slice, or Instantaneous Decoder Refresh (IDR) slice. An IDR slice is a special form of I-slice that indicates that this slice cannot reference any slice before it, and is used to clear the contents of the reference frame buffer. Each slice contains a slice header and a slice data containing a number of macroblocks (MBs). Slice header contains information common to all the MBs within a slice. If a slice

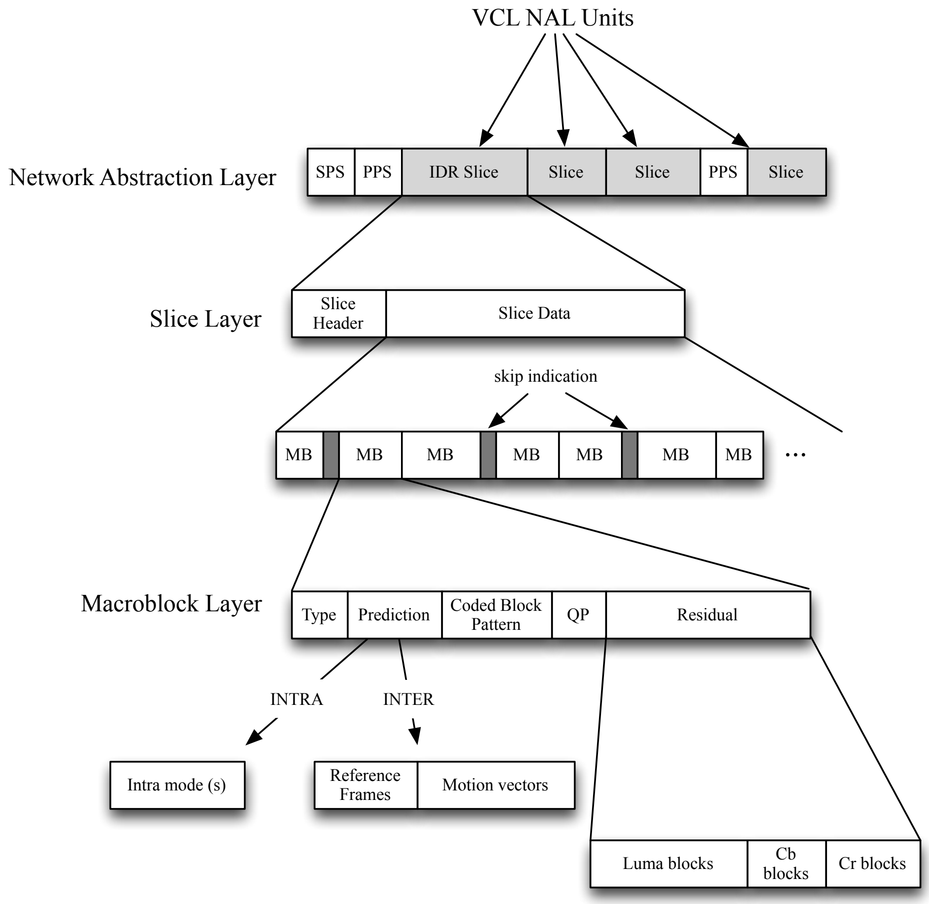
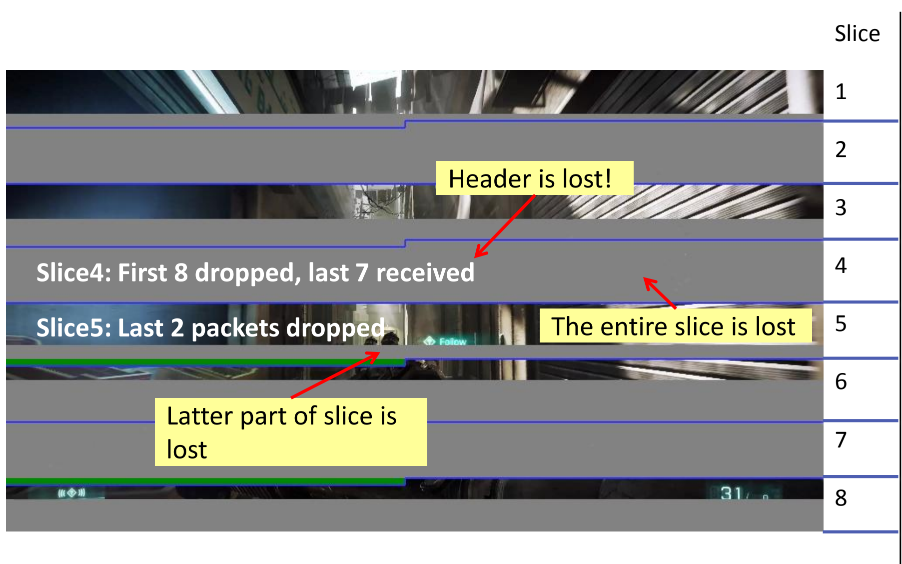


FIGURE 2.2: H.264 bitstream syntax [1].

header is lost, then the entire slice cannot be decoded even if the slice data is properly received [1, 5].

Fig. 2.3 illustrates the effect of packet loss on a frame from a HD video clip called "battlefield" streamed via UDP using VLC media player [10] and analyzed using Wireshark [11] and Elecard StreamEye Studio [12]. Fig. 2.3a shows the original transmitted frame, while Fig. 2.3b shows the received frame with some information missing due to packet loss. In this example, the slice header for Slice 4 is lost, thus the entire slice cannot be decoded. In contrast, the slice header for Slice 5 is received but the last two RTP packets are lost, which allowed most of the slice to be decoded. Afterwards, Error Concealment (EC) techniques can be used to recover the lost information with some artifacts (which is not shown). Therefore, PPS, SPSs, and slice headers are the most important data, and thus more care should be given to them during video streaming.

(a) The original frame.



(b) The received frame.

FIGURE 2.3: Effect of slice header loss.

2.3. Data Partitioning (DP)

DP is an error-resilience feature in H.264. The coded data for each slice is placed in three separate data partitions A, B, and C. Partition A contains the slice header and a header for each MB (i.e., MB type, quantization parameter, and motion vectors), Partition B contains Coded Block Patterns (CBPs) and coefficients for intra-coded MBs, and Partition C contains CBPs and coefficients for inter-coded MBs [5]. To decode Partition B, Partition A must be present. To decode Partition C, both Partition A and B must be present. DP can be used with Unequal Error Protection (UEP) methods to improve streaming performance. UEP will be discussed further in Chapter. 3. Although DP is a powerful tool for error resiliency, it has not yet been widely adopted because it requires videos to be re-encoded and 802.11e networks [13].

2.4. Streaming Protocols

Existing streaming protocols include Real Time Streaming Protocol (RTSP), Hyper-Text Transfer Protocol (HTTP), Microsoft Media Server (MMS), and Real-time Transport Protocol (RTP). Note that RTSP, HTTP, MMS, and RTP are Application Layer protocols so they do not deliver the streams themselves. For example, RTP uses UDP or TCP to deliver multimedia data. RTSP, HTTP, and MMS add more control features for streaming but they also use TCP or UDP to deliver multimedia data.

RTSP allows a client to remotely control a streaming media server. For example, a client can play, pause, and seek a video during streaming. RTSP can be used together with RTP Control Protocol (RTCP) to obtain statistical data on Quality of Service (QoS). Typically, RTSP uses TCP to deliver control signal and RTP/UDP to deliver multimedia data.

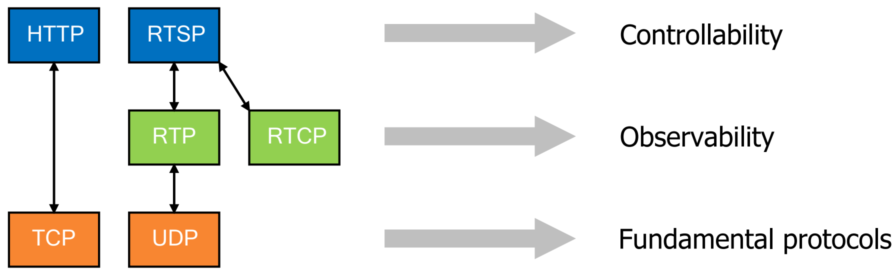


FIGURE 2.4: The relationship between streaming protocols.

HTTP also allows a client to control streaming, and uses TCP to transmit both multimedia and control data. Since HTTP uses TCP, packets are never lost. Another advantage of HTTP is that it works across firewalls as the HTTP port is usually turned on. However, HTTP will incur high end-to-end delay when lost packets need to be retransmitted.

RTP typically uses UDP to deliver multimedia data. An RTP header contains a sequence number and a timestamp. Sequence number is increased by one for each packet sent and is used for packet-loss detection. Timestamp can be used to synchronize multiple streams such as video and audio. Note that there is no control functionality by using only RTP/UDP.

The relationship between these streaming protocols is shown in Fig. 2.4. For our purpose, the focus is on RTP/UDP and RTP/TCP direct streaming as they are fundamental to all other streaming protocols.

2.5. Packetization Methods

Packetization method defines how a H.264 NAL unit is encapsulated into RTP packets. Different packetization methods can affect video streaming performance and thus visual quality. Two basic packetization methods are discussed below.

2.5.1 Method 1 (H264→TS→RTP)

Transport Stream (TS) is a legacy format designed to transport MPEG-2 video streams and is still used to carry H.264 video. This method is illustrated in Fig. 2.5. First, each TS packet is filled with H.264 bitstream as much as possible until the limit of 188 bytes is reached. Afterwards, each RTP packet is filled with as many TS packets as possible until the Maximum Transmission Unit (MTU) size is reached. Because the MTU size is 1500 bytes for Ethernet, there are typically seven TS packets in one RTP packet. Method 1 does not consider the H.264 NAL unit structure.

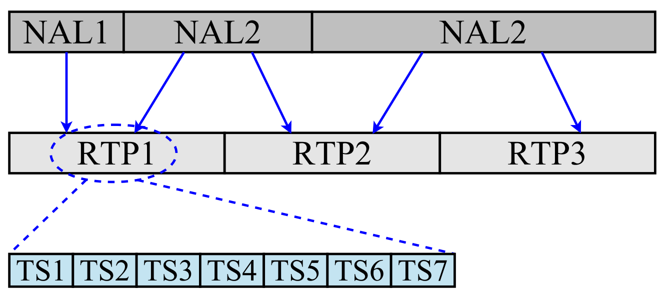


FIGURE 2.5: Method 1 H264→TS→RTP.

2.5.2 Method 2 (H264→RTP)

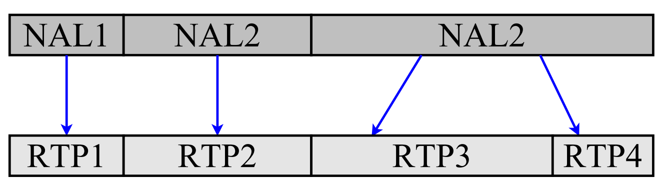


FIGURE 2.6: Method 2 H264→RTP.

This method shown in Fig. 2.6 is specifically designed for H.264 video. If the size of a NAL unit is less than or equal to the MTU size, one RTP packet contains only one NAL unit. If the size of a NAL unit is greater than the MTU size, the NAL unit will be fragmented into multiple RTP packets. This method also supports NAL unit aggregation, which is used when NAL units are very small. For example, since SPS and PPS have a few bytes at most, they can be aggregated with other NAL units into a single RTP packet to reduce the overhead for headers [5].

Method 2 has several advantages over Method 1 [14]. The most important one is that intelligently mapping NAL units to RTP packets provides better error resilience since the loss of one RTP packet affects only the NAL unit inside that packet (unless aggregation is used). In contrast, the size of RTP packets in Method 1 is fixed regardless of the size of NAL units. Therefore, multiple NAL units can be corrupted. In addition, Method 2 does not use TS packetization so there is less packet header overhead. For these reasons, Method 2 is used in the proposed FDSP method.

3. RELATED WORK

UDP is generally accepted to be more suitable than TCP for real-time video streaming since it offers low end-to-end delay for smooth video playout [5, 15]. Although UDP is prone to data loss, multimedia data to a certain degree (unlike traditional data) is loss-tolerant. In addition, a decoder uses EC techniques to reduce the artifacts caused by data loss. Numerous EC techniques have been developed to reduce the impact caused by packet loss [16, 17]. However, as discussed in Sec. 2.2.1, if lost packets contain important data, such as SPS, PPSs, and slice headers, the decoder simply cannot reconstruct the video even with the aid of EC.

In order to tolerate packet loss caused by UDP, UEP is often used in UDP-based streaming [5, 18, 19]. UEP aims to prioritize important data over the others because some syntax elements are more critical than others. A basic UEP method is to send important packets more than once, which raises the probability for the packets to arrive at the receiver [5]. More advanced UEP methods incorporate Forward Error Correction (FEC) [18, 19]. By using FEC to code important packets with redundancy, a receiver can recover these lost packets without retransmission. However, FEC introduces additional overhead, which increases network bandwidth required to transmit video.

Despite the conventional wisdom that TCP is not desirable for streaming, a significant fraction of commercial video streaming traffic uses it [20]. TCP provides guaranteed service so the transmitted packets are always preserved. Nevertheless, TCP's re-transmission and rate control mechanisms incur delay, which can cause packets to arrive after the playout deadline. A typical solution for this problem is to add a buffer in front of the video decoder. At the beginning of video streaming, the decoder waits until the buffer is filled before displaying video to accommodate initial throughput variability or inter-packet jitters. This waiting time is called *initial buffering*. After the decoder starts to decode video data in the buffer, decrease in throughput within a TCP session may cause buffer starvation. When this happens, the decoder stops displaying video until sufficient number of video packets are received. This waiting time is called *rebuffering* [6]. Buffering prevents late packets to be dropped; however, network congestion can cause long initial buffering and frequent rebuffering that degrades users' experience. Mok *et al.* performed a subjective assessment to measure Quality of Experience (QoE) of video streaming [21]. Their analysis showed that the frequency of rebuffering is the main factor responsible for variations in user experience. Much research has been done on determining the appropriate buffer size to reduce the frequency of rebuffering [6, 7]. Besides buffer size estimation, Brosh *et al.* presented packet splitting and parallel connection methods to reduce TCP delay [8].

Another approach to improve wireless video streaming is using IEEE 802.11e networks, which define a set of QoS enhancements through modifications to the Media Access Control (MAC) layer [13]. In an 802.11e network, delay-sensitive data such as video and audio can be assigned to higher priority class. If contention occurs at the MAC layer, smaller contention window size is used to transmit data with higher priority, and thus lower transmission delay can be achieved. 802.11e is specially tailored for multimedia, but it has not been widely adopted perhaps due to the hardware changes required.

To the best of our knowledge, the work closest to ours was done by Porter and Peng [9]. The authors proposed the Hybrid TCP/UDP method that prioritizes video data by using H.264 Data Partitioning (see Sec. 2.3.). However, our proposed FDSP method has a couple of advantages over their method.

First, FDSP is much more flexible because it is not tied to DP and is able to prioritize any syntax element. FDSP integrates a H.264 Syntax Parser within the streamer to segregate SPS, PPSs, and slice headers from rest of data. Thus, videos do not have to be re-encoded and the network does not have to support DP. Moreover, FDSP is able to segregate and prioritze any syntax element from a video stream. For example, some of the slice data (in addition to SPS, PPS and slice header) can be segregated and prioritized to further improve visual quality. In contrast, DP strictly defines contents of the three partitions and therefore the syntax elements that can be prioritized are fixed.

Second, the authors did not implement their proposed method as a complete system that includes a network simulator. For example, they used a modified *rtp_loss* utility (provided by *JM reference software*) to preserve partition A and randomly drop other partitions for a given video to simulate network behavior. In contrast, FDSP was implemented on a complete video streaming simulator (see Chapter 4.) that simulates realistic network scenarios and provides more insight on frame-by-frame behavior and buffering time, which are key indicators of video streaming QoE.

Finally, our simulation study is based on HD video, which is state-of-the-art and is also more demanding in terms of bandwidth and delay requirements.

4. FLEXIBLE DUAL TCP/UDP STREAMING PROTOCOL

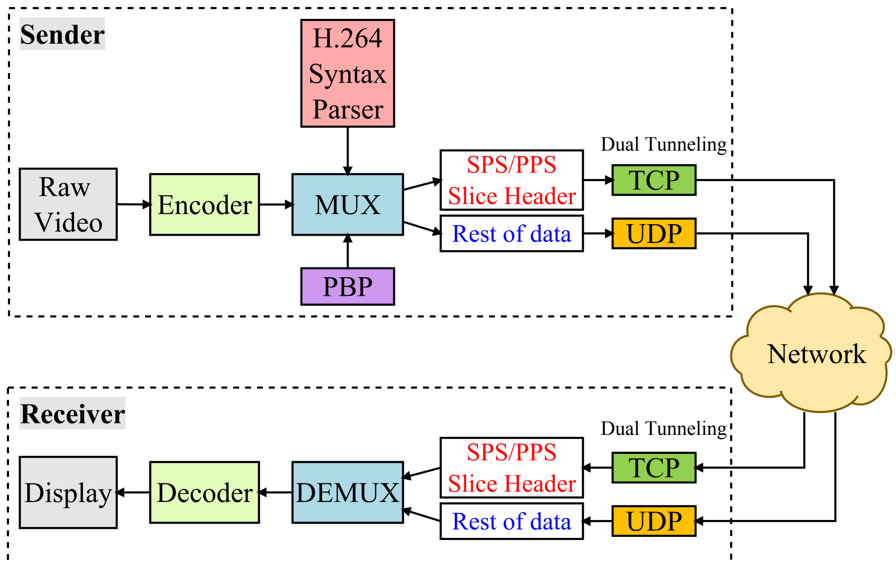


FIGURE 4.1: Flexible Dual-protocol Streaming system diagram.

The system diagram of the proposed FDSP is shown in Fig. 4.1.

The Sender consists of an encoder, MUX, Dual Tunneling (UDP+TCP), and the H.264 Syntax Parser. *MUX* together with the *H.264 Syntax Parser* are responsible for segregating critical data from the video bitstream and steering the two parts to UDP and TCP tunnels. *Dual Tunneling* keeps both UDP and TCP sessions active during video streaming. The operation of the Percentage Based Prioritization (PBP) will be discussed in Chapter 6.

The Receiver consists of Dual Tunneling, DEMUX and a decoder. *DEMUX* drops late UDP packets and merges on-time UDP packets with TCP packets. Afterwards, *DEMUX* sends on-time UDP packets together with TCP packets to the *Decoder*, which is implemented using FFmpeg [22].

The proposed FDSP was implemented within *Open Evaluation Framework for Mul-*

timedia Over Networks (OEFMON) [23], which integrates the DirectShow multimedia module [24] and the QualNet network simulator [25]. A simplified diagram of OEFMON is shown in Fig. 4.2. The main components used by FDSP are QualNet Connector, Video Source Filter, and Video Writer Filter. The QualNet Connector is responsible for RTP paketization. The Video Source Filter reads an H.264 file and sends the data to the QualNet Connector and the Video Writer Filter writes the decoded frame data to a raw video file. The detailed discussion of OEFMON can be found in [23].

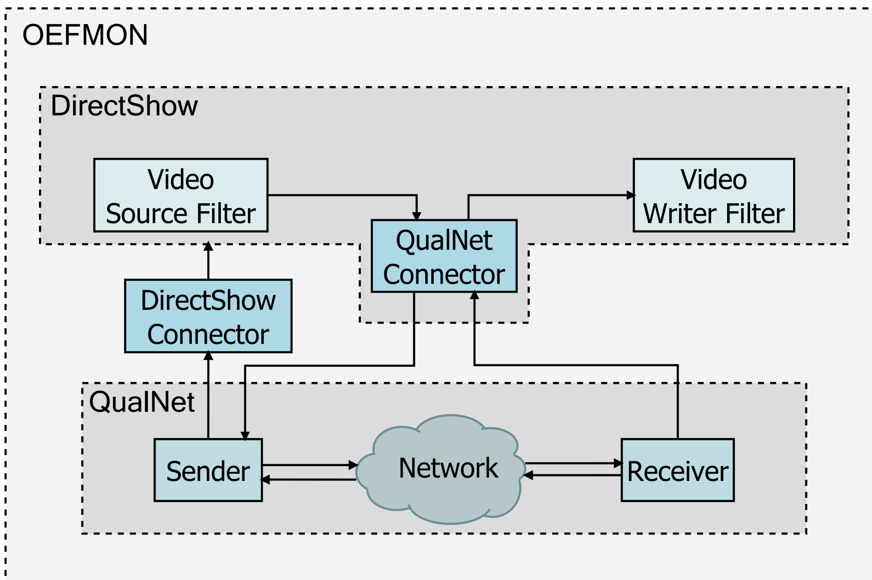


FIGURE 4.2: The general structure of OEFMON.

The following three sections (4.1.∼4.4.) discuss the implementation of the key components of FDSP.

4.1. Dual Tunneling (UDP+TCP)

OEFMON already implements UDP streaming in the QualNet network simulator. In order to implement Dual Tunneling, the existing code for UDP streaming required modification and a TCP streaming module needed to be implemented.

QualNet is a discrete-event simulator and an event is represented by a data structure called `MESSAGE`. The original code already contains `MESSAGE` for UDP and there is a pair of `MESSAGE`s (one for the sender and another for the receiver). The changes required for UDP mainly involved restructuring the code to handle the corresponding `MESSAGE`s. However, the implementation of TCP requires more `MESSAGE`s because it uses three-way handshaking. QualNet `API`s such as `APP_TcpOpenConnectionWithPriority` (request to open TCP socket) and `MESSAGE`s such as `MSG_APP_FromTransListenResult` (respond to request) must be properly handled before transmitting video data.

To enable Dual Tunneling, the functions for handling both UDP and TCP `MESSAGE`s were implemented inside a single application file called `app_fdspvideo.cpp` in QualNet.

4.2. TCP Segment Reassembly

TCP is a stream-oriented protocol (instead of being packet-oriented) and data is viewed as an unstructured, but ordered, stream of bytes [26]. Because TCP does not preserve data message boundaries, an RTP packet carried by TCP may be divided into several segments. Fig. 4.3 illustrates the fragmentation caused by TCP. In order to recover RTP packets from a TCP byte stream, TCP segments must be reassembled.

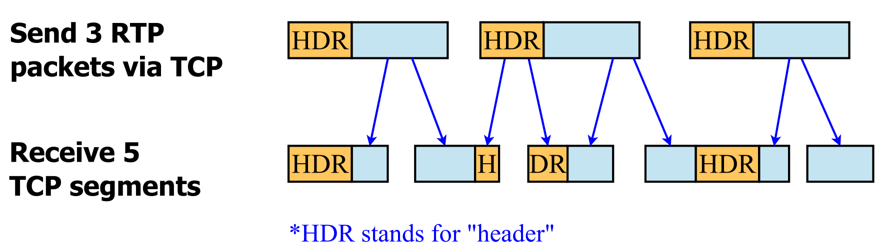


FIGURE 4.3: TCP Fragmentation.

The length information of a RTP packet is required to accomplish TCP segment reassembly. Because the standard RTP header does not include length information, the 12-byte RTP header is extended by two bytes to indicate the length of each RTP packet. This allows the receiver to determine whether or not the current TCP segment contains a complete RTP packet.

The algorithm for TCP reassembly involves first checking whether an RTP header is complete. If not, it waits for next TCP segment(s) to complete the RTP header. Once the RTP header is complete, it determines whether the RTP payload is complete using the RTP length information. If not, it waits for the next TCP segment(s) to complete the RTP payload.

4.3. H.264 Syntax Parser and MUX

The H.264 Syntax Parser was developed based on an open source library called *h264bitstream* [27]. The parser was implemented within QualNet and linked to `app_fdspvideo.cpp`. Before streaming, the parser parses the video bitstream and returns its syntax information (such as start address and length and type of each NAL unit) as input to MUX. During streaming, each NAL unit is encapsulated into an RTP packet by the QualNet Connector in OEFMON. At the same time, MUX uses the stored syntax information to determine whether an RTP packet contains a NAL unit that is SPS, PPS or slice header. If an RTP packet contains an important NAL unit, MUX will steer it to the TCP tunnel; otherwise, the packet will be steered to the UDP tunnel.

4.4. DEMUX

The DEMUX drops late UDP packets and merges on-time UDP packets with TCP packets.

When the receiver receives a TCP packet, DEMUX will store the packet in a file called “tcpdata.h264” on the disk drive. When the receiver receives a UDP packet, DEMUX will first parse the UDP packet to obtain the RTP timestamp. If the timestamp is larger than playout deadline, DEMUX will drop this UDP packet because it is late. If the timestamp is smaller than playout deadline, DEMUX will then parse the UDP packet to obtain the RTP sequence number. Afterwards, the DEMUX will parse the “tcpdata.h264” file to check whether there is any TCP packet whose RTP sequence number is smaller than the RTP sequence number of the UDP packet. If so, DEMUX will merge these TCP packets with the current UDP packet and send them to the decoder.

4.5. Simulation and Results

4.5.1 Simulation Setup

The primary video selected for our experiments is 360 frames of raw HD YUV video (12 seconds of 1920×1080 @30fps) from an “African Cats” trailer. The YUV file is encoded using x264 with an average bitrate of 4 Mbps and single slice per frame[1].

Using OEFMON, a 802.11g ad-hoc network with 18 Mbps bandwidth was setup and three network scenarios were created to evaluate video streaming performance. The placement of devices for three scenarios are shown in Fig. 4.4.

- Scenario 1: One pair of nodes stream the primary video over the network. At the same time, a second pair of nodes generates a 10 Mbps of constant bitrate (CBR) data as background traffic.
- Scenario 2: One more CBR data of 4 Mbps is added so that the network becomes saturated.

[1]The current version of OEFMON does not support multiple slices.

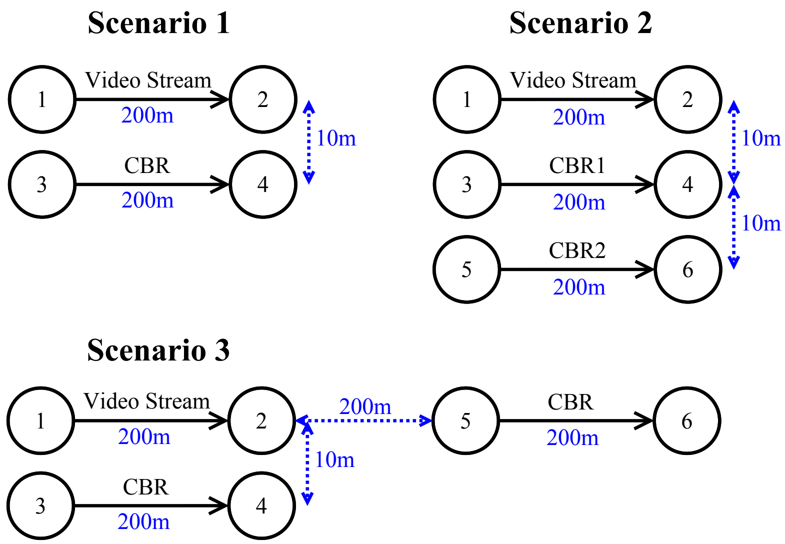


FIGURE 4.4: Simulated network scenarios.

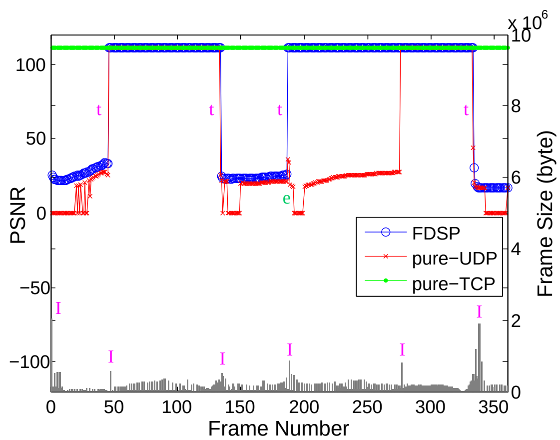
- Scenario 3: Repeats the network traffic of Scenario 1, but the nodes are now positioned in a classic hidden-node arrangement.

After video streaming completes, the sent and received video files are decoded using FFmpeg and PSNR is computed between the two YUV files using Avisynth [28]. Since PSNR calculation for missing frames and two identical frames are not well defined, this thesis uses 0 dB as the PNSR value for missing frames and follows the method used by Avisynth that uses 111 dB to indicate perfect PSNR. In addition to PSNR information, initial buffering and rebuffering are recorded to evaluate end-to-end delay.

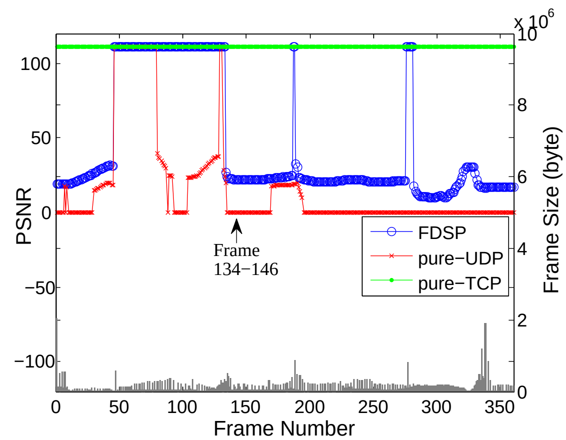
Our main objective for the experiments is to show the advantage of the proposed FDSP over traditional pure-UDP and pure-TCP streaming methods. For FDSP, all the important data (SPS, PPSs, and slice headers) will be first sent via TCP and then the rest of data will be sent via UDP. The time spent sending all the important data in FDSP is treated as initial buffering. For the pure-TCP method, a buffer is added to simulate initial buffering and rebuffering. In order to compare between FDSP and pure-TCP, the size of the buffer for pure-TCP is properly adjusted so that both methods have the same initial buffering time.

4.5.2 Results

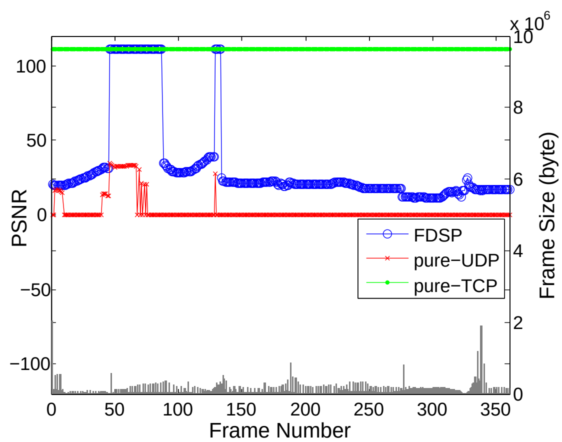
The PSNR comparison is shown in Fig. 4.5. The figures contain PSNR values as well as frame sizes (shown on the bottom of the graphs) for the three streaming methods. Note that x264 does not generate an uniform GOP by default, and therefore, the number frames between I-frames (marked as "I") can vary.



(a) PSNR comparison for Scenario 1.



(b) PSNR comparison for Scenario 2.



(c) PSNR comparison for Scenario 3.

FIGURE 4.5: PSNR comparison for all three scenarios.

For Scenario 1 shown in Fig. 4.5a, as expected, pure-UDP has the worst PSNR with an average of 54 dB, while FDSP achieves 79 dB. Pure-TCP has perfect PSNR of 111 dB because there is no packet loss.

The PSNR trends for FDSP and pure-UDP are in general similar. The transition points (marked as t) between non-perfect PSNR and perfect PSNR match fairly well with the position of I-frames (marked as "I"). This is because I-frames usually have much larger data size than P- or B-frames. If the network does not have enough bandwidth, a certain portion of the packets for an I-frame will most likely be lost or delayed, which reduces PSNR down from 111 dB. On the other hand, if an I-frame is received without any loss, it indicates that the network has enough bandwidth to transmit large frames. Therefore, the subsequent loss of P- or B-frames are unlikely under the same network condition since they are smaller than I-frames. Thus, PSNR remains at 111 dB until the next I-frame. An exception to this behavior starts at point e, which is caused by a packet loss. The packet-level log in OEFMON indicates the last packet for this I-frame is lost for the pure-UDP streaming, thus PSNR is not perfect. Due to error propagation, the subsequent frames also cannot achieve perfect PSNR until the next I-frame. Moreover, in contrast to FDSP, pure-UDP has many missing frames whose PSNR value is 0. The packet-level log indicates that those frames having PSNR value of 0 dB are caused by slice header loss. This again shows the importance of slice header and the advantage of FDSP.

For Scenario 2 shown in Fig. 4.5b, the network is saturated by introducing another CBR data of 4 Mbps. The pure-UDP method has a very low average PSNR value of 16 dB mainly because there are only 120 frames out of 360 that can be reconstructed by FFmpeg. In contrast, all 360 frames are reconstructed using FDSP, which results in much better PSNR value of 44 dB. FDSP experiences no missing frames because all the slice headers are properly received. For example, Fig. 4.5b shows that frames 134-146 are lost for pure-UDP, but these frames are not lost in FDSP. The packet-loss log in OEFMON

FIGURE 4.6: The decoded Frame 134 for FDSP.

shows that both pure-UDP and FDSP lose all the packets from the UDP tunnel for these frames. The only difference is that FDSP properly receives the slice headers for these frames from the TCP tunnel. As discussed before, the presence of slice headers is critical for a decoder to reconstruct a frame. Once a slice header is properly received, the decoder can use various EC techniques to conceal missing MBs even if rest of data is lost. For example, Fig. 4.6 shows the decoded Frame 134, which is a P-frame, reconstructed from a single packet containing the slice header and part of the slice. The upper-right watermark shows the frame number 134, which is the information retrieved from the packet. The lower-right watermark shows frame number 132, which indicates that FFmpeg using EC copied information from the previous frame 132 to the current frame 134.

For Scenario 3 shown in Fig. 4.5c, PSNR degrades further compared to Scenarios 1 and 2. This is caused by packet collisions resulting from the hidden-node effect. For pure-UDP, the average PSNR value is only 3 dB and there are 320 missing frames out of total of 360 frames. FDSP has an average PSNR value of 32 dB, which is significantly better than pure-UDP.

The buffering requirements are shown in Table 4.1. Both pure-TCP and FDSP have same initial buffering times, and these increase as the level of network congestion increases. Moreover, pure-TCP incurs very frequent rebuffering. During 12 seconds of video streaming, rebuffering occurs 3˜6 times and lasts on average 1.2˜2.1 seconds. As mentioned in Chapter 3., frequency of rebuffering is the main factor responsible for the variations in users' experience. Such a high frequency of rebuffering can be very annoying even though pure-TCP provides perfect visual quality. This is a clear advantage of FDSP over pure-TCP.

	Streaming Method	Initial Buffering (sec.)	Rebuffering Count	Avg. Rebuffering Time (sec.)
Scenario1	FDSP	1.1	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	1.1	3	1.2
Scenario2	FDSP	1.6	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	1.6	6	1.6
Scenario3	FDSP	1.9	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	1.9	6	2.1

TABLE 4.1: Buffering comparison for all three scenarios.

5. SUB-STREAM OVERLAPPING

5.1. Motivation

As mentioned earlier, FDSP will first send all the important data via TCP and then send the rest of data via UDP. For the 12-second test video, the initial buffering (i.e., the time spent sending all the important data) is less than 2 seconds. However, initial buffering will be unacceptably long when streaming an entire movie. Therefore, this chapter presents a technique called *Sub-stream Overlapping* to reduce the initial buffering requirement.

5.2. Solution

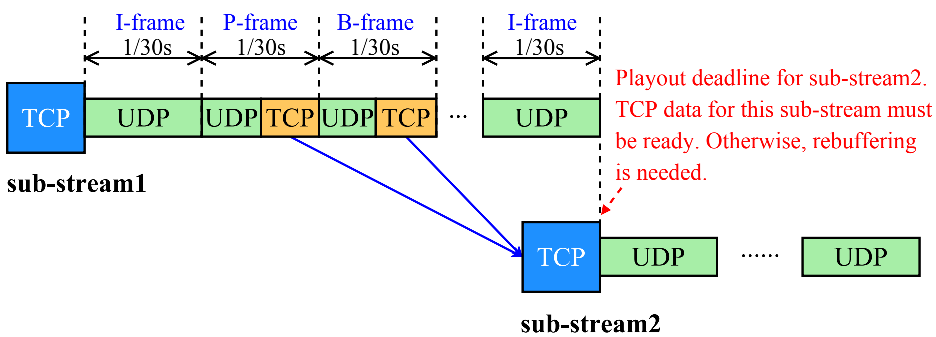


FIGURE 5.1: Sub-stream Overlapping.

Fig. 5.1 illustrates the basic idea of Sub-stream Overlapping. An entire movie is subdivided to several n-second sub-streams (Note that Fig. 5.1 only shows the first two sub-streams). FDSP will only send all the important data for the first n-second sub-stream via TCP and then start the normal UDP streaming. As long as FDSP is not sending any UDP packet, it will send important data for next n-second sub-stream(s) via TCP. FDSP will stop and perform rebuffering if the important data for an n-second sub-stream is not ready by the time it has to be played out.

The process for Sub-stream Overlapping can be divided into four steps.

- Step 1: Divide a video into several n-sec sub-streams.
- Step 2: Send important data for only the first n-sec sub-stream via TCP.
- Step 3: Start normal UDP streaming.
- Step 4: During UDP streaming, send important data for next n-second sub-stream(s) via TCP if network is relatively idle.

MUX, DEMUX and Dual Tunneling in Fig. 4.1 were modified to support Sub-stream Overlapping. For Step 4, the condition that determines whether network is relatively idle is done by monitoring the Network Layer queue within QualNet. If the number of packets in the queue is less than the threshold (marked as "thresh" in the figure), FDSP will send TCP packets for next sub-stream(s). Fig. 5.2 shows the queue status during video streaming using pure-UDP for Scenario 1 described in Sec. 4.5.1. The X-axis represents each frame that FDSP is sending. "Num. of Pkts in Queue" indicates the number of packets in the Network Layer queue when sending each frame. "Num. of UDP to Be Sent" indicates the number of UDP packets that comprise the current frame. These are packets that are about to be pushed onto queue and sent out. For example, for Frame 1, the queue should be empty, and therefore "Num. of Pkts in Queue" is 0. Frame 1 consists of 177 UDP packets and therefore "Num. of UDP to Be Sent" is 177.

5.2.1 Choice of the Threshold and Sub-stream Length

The choice of the threshold depends on network condition and sub-stream length. If a network is congested, "Num. of Pkts in Queue" will drop slowly. For example, "Num. of Pkts in Queue" might not drop to 20 until Frame 30 instead of Frame 18 shown in Fig. 5.2.

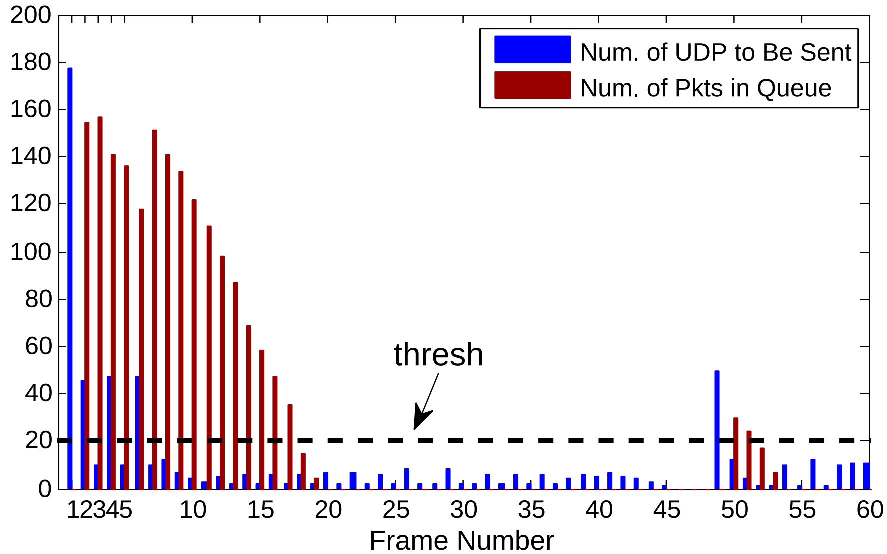


FIGURE 5.2: Queue status during streaming.

In order to allow TCP data for the next sub-stream to be ready before its playout deadline, the threshold needs to be increased to a value higher than 20 so that there is more time for FDSP to send TCP data. When the sub-stream length increases, TCP data required for the next sub-stream increases, but at the same time, the time available to transmit TCP data also increases. If network is not congested, there will be plenty of time available to send TCP data. In contrast, if the network is congested, "Num. of Pkts in Queue" can drop so slowly that, even if the time available to send TCP data increases, there may not be enough time to transmit TCP data. In this situation, the threshold should be increased.

Finding the optimum values for the threshold and the sub-stream length is a challenging issue, and thus left as future work. In this paper, 20 packets is chosen as the threshold and 10 seconds is chosen as the sub-stream length.

5.3. Simulation and Results

5.3.1 Simulation Setup

The simulation setup is very similar to the previous setup described in Sec. 4.5.1. The differences are listed as follows.

- A longer video clip is selected, which is a 40 second clip from the "African Cats" trailer consisting of 1,200 frames, in order to validate Sub-stream Overlapping.
- The bandwidth of the 802.11g ad-hoc network was increased to 54 Mbps and the distance between nodes were adjusted to better model a home environment. The placement of devices for three scenarios are shown in Fig. 5.3.
  - Scenario 1: One pair of nodes stream the primary video over the network. At the same time, two pair of nodes each generates a 20 Mbps of CBR data as background traffic.
  - Scenario 2: One more CBR data of 10 Mbps is added so that the network becomes saturated.
  - Scenario 3: Repeats the network traffic of Scenario 1, but the nodes are now positioned in a hidden-node arrangement.

5.3.2 Results

The PSNR comparison and packet loss for all three scenarios are shown in Fig. 5.4, which also includes frame size. Since 1200 frames are too many to fit in a graph, PSNR, packet loss, and frame size are all averaged over 1 second (which translates to 30 frames because fps for the video clip is 30.). In addition, PSNR and packet loss for the pure-TCP method are omitted because it always achieves PSNR of 111 dB and 0% packet loss.

For Scenario 1 shown in Fig. 5.4a, as expected PSNR for pure-UDP is worse than FDSP. Pure-UDP achieves an average of 93 dB, while FDSP achieves 102 dB. For Scenario

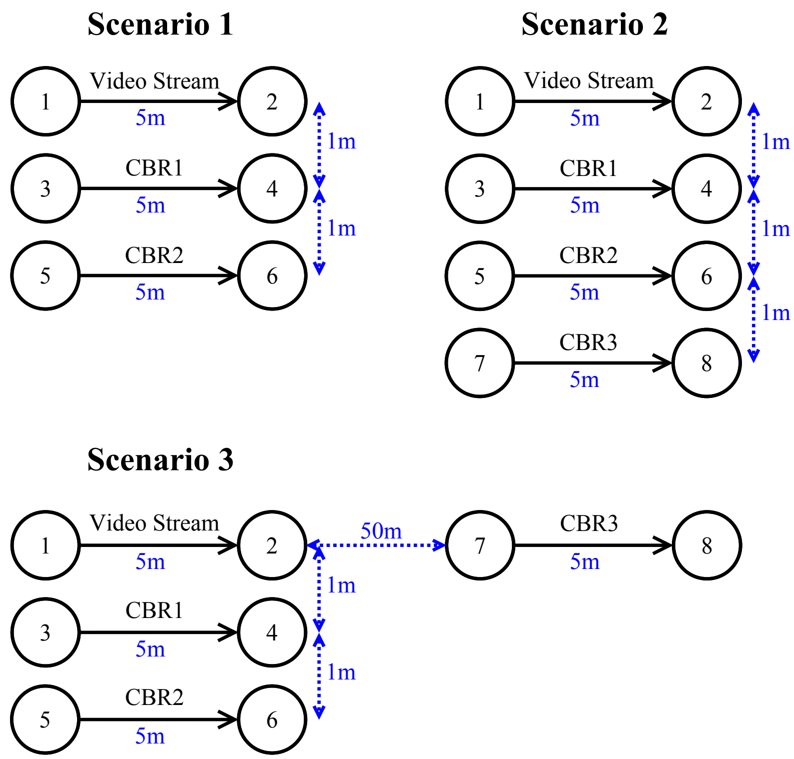
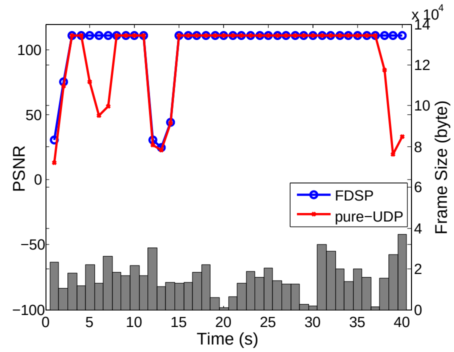


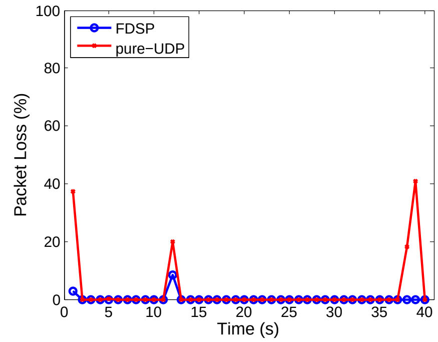
FIGURE 5.3: Simulated network scenarios (Sub-stream Overlapping).

2 shown in Fig. 5.4c, the pure-UDP method has an average PSNR value of 51 dB, which is much lower than the FDSP method with 83 dB. For Scenario 3 shown in Fig 5.4e, the node pair 7 and 8 causes a hidden node effect further degrading PSNR and delay compared to Scenarios 1 and 2. Pure-UDP achieves an average PSNR value of 52 dB. In contrast, FDSP has an average PSNR value of 76 dB, which is still better than pure-UDP.

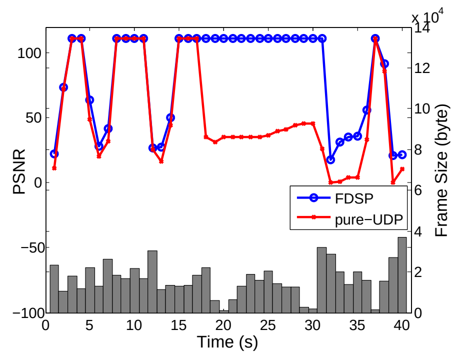
There is a direct correlation between packet loss and PSNR. In these graphs, each PSNR degradation is caused by some packet loss. For example, although it is hard to observe due to the whole-second averaging process, in Fig. 5.4b there is packet loss ratio of 0.2% at 5 second for pure-UDP, which reduces PSNR down from 111 dB to 76 dB. As can be seen in Fig. 5.4d and Fig. 5.4f, for packet loss higher than 85%, PSNR value is 0 dB indicating that frames are lost. In Scenario 1, there are 32 frames lost out of total of 1200 frames for pure-UDP. In contrast, slice headers for all 1200 frames are received using



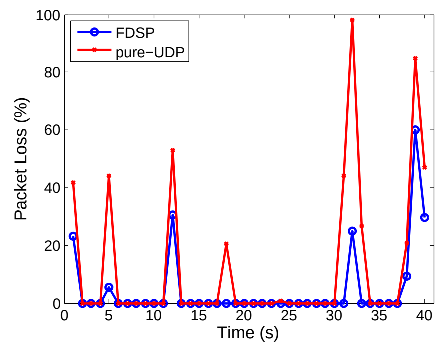
(a) PSNR comparison for Scenario 1.



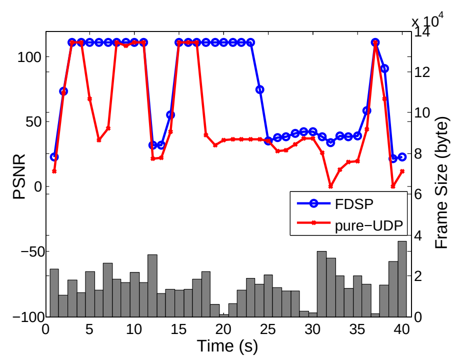
(b) Packet loss for Scenario 1.



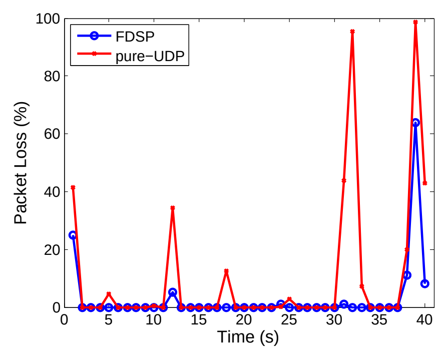
(c) PSNR comparison for Scenario 2.



(d) Packet loss for Scenario 2.



(e) PSNR comparison for Scenario 3.



(f) Packet loss for Scenario 3.

FIGURE 5.4: PSNR comparison and packet loss for all three scenarios

FDSP, and thus all 1200 frames are reconstruced/decoded. In Scenario 2, there are 148 frames lost by using the pure-UDP method, whereas there is no missing frames for FDSP. In Scenarios 3, there are total of 103 missing frames out of 1200 for pure-UDP. Again, FDSP experiences no missing frames because slice headers are prioritized using TCP.

Table 5.1 shows the buffering requirements for all three scenarios. Both pure-TCP and FDSP have same initial buffering time, and increases from 2 second to 2.56 second as a reaction to network saturation. However, pure-TCP incurs frequent rebuffering. During 40 seconds of video streaming, rebuffering occurs 6 ˜19 times and each lasts 0.95 ˜1.46 seconds. In contrast, FDSP does not have any rebuffering. This again shows that FDSP is very effective in a congested network because pure-UDP and pure-TCP tend to be unacceptable in terms of visual quality and delay, respectively.

	Streaming Method	Initial Buffering (sec.)	Rebuffering Count	Avg. Re-buffering Time (sec.)
Scenario1	FDSP	2.0	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	2.0	6	0.95
Scenario2	FDSP	2.3	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	2.3	17	1.34
Scenario3	FDSP	2.56	0	N/A
	pure-UDP	0.0	0	N/A
	pure-TCP	2.56	19	1.46

TABLE 5.1: Buffering comparison for all three scenarios (Sub-stream Overlapping).

Table 5.2 shows the ready times and playout deadlines for sub-streams for FDSP. Note that sub-stream1 is not listed in this table because its ready time is in fact the initial buffering time shown in Table 5.1. The playout deadline for each sub-stream is determined by its length, i.e., 10 seconds. As long as the ready time is less than playout deadline, no rebuffering is required. As network congestion increases, ready times also increase. However, ready times for all sub-streams are still much earlier than their playout deadlines. Therefore, no rebuffering is required as indicated in Table 5.1.

	Sub-stream	Ready Time (sec.)	Playout Deadline (sec.)
Scenario1	2	3.78	10
	3	5.37	20
	4	7.01	30
Scenario2	2	4.80	10
	3	7.92	20
	4	11.47	30
Scenario3	2	4.82	10
	3	6.95	20
	4	10.91	30

TABLE 5.2: Sub-stream ready times for FDSP for all three scenarios.

6. PERCENTAGE BASED PRIORITIZATION

6.1. Motivation

As indicated in Sec. 5.3.2, the ready times for sub-streams for all three scenarios is earlier than playout deadline. For example, in Scenario 2, the 4^{th} sub-stream's ready time is at 11.47 seconds, which is 18.53 seconds earlier than the playout deadline of 30 seconds. This implies that the network condition can handle more data to be prioritized and sent via the TCP tunnel. If FDSP can send additional data via TCP, the visual quality can be further improved.

6.2. Solution

In order to utilize the slack time (i.e., difference between sub-stream ready time and playout deadline) and further improve visual quality, a new module called *Percentage Based Prioritization* (PBP) (see Fig. 4.1) is added so that FDSP can prioritize more bitstream syntax elements in addition to SPS, PPS and slice header. PBP selects syntax elements according to an input parameter called `PERCENT`. For example, if `PERCENT` is defined to be 10%, then 1 out of 10 packets will be sent via TCP. PBP extends the flexility of FDSP because, if desired, any syntax element can be prioritized.

In our implementation, PBP is used to prioritize frames larger than 100 Kbytes. These large frames are usually I-frames, and therefore, are more important than B-frames as mentioned in Sec. 2.1.

6.3. Simulation and Results

FDSP with PBP was simulated using network Scenario 2 described in Sec. 5.3. Input parameter PERCENT was set to two different values in order to progressively show the visual improvement. For the first case, PERCENT is set to 50%, and thus, FDSP will send 50% of the packets of frames that are larger than 100 Kbytes via TCP in addition to SPS, PPS, and slice headers. For the second case, PERCENT is set to 90% that allows even more data to be sent via TCP.

Fig. 6.1 shows PSNR comparison between FDSP with no PBP, FDSP with 50% PBP, and FDSP with 90% PBP. Note that the result for FDSP with no PBP is obtained directly from Sec. 5.3. As expected, higher PBP percentage results in better PSNR. The average PSNR for FDSP with no PBP is 83.08 dB, while PSNR for PBP with 50% PBP is 90.99 dB. In particular, PSNR for 90% PBP achieves perfect PSNR of 111 dB. This indicates that FDSP with PBP is very effective in improving visual quality because it guarantees more data to be received in addition to SPS, PPS, and slice header.

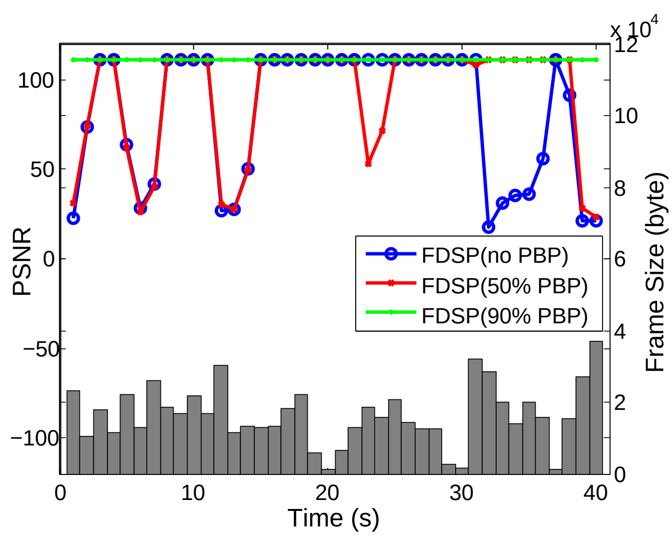


FIGURE 6.1: PSNR comparison for FDSP with no PBP, FDSP with 50% PBP, and FDSP with 90% PBP.

Fig. 6.2 shows visual comparison for Frame 1187. Clearly, FDSP has better performance than pure-UDP. In addition, visual quality progressively improves as PBP percentage increases.

(a) Frame 1187 for pure-UDP.

(b) Frame 1187 for FDSP (no PBP).

(c) Frame 1187 for FDSP (50% PBP).

(d) Frame 1187 for FDSP (90% PBP).

FIGURE 6.2: Visual quality comparison of Frame 1187.

Table 6.1 shows ready times for sub-streams for FDSP with no PBP, FDSP with 50% PBP, and FDSP with 90% PBP. Ready time for each sub-stream increases as the PBP `PERCENT` parameter increases. However, sub-streams' ready times are still earlier than playout deadlines and therefore no rebuffering is needed.

In summary, FDSP with 90% PBP achieves perfect PSNR and yet there is no rebuffering. In comparison, pure-TCP also achieves perfect PSNR but rebuffers 17 times. Moreover, compare to pure-UDP, FDSP with 90% PBP achieves 60 dB higher PSNR. The results achieved by FDSP with 90% PBP are considerably better than pure-TCP and pure-UDP and therefore FDSP clearly has advantages over pure-TCP and pure-UDP methods.

Sub-stream	FDSP Ready Time (sec.)			Playout Deadline (sec.)
	no PBP	\w 50% PBP	\w 90% PBP	
2	4.80	7.35	9.83	10
3	7.92	12.02	14.13	20
4	11.47	16.47	17.84	30

TABLE 6.1: Sub-stream ready times for FDSP with no PBP, FDSP with 50% PBP, and FDSP with 90% PBP.

7. CONCLUSION

This thesis presented a new streaming method called FDSP, which utilizes the benefit of both UDP and TCP. In addition, two techniques called *Sub-stream Overlapping* and *Percentage Based Prioritization* were presented to further improve its performance. The proposed FDSP was implemented within OEFMON and compared against traditional pure-UDP and pure-TCP methods. Two 1920×1080 @30fps HD video clips were used to evaluate the three streaming methods in an ad-hoc network environment. Six network scenarios were simulated and both delay and visual quality were analyzed. Our analysis shows that FDSP achieves higher PSNR than pure-UDP and less buffering time than pure-TCP for all six scenarios. This shows that FDSP is effective in striking a balance between delay and visual quality and thus has advantage over pure-UDP and pure-TCP methods.

BIBLIOGRAPHY

1. I. E. Richardson, *The H.264 Advanced Compression Standard*, 2nd ed. John Wiley and Sons, Ltd., 2010.
2. "AirPlay," Available at http://www.apple.com/itunes/airplay/.
3. "How to Connect Laptop to TV with Intel Wireless Display (WiDi)," Available at http://www.intel.com/content/www/us/en/architecture-and-technology/intel-wireless-display.html.
4. "ViVu," Available at http://www.cavium.com/PureVu_WiVu-Solution.html.
5. S. Wenger, "H.264/AVC over IP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 645 – 656, July 2003.
6. T. Kim and M. H. Ammar, "Receiver Buffer Requirement for Video Streaming over TCP," in *Proceedings of Visual Communications and Image Processing Conference*, 2006, pp. 422–431.
7. X. Shen, A. Wonfor, R. V. Penty, and I. H. White, "Receiver Playout Buffer Requirement for TCP Video Streaming in the Presence of Burst Packet Drops," in *London Communications Symposium*, 2009.
8. E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The Delay-Friendliness of TCP," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 49–60, June 2008. [Online]. Available: http://doi.acm.org/10.1145/1384529.1375464
9. T. Porter and X. H. Peng, "Hybrid TCP/UDP Video Transport for H.264/AVC Content Delivery in Burst Loss Networks," in *2011 IEEE International Conference on Multimedia and Expo (ICME)*, July 2011, pp. 1–5.
10. "Video LAN Client," Available at http://www.videolan.org/vlc/index.html.
11. "Wireshark," Available at http://www.wireshark.org/.
12. "Elecard," Available at http://www.elecard.com/en/index.html.
13. "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements," *IEEE Std 802.11e-2005 (Amendment to IEEE Std 802.11, 1999 Edition (Reaff 2003)*, 2005.

14. A. MacAulay, B. Felts, and Y. Fisher, "Whitepaper – IP Streaming of MPEG-4: Native RTP versus MPEG-2 Transport Stream," http://http://www.envivio.com/files/white-papers/RTPvsTS-v4.pdf, 2005.

15. D. Wu, Y. T. Hou, W. Zhu, Y. Q. Zhang, and J. M. Peha, "Streaming Video over the Internet: Approaches and Directions," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 282 –300, March 2001.

16. Y. Wang and Q. F. Zhu, "Error Control and Concealment for Video Communication: A Review," *Proceedings of the IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.

17. Y. Xu and Y. Zhou, "H.264 video communication based refined error concealment schemes," *Consumer Electronics, IEEE Transactions on*, vol. 50, no. 4, pp. 1135 – 1141, Nov. 2004.

18. A. Nafaa, T. Taleb, and L. Murphy, "Forward Error Correction Strategies for Media Streaming over Wireless Networks," *IEEE Communications Magazine*, vol. 46, no. 1, pp. 72–79, Jan. 2008.

19. J. Kim, R. M. Mersereau, and Y. Altunbasak, "Distributed Video Streaming Using Multiple Description Coding and Unequal Error Protection," *IEEE Transactions on Image Processing*, vol. 14, no. 7, pp. 849–861, July 2005.

20. B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: An Analytic Performance Study," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 4, no. 2, pp. 16:1–16:22, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1352012.1352020

21. R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang, "Measuring the Quality of Experience of HTTP Video Streaming," in *2011 IFIP/IEEE International Symposium on Integrated Network Management*, May 2011, pp. 485–492.

22. "FFmeg," Available at http://ffmpeg.org.

23. C. Lee, M. Kim, S. J. Hyun, S. Lee, B. Lee, and K. Lee, "OEFMON: An Open Evaluation Framework for Multimedia Over Networks," *IEEE Communications Magazine*, vol. 49, no. 9, pp. 153–161, Sept. 2011.

24. "DirectShow," Available at http://msdn.microsoft.com/en-us/library/windows/desktop/dd375454

25. "QualNet," Available at http://www.scalable-networks.com/content/products/qualnet.

26. J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th ed. USA: Addison-Wesley Publishing Company, 2009.

27. "h264bitstream," Available at http://h264bitstream.sourceforge.net/.

28. "AviSynth," Available at http://avisynth.org/mediawiki/Main_Page.