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

Jing Zhao and Ben Lee
Oregon State University
School of Electrical Engineering and Computer Science
Corvallis, OR 97331
{zhaoji, benl}@eecs.orst.edu

Tae-Wook Lee, Chang-Gone Kim, and Jone-Keun Shin
LG Display Co. Ltd
LCD Laboratory
Paju-si, Gyeonggi-do, Korea
{twlee, cgkim02, rgbshin}@lgdisplay.com

Jinsung Cho
Kyung Hee University
Department of Computer Engineering
Yongin-si 446-701, Korea
chojs@khu.ac.kr

ABSTRACT

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 paper 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. 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.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Communications Applications; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*wireless communication*

General Terms

Design, experimentation, performance

Keywords

HD Video Streaming, H.264/AVC, WLANs, TCP, UDP

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® [1], Intel WiDi® [2], and Cavium WiVu® [3]. 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 [4]. However, wireless 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 [5–7], 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 [8]. 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

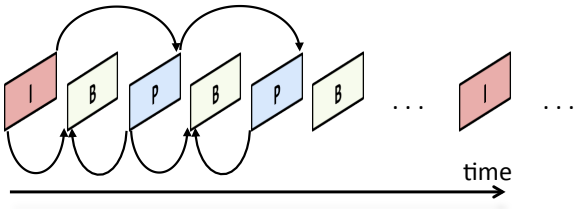


Figure 1: Typical GOP structure. Each arrow indicates the relationship between current (i.e., predicted) frame and reference frame(s).

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 paper 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 paper 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 paper is organized as follows: Sec. 2 presents a background on H.264, streaming protocols, and packetization methods. Sec. 3 discusses the related work. The proposed streaming method is presented in Sec. 4 and our experimental study and results are discussed in Sec. 5. Finally, Sec. 6 concludes the paper and discusses possible future work.

2. BACKGROUND

This section 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 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 mobile 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.

2.1.1 I, P, and B-Frame

An encoded video stream consists of a sequence of Group of Pictures (GOPs) as shown in Fig. 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

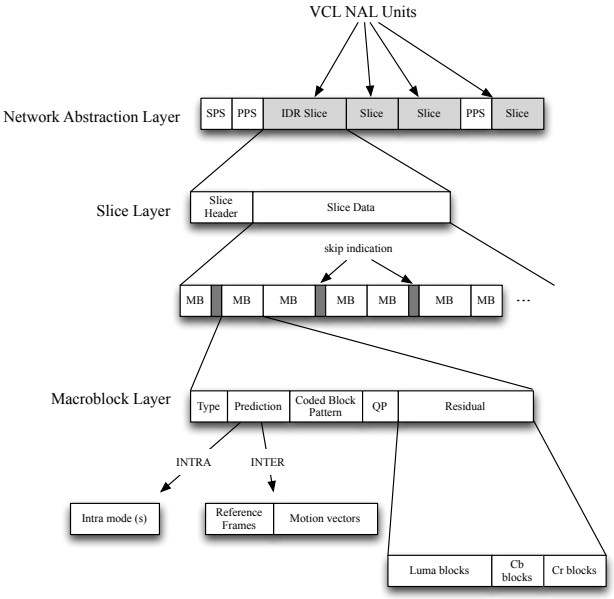


Figure 2: H.264 bitstream syntax [9].

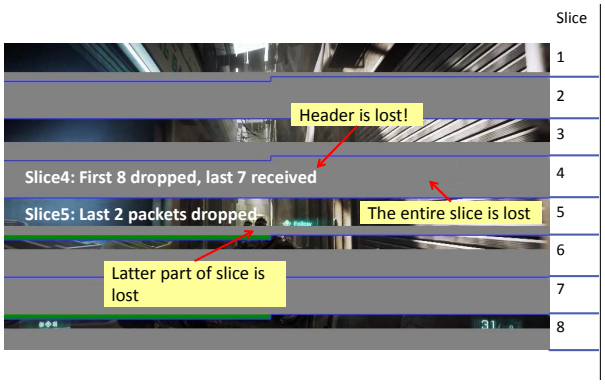
reference frame(s) during decoding. If a reference frame contains errors, these errors will propagate through subsequent frames that refer to this frame. Since an I-frame does not depend on any other frame, error propagation will cease when a new I-frame arrives. Consequently, I-frames should be given higher priority if possible during video streaming.

2.1.2 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. 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. 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 mode employed. 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 header is lost, then the entire slice cannot be decoded even if the slice data is properly received [4, 9].

Fig. 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. 3a shows the original transmitted frame, while Fig. 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

(a) The original frame.



(b) The received frame.

Figure 3: Effect of slice header loss.

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.

2.1.3 *Data Partitioning*

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 [4]. 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 Sec. 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].

2.2 Streaming Protocols

Existing streaming protocols include Real Time Streaming Protocol (RTSP), HyperText Transfer Protocol (HTTP), Microsoft Media Server (MMS), and Real-time Transport Protocol (RTP). Note that RTSP, HTTP, MMS, and RTP

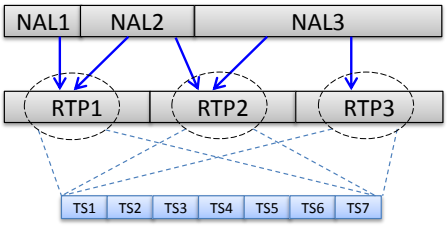


Figure 4: Method 1 H264→TS→RTP.

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.

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.

For our purpose, the focus is on RTP/UDP and RTP/TCP direct streaming as they are fundamental to all other streaming protocols.

2.3 Packetization Methods

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

2.3.1 *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. 4. 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.

2.3.2 *Method 2 (H264→RTP)*

This method shown in Fig. 5 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

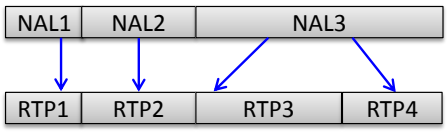


Figure 5: Method 2 H264→RTP.

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 [4].

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 FDPS 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 [4, 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. 2.1.2, 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 [4, 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 [4]. 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* [5]. 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 [5,6]. Besides buffer size estimation, Brosh *et al.* presented packet splitting and parallel connection methods to reduce TCP delay [7].

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 [8]. The authors proposed the Hybrid TCP/UDP method that prioritizes video data by using H.264 Data Partitioning (see Sec. 2). However, our proposed FDSP method has a couple of advantages over their method. First, FDSP is more flexible because it is not tied to DP. 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, any syntax element from a video stream can be segregated and prioritized. 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 so what can be prioritized are fixed. Second, 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 Sec. 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. Moreover, 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-PROTOCOL VIDEO STREAMING

The system diagram of the proposed FDPS is shown in Fig. 6. 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 steer-

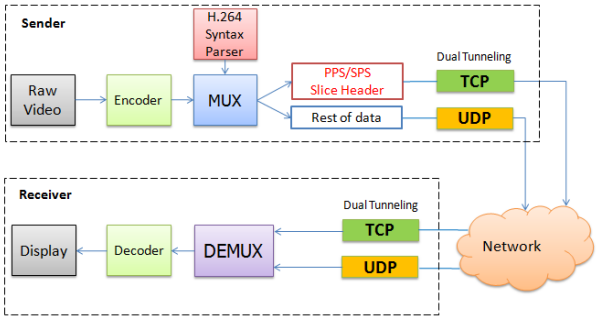


Figure 6: Flexible Dual-protocol Streaming system diagram.

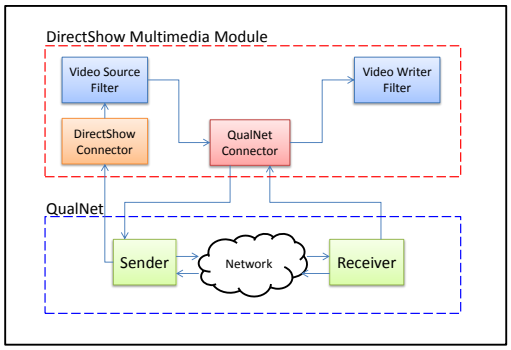


Figure 7: The general structure of OEFMON.

ing the two parts to UDP and TCP tunnels. *Dual Tunneling* keeps both UDP and TCP sessions active during video streaming. The Receiver consists of Dual Tunneling, DEMUX and a decoder. *DEMUX* re-orders the packets, merges UDP packets with TCP packets, and sends them to the *Decoder*, which is implemented using FFmpeg [22].

The proposed FDPS was implemented within *Open Evaluation Framework for Multimedia 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. 7. 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].

The following subsections describe 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. `MESSAGE`s such

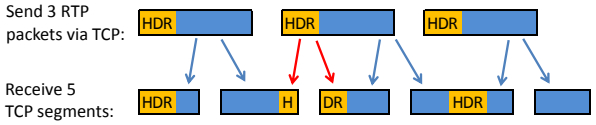


Figure 8: TCP Fragmentation.

as `MSG_TRANSPORT_FromAppOpen` (request to open TCP socket) and `MSG_TRANSPORT_FromAppListen` (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. 8 illustrates the fragmentation caused by TCP. In order to recover RTP packets from a TCP byte stream, TCP segments must be reassembled.

The length information of a UDP packet is required to accomplish TCP segment reassembly. Because the standard UDP header does not include length information, the 12-byte RTP header is extended by two bytes to indicate the length of each UDP 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

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 that contains a NAL unit 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 MUX/DEMUX

The implementation of MUX and DEMUX is relatively straightforward. At the sender, MUX takes as input the syntax information generated from the H.264 Syntax Parser. Based on this, MUX steers the corresponding RTP packets to either TCP or UDP tunnel. At the receiver, DEMUX first checks the timestamps of received RTP packets from the UDP tunnel and drops late RTP packets. If an RTP packet arrives in time, DEMUX merges this packet with

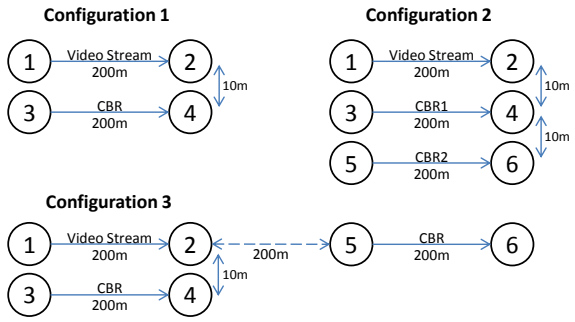


Figure 9: Simulated network scenarios.

other RTP packets from the TCP tunnel. DEMUX keeps merging on-time RTP packets and forms the final received video file.

5. EXPERIMENT SETUP AND RESULTS

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. In 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 adds one more CBR data of 4 Mbps so that the network becomes saturated. Scenario 3 repeats the network traffic of Scenario 1, but the nodes are now positioned in a classic hidden-node arrangement. The placement of devices for three scenarios are shown in Fig. 9. 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 paper 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.

The PSNR comparison for Scenario 1 is shown in Fig. 10. The figure contains PSNR values as well as frame sizes for the three streaming methods. As expected, pure-UDP has the worst PSNR with an average of 54 dB, while FDSP

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

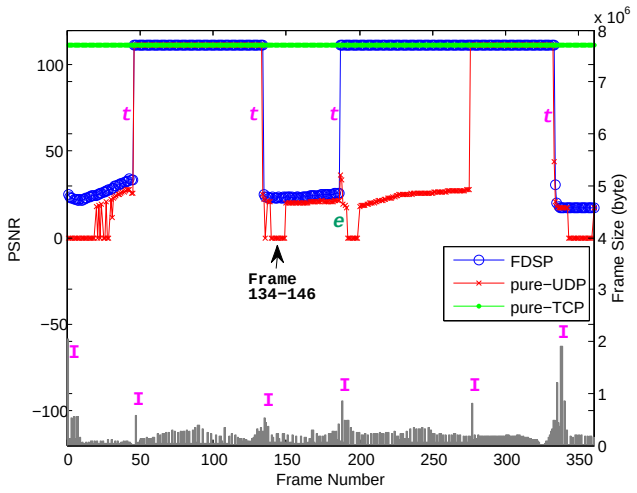


Figure 10: PSNR comparison for Scenario 1.

Streaming Method	Initial Buffering (sec.)	Rebuffering Count	Avg. Rebuffering Time (sec.)
FDSP	1.1	0	N/A
pure-UDP	0.0	0	N/A
pure-TCP	1.1	3	1.2

Table 1: Buffering comparison for Scenario 1.

achieves 79 dB. This clearly illustrates the advantage of FDSP over pure-UDP in terms of visual quality. Pure-TCP has perfect PSNR of 111 dB because there is no packet loss.

In Fig. 10, 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 P- or B-frames are unlikely be lost 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 similarity 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.

The delay analysis for Scenario 1 is shown in Table 1. Although pure-TCP and FDSP have same initial buffering time, pure-TCP incurs very frequent rebuffering. During 12 seconds of video streaming, rebuffering occurs three times and each lasts on average 1.2 seconds. As mentioned in Sec. 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

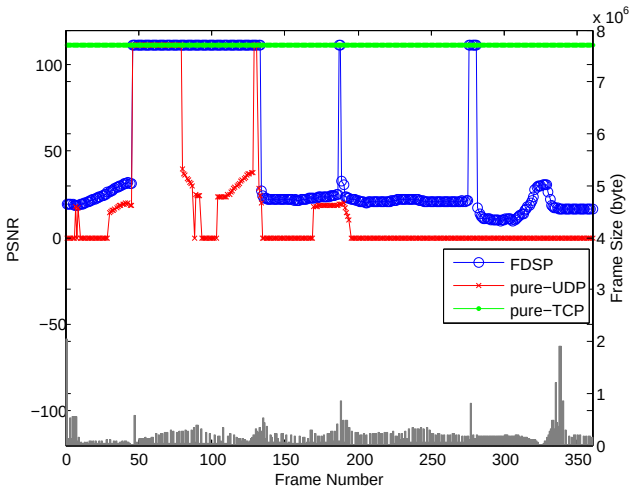


Figure 11: PSNR comparison for Scenario 2.

Streaming Method	Initial Buffering (sec.)	Rebuffering count	Avg. Rebuffering Time (sec.)
FDSP	1.6	0	N/A
pure-UDP	0.0	0	N/A
pure-TCP	1.6	6	1.6

Table 2: Buffering comparison for Scenario 2.

provides perfect visual quality. This is a clear advantage of FDSP over pure-TCP.

For Scenario 2, the network was saturated by introducing another CBR data of 4 Mbps. The PSNR and delay comparisons are shown in Fig. 11 and Table 2, respectively. 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. 11 shows that frames 134-146 are lost for pure-UDP, but these frames are not lost in FDSP. The packet-loss log in OEFMON shows that both pure-UDP and FDSP lose all the packets from 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. 12 shows the decoded frame 134. The upper-left watermark shows the frame number 134, which is the information retrieved from the packet that contains the slice header. 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.

Our delay analysis shows that initial buffering time for both FDSP and pure-TCP increases to 1.6 seconds, which is caused by network saturation. However, rebuffing for pure-TCP occurs six times and each lasts on average 1.6 seconds. This implies 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. Although FDSP incurs delay and results in non-perfect PSNR,

Figure 12: The decoded Frame 134 for FDSP.

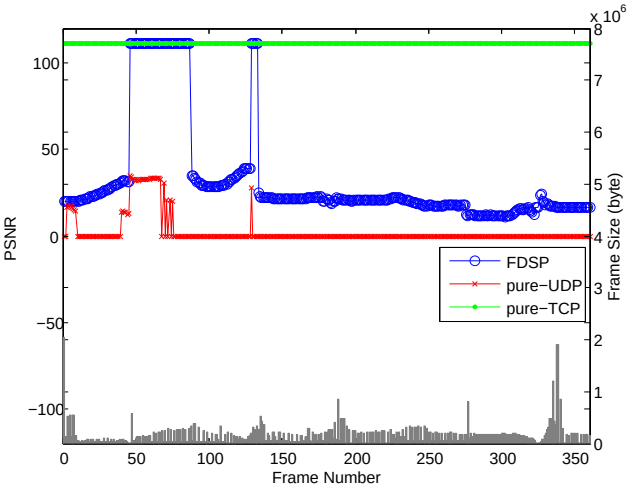


Figure 13: PSNR comparison for Scenario 3.

it effectively achieves a balance between delay and visual quality.

Fig. 13 and Table 3 show the PSNR and delay comparisons for Scenario 3, respectively. As can be seen, PSNR and delay further degrade 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 total of 320 missing frames out of 360. FDSP has an average PSNR value of 32 dB, which is significantly better than pure-UDP. Again, FDSP experiences no missing frames and this is due to prioritizing slice headers using TCP. Our delay analysis shows that hidden node further increases initial buffering time to 1.9 seconds for FDSP and pure-TCP. In addition, although pure-TCP has the same rebuffering count, each lasts on average 2.1 seconds.

As mentioned earlier, the current implementation of FDSP will first send all the important data via TCP and then send rest of data via UDP. For the test video, the initial buffering (i.e., the time spent sending all the important data) is less than 2 seconds. However, initial buffering can be quite long when streaming an entire movie. This issue can be solved

Streaming Method	Initial Buffering (sec.)	Rebuffering Count	Avg. Rebuffering Time (sec.)
FDSP	1.9	0	N/A
pure-UDP	0.0	0	N/A
pure-TCP	1.9	6	2.1

Table 3: Buffering comparison for Scenario 3.

by adding a special buffer. The streaming process for FDSP with the special buffer is as follows. First, an entire movie will be divided to serval sub-streams with a fixed length (i.e., n seconds). FDSP will send all the important data for the first n-second sub-stream via TCP and then rest of data for the first n-second sub-stream via UDP. As long as FDSP is not sending any UDP packet, it will send important data for next n-second video sub-streams via TCP. FDSP will stop and perform re-buffering only if the important data for a n-second sub-stream is not in the special buffer. This solution is similar to pure-TCP but the special buffer only stores important data instead of all the video data. Because the data required to send via TCP for FDSP is significantly less than pure-TCP, FDSP will experience much less rebuffering than pure-TCP. In addition, the size of the special buffer is determined by the fixed length (n seconds) of sub-streams and the choice of buffer size should be based on network conditions as discussed in [5]. The implementation of this special buffer is left as future work.

6. CONCLUSION AND FUTURE WORK

This paper presented a new streaming method called FDSP, which utilizes the benefit of both UDP and TCP. The proposed FDSP was implemented within OEFMON and compared against traditional pure-UDP and pure-TCP methods. A 1920×1080 @30fps HD video clip was used to evaluate the three streaming methods in an ad-hoc network environment. Three network scenarios were constructed and for each scenario 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 three 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.

As a future work, the special buffer will be added to FDSP to handle rebuffering and evaluated by streaming longer HD videos. The frequency of rebuffering for FDSP is expected to be very low because the data required to send via TCP is significantly less than the pure-TCP method. In addition, the function for supporting multiple slices per frame will be added to OEFMON. This will allow us to explore whether FDSP can provide better performance in terms of delay and visual quality by using videos encoded with multiple slices per frame.

Acknowledgment

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

7. REFERENCES

[1] AirPlay. Available at http://www.apple.com/itunes/airplay/.

[2] 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.

[3] ViVu. Available at http://www.cavium.com/PureVu_WiVu-Solution.html.

[4] S. Wenger. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):645 – 656, July 2003.

[5] T. Kim and M. H. Ammar. Receiver Buffer Requirement for Video Streaming over TCP. In *Proceedings of Visual Communications and Image Processing Conference*, pages 422–431, 2006.

[6] 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.

[7] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne. The Delay-Friendliness of TCP. *ACM SIGMETRICS Performance Evaluation Review*, 36(1):49–60, June 2008.

[8] 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)*, pages 1–5, July 2011.

[9] I. E. Richardson. *The H.264 Advanced Compression Standard*. John Wiley and Sons, Ltd., second edition, 2010.

[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*, 11(3):282 –300, March 2001.

[16] Y. Wang and Q. F Zhu. Error Control and Concealment for Video Communication: A Review. *Proceedings of the IEEE*, 86(5):974–997, May 1998.

[17] Y. Xu and Y. Zhou. H.264 video communication based refined error concealment schemes. *Consumer Electronics, IEEE Transactions on*, 50(4):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*, 46(1):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*, 14(7):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*, 4(2):16:1–16:22, May 2008.

[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*, pages 485–492, May 2011.

[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*, 49(9):153–161, Sept. 2011.

[24] DirectShow. Available at http://msdn.microsoft.com/en-us/library/windows/desktop/dd3754549.aspx.

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

[26] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 5th edition, 2009.

[27] h264bitstream. Available at http://h264bitstream.sourceforge.net/.

[28] AviSynth. Available at http://avisynth.org/mediawiki/Main_Page.