# Adaptive Bitstream Prioritization for Dual TCP/UDP Streaming of HD Video

Arul Dhamodaran, Mohammed Sinky, and Ben Lee

School of Electrical and Computer Science
Oregon State University
Corvallis, Oregon 97331
Email: {dhamodar, sinky, benl}@eecs.orst.edu

***Abstract*—Flexible Dual-TCP/UDP Streaming Protocol with Bitstream Prioritization (FDSP-BP) is a new method for streaming H.264-encoded High-definition (HD) video over wireless networks. This paper presents a novel technique to adaptively modify the Bitstream prioritization (BP) parameter based on network conditions. This technique selects the maximum BP value that satisfies the Transmission Control Protocol (TCP) rebuffering and User Datagram Protocol (UDP) packet loss rate constraints for each substream. This is achieved by passively estimating the UDP packet loss ratio and TCP rebuffering time on the sender side based on parameters, such as TCP Roundtrip Time (RTT), queue dispersal rate, peak delay, etc. Our simulation results show that FDSP with Adaptive-BP is able to significantly outperform FDSP-BP with static BP values and pure-TCP in terms of rebuffering time, and FDSP-BP with fixed BP values and pure-UDP in terms of packet loss. The end result is a better overall viewing experience during network congestion.**

***Keywords–Bitstream Prioritization; HD Video Streaming; TCP; UDP.***

## I. INTRODUCTION

HD video streaming applications can be broadly classified into Client-Server and Peer-to-Peer streaming services, which rely on either TCP or UDP protocol. Popular Client-Server streaming applications, such as Apple's Hypertext Transfer Protocol (HTTP) Live Streaming (HLS) [1] and Microsoft's Smooth Streaming [2], use HTTP-based streaming techniques that rely on TCP.

TCP is a reliable protocol and thus it guarantees perfect video frame quality. However, when network congestion occurs, TCP retransmissions cause delay leading to (re)buffering. A significant amount of work has been done to reduce the delay caused by TCP [3][4], but this issue still remains a major problem for video streaming. Figure 1 illustrates the effect of rebuffering caused by TCP packet delay, which occurs when TCP packets arrive at the receiver after the playout deadline. This delay causes the receiver to freeze frame and wait for enough TCP packets to arrive before resuming playback. In contrast, UDP minimizes delay but does not guarantee packet delivery. These lost packets, in turn, cause errors that propagate to subsequent frames. Figure 2 illustrates the detrimental effects of UDP packet loss on video quality.

Both *TCP rebuffering* and *UDP packet loss* affect the Quality of Experience (QoE) perceived by users. In our previous work, a new streaming technique called FDSP was proposed to exploit the benefits of both TCP and UDP protocols for streaming H.264 HD videos [5]. This is done by sending packets containing important information, such as Sequence Parameter Set (SPS), Picture Parameter Set (PPS), and slice

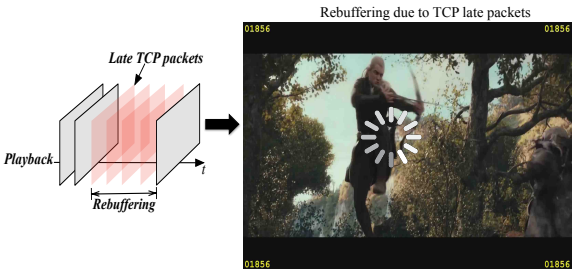



Figure 1. Rebuffering due to late TCP packets.

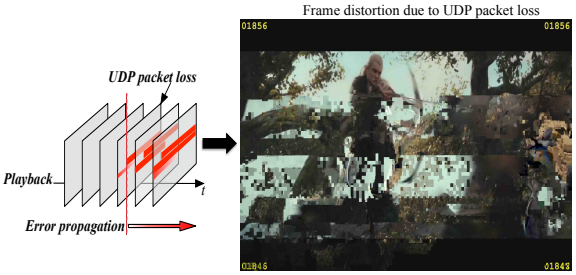



Figure 2. Frame distortion due to UDP packet loss. Note that packet loss also causes frame distortion in subsequent frames due to error propagation.

headers via TCP for guaranteed delivery and the rest of slice data packets via UDP. By utilizing both TCP and UDP streams, FDSP adds reliability to UDP while reducing the latency caused by TCP. FDSP was enhanced in [6] with the goal of reducing the impact of UDP packet loss during video stream using *Bitstream Prioritization* (BP). This method *statically* chooses the BP metric to classify select packets from an H.264 bitstream as high priority, which are then transported over TCP for guaranteed delivery. Our analysis of the BP parameter in [6] showed that an increase in BP resulted in a monotonic decrease in packet loss. However, an increase BP also increases TCP rebuffering time and instances due to the increase in the number of packets that are sent over TCP. Therefore, this paper proposes an *Adaptive-BP* technique to further improve the effectiveness of FDSP-BP based video streaming. This is achieved by *dynamically* adjusting the BP parameter in response to network conditions as well as QoE thresholds with the goal of minimizing both TCP rebuffering and UDP packet loss. Our simulation study shows that the proposed Adaptive-BP technique significantly reduces the TCP rebuffering time and UDP packet loss rate as compared to pure-TCP, pure-UDP, and static FDSP-BP streaming.

This paper is organized as follows. Section II discusses other TCP and UDP streaming techniques. An overview of the

 ISBN: 978-1-61208-439-8 

FDSP-BP method is shown in Section III. Section IV presents the proposed Adaptive BP technique. Sections V goes over the experimental setup and Section VI discusses the results of Adaptive BP as compared to that of FDSP BP, pure-UDP and pure-TCP. Finally, Section VII concludes the paper.

## II. 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 video playout [7]. UDP performance can be further improved by employing *Error Concealment* (EC) techniques to reduce the impact of data loss [8]. However, if important data, such as SPS, PPS, and slice headers are lost, the decoder simply cannot reconstruct the video even with the aid of EC. UDP packet loss can be tolerated by employing *Unequal Error Protection* (UEP), which prioritizes important data [7][9]. More advanced UEP methods incorporate *Forward Error Correction* (FEC) [9]. These methods are orthogonal to the proposed FDSP with Adaptive-BP technique, and thus, they can be used together.

Despite the latency issue with TCP, a significant fraction of commercial video streaming applications are based on TCP [10]. TCP provides guaranteed service so the transmitted packets are always preserved. Nevertheless, TCP's retransmission and rate control mechanisms incur delay, which can cause packets to arrive after their playout deadline. Much work has been done to minimize TCP rebuffering. Once such example is *Progressive Download*, which is widely used in HTTP streaming services such as Apple's HTTP live streaming (HLS) [1], HTTP Dynamic Streaming (HDS) [11], and Microsoft Smooth Streaming [2]. These techniques employ adaptive bitrate throttling based on available bandwidth to reduce TCP rebuffing.

Another approach to ensure proper delivery of important data to the destination is *bitstream prioritization*. In [12], a cross-layer packetization and retransmission strategy is proposed, where a video bitstream is prioritized based on distortion impact, delay constraints, and changing channel conditions. However, these parameters are heavily dependent on accurate feedback information, which incurs additional overhead on the bandwidth requirement. A *modified slicing scheme* that provides in-frame packet prioritization is proposed in [13], which exploits the unequal importance of different regions within a frame. These prioritization techniques, however, do not consider SPS, PPS, and slice header information for prioritization and hence are prone to slice and frame losses. Furthermore, authors in [12] do not consider H.264 videos, while authors in [13] employ custom modifications to H.264 slicing making it unsuitable for any H.264-encoded videos.

This paper expands the scope of our prior research on FDSP-BP [6] (see Section III) by dynamically modifying the bitstream prioritization (BP) parameter. The proposed Adaptive-BP technique passively estimates the network conditions and adaptively modifies the BP parameter to minimize TCP rebuffering time and UDP packet loss.

## III. FDSP Overview

This section provides a brief overview of the underlying details of FDSP for completeness (see [5] and [6] for details).

Basic FDSP architecture is shown in Figure 3 [5]. Here, the FDSP sender consists of five main components: (1) H.264 Syntax Parser, (2) RTP Packetizer, (3) Demultiplexer (DEMUX), (4) the BP selection module, and (5) Dual Tunneling (UDP+TCP). The *H.264 Syntax Parser* is responsible for identifying SPS, PPS, and slice headers (SH). It also works with the *RTP Packetizer* to generate the RTP payload format containing Network Abstraction Layer (NAL) units for H.264 video [14] as illustrated in Figure 4 . This allows SPS and PPS information to be combined with the slices. The *Demultiplexer* splits the RTP packets into the TCP or UDP stream based on the packet contents. The *BP Selection module* sets the BP parameter. *Dual Tunneling* is employed to keep both TCP and UDP sessions active during video streaming.

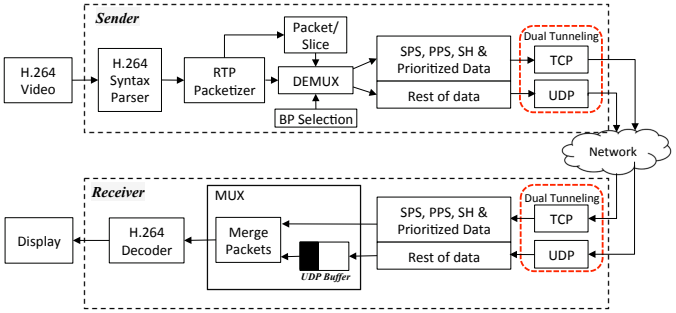



Figure 3. Flexible Dual-tunnel Streaming Protocol (FDSP) Architecture [5] augmented with modified MUX and DEMUX modules for FDSP-BP.

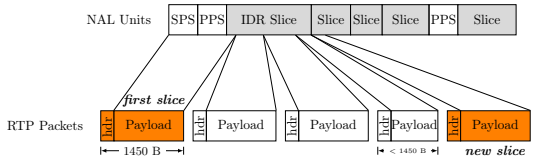



Figure 4. IETF RFC 6184 RTP packetization of H.264 NAL Units modified to allow parameter set NAL Units to be grouped with VCL NAL Units (slices). RTP packets that hold H.264 slice headers are shown in orange. [6]

The FDSP receiver comprises of three modules: (1) Dual Tunneling (TCP+UDP), (2) Multiplexer (MUX), and (3) H.264 decoder. The *Dual Tunneling* is employed to receive the TCP and UDP packets. The *Multiplexer* is responsible for rearranging the packets and discarding late UDP packets. It also combines the TCP and UDP packets based on the timestamp information to reassemble the frames.

The FDSP sender first segments a video into 10 sec. *substreams*, as done in HLS [1]. Then, all the TCP packets containing SPS, PPS, and slice headers are sent prior to sending UDP packets containing slice data. Thus, the receiver must wait for its respective TCP data to arrive before playback. To avoid frequent rebuffering caused by TCP packet delay, the transmission of UDP packets for the current substream is overlapped with the transmission of TCP packets for the next substream.

### A. FDSP-BP

FDSP-BP assigns the BP parameter *statically* to further reduce packet loss by sending additional high priority data, such as I-frame packets, over TCP. FDSP-BP can be applied to any types of frames, or even to all the frame types. However, BP is only applied to packets containing I-frame data because they serve as reference frames and any loss in I-frame data leads to error propagation to the entire Group Of Picture (GOP) sequence.

 ISBN: 978-1-61208-439-8 

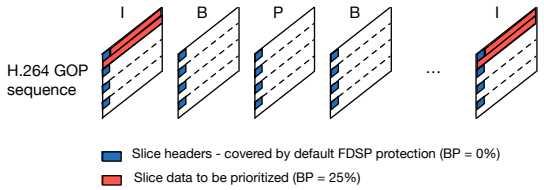



Figure 5. BP applied to a 4-slice H.264 video sequence. When BP is applied, packets are selected sequentially from the start of the frame.

```
1: procedure ADAPTIVE-BP(i)
2: BP_flag = 0
3: NBP_flag = 0
4:     for each BP do; where BP = 0.0, 0.1, 0.2, ..., 1.0
5:         Calculate E[RBT_i^TCP]^BP and E[PLR_i^UDP]^BP
6:     end for
7:     for each BP do; where BP = 0.0, 0.1, 0.2, ..., 1.0
8:         if E[RBT_i^TCP]^BP ≤ RBT_th && E[PLR_i^UDP]^BP ≤ PLR_th then
9:             BP_i = BP
10:            BP_flag = 1
11:        end if
12:    end for
13:    if BP_flag == 0 then
14:        BP_i = 1.0
15:        for each BP do; where BP = 0.0, 0.1, 0.2, ..., 1.0
16:            if E[PLR_i^UDP]^BP ≤ PLR_th && BP_i > BP then
17:                BP_i = BP
18:                NBP_Flag = 1
19:            end if
20:        end for
21:    end if
22:    if BP_flag == 0 && NBP_flag == 0 then
23:        BP_i = 0.0
24:    end if
25: return BP_i
26: end procedure
```

Figure 6. Adaptive-BP algorithm.

The prioritization of I-frame packets using the BP parameter is shown in Figure 5. Here, if BP parameter is set to zero, then it defaults to basic FDSP, where SPS, PPS, and slice headers are the only packets that will be sent via TCP. If BP is 25% then a quarter of all I-frame packets would be sent via TCP. Although it is possible to select any distribution of the I-frame to be sent via TCP, a sequential order of I-frame packets are selected to be sent via TCP to achieve QoE. Increasing BP results in increasing the number of TCP packets, thus increasing the probability of TCP rebuffering, but it reduces UDP packet loss and error propagation due to the proportional reduction in the number of UDP packets.

## IV. Adaptive Bitstream Prioritization

The procedure of the proposed Adaptive-BP algorithm is shown in Figure 6, which consists of four parts. In the first part (*lines* 4-6), the *estimated TCP rebuffering time for substream* $i$, $E[RBT_i^{TCP}]$, and the *estimated UDP packet loss rate for substream* $i$, $E[PLR_i^{UDP}]$, are calculated for each $BP$, where $BP$ = 0%, 10%, ..., 100%, for $i \geq 2$. BP increments of 10% was chosen based on our experiments, which showed that it provides the right balance between computational requirement and its effect on QoE. Note that Figure 6 is executed after the completion of transmission of the TCP portion for substream $i-1$ so that information on rebuffering time and packet loss can be gathered. The calculations of $E[RBT_i^{TCP}]$ and $E[PLR_i^{UDP}]$ will be discussed in Secions. IV-A and IV-B, respectively. By default, the BP value for the first substream, $BP_1$, is set to 100%, which is done to reduce the possibility of UDP packet loss in case the network is congested when streaming starts.

In the second part (*lines* 7-12), a check is made for each $BP$ value to determine if $E[RBT_i^{TCP}]^{BP}$ and $E[PLR_i^{UDP}]^{BP}$ are less than equal to the *TCP rebuffering time threshold*, $RBT_{th}$, and the *UDP Packet Loss Rate (PLR) threshold*, $PLR_{th}$, respectively, which are the adjustable QoE thresholds. If both these conditions are satisfied, then the BP value for the $i^{th}$ substream, $BP_i$, is set to $BP$ and *BP_flag* is set to 1 to indicate that both threshold conditions were satisfied and $BP_i$ has been set. Since this is done for all the $BP$ values starting with $BP$ equal to 0%, the for-loop will select the *highest* value of $BP$ that satisfies both QoE thresholds. This is because packet loss is more detrimental to video quality as it also leads to error propagation. Therefore, the proposed Adaptive-BP algorithm is more sensitive to packet loss over rebuffering.

The third part of the algorithm (*lines* 13-21) is executed only when none of the $BP$ values satisfy both QoE requirements. If so, $BP_i$ is initially set to 100% and then a check is made to determine if $E[PLR_i^{UDP}]$ satisfies $PLR_{th}$ and $BP_i$ is greater than $BP$ for each $BP$ value. For each iteration, if both of these conditions are satisfied, then $BP_i$ is set to $BP$ and the *NBP_flag* is set 1 to indicate that $PLR_{th}$ was satisfied and $BP_i$ was set. Note that this for-loop will select the *lowest* value of $BP$ that satisfies these two conditions. This is because as $BP$ increases within the range of acceptable $BP$ values, the improvement in video quality is marginal compared to the increase in rebuffering time.

Finally, the fourth part of the algorithm (*lines* 22-24) is executed if none of the $BP$ values satisfy any of the QoE thresholds. In this case, the visual quality of a given substream is considered bad because the $PLR_{th}$ constraint cannot be met resulting in excessive frame distortion and error propagation. In addition, since none of the $BP$ values satisfy $PLR_{th}$, there is no reason to increase $BP$ as this will increase rebuffering time. Therefore, $BP_i$ is set to 0% to minimize TCP rebuffering time. The only drawback of the estimation algorithm is the accuracy of UDP packet loss estimation at the sender. However, since BP selection is dependent on both TCP rebuffering estimate and UDP PLR estimate, the impact of UDP PLR estimation errors is marginal.

### A. Estimating TCP Rebuffering Time

$E[RBT_i^{TCP}]$ for $i \geq 2$ can be calculated as

$$E[RBT_i^{TCP}] = \begin{cases} E[T_i^{TCP}] - P_i, & \text{if } E[T_i^{TCP}] > P_i \\ 0 & otherwise, \end{cases} \quad (1)$$

where $E[T_i^{TCP}]$ represents the *estimated TCP transmission time for substream* $i$ and $P_i$ is the *playout time for the* $i^{th}$ *substream*. Here, rebuffering occurs whenever $E[T_i^{TCP}]$ exceeds $P_i$ as shown in (1).

$E[T_i^{TCP}]$ is represented using the following equation:

$$E[T_i^{TCP}] = \frac{RTT_{avg_{i-1}}}{2} \times [N_i^{TCP_D} + (BP_i \times N_i^I) + N_{i-1}^{UDP}], \quad (2)$$

where $RTT_{avg_{i-1}}$ represents the average round trip time of TCP packets for substream $i-1$, $N_i^{TCP_D}$ is the default number of TCP packets sent for SPS, PPS and slice headers

for substream $i$, $N_i^I$ is the total number of I-frame packets for substream $i$, and $N_{i-1}^{UDP}$ is the number of UDP packets for substream $i-1$. Note that the $N_{i-1}^{UDP}$ term in (2) takes into consideration that all the UDP packets of substream $i-1$, as well as all the TCP packets for substream $i$ need to be transmitted before UDP packets for substream $i$ can be transmitted. Equation (2) also shows that increasing or decreasing the BP parameter will result in proportional increase or decrease in the number of TCP packets, which in turn determines the total TCP transmission time for a substream.

$P_i$ can be calculated using the equation below:

$$P_i = T_{sl} + E[RBT_{i-1}^{TCP}], \tag{3}$$

where $T_{sl}$ refers to the fixed substream length and $E[RBT_{i-1}^{TCP}]$ refers to the estimated rebuffing time of substream $i-1$ that reflects the shift in playout time due to rebuffering. In our implementation, $T_{sl}$ is set to be 10 sec.

### B. Estimating UDP Packet Loss Ratio

$E[PLR_i^{UDP}]$ is represented by the following equation:

$$E[PLR_i^{UDP}] = \frac{E[PLR_{i-1}^{UDP}] \times N_i^{UDP}}{N_i^T}, \tag{4}$$

where $E[PLR_{i-1}^{UDP}]$ represents the estimated UDP PLR for substream $i-1$, $N_i^{UDP}$ represents the total number of packets to be sent through UDP for stream $i$, and $N_i^T$ is the total number of packets for substream $i$. Since FDSP sends both TCP and UDP packets, $N_i^{UDP}/N_i^T$ is used to estimate UDP PLR based on the total number of packets in substream $i$.

$N_i^{UDP}$ can be calculated using the following equation:

$$N_i^{UDP} = N_i^T - [N_i^{TCP_D} + (BP_i \times N_i^I)]. \tag{5}$$

Equation (5) shows that the number of UDP packets depends on the number of TCP packets, which is a function of $BP_i$. On the other hand, $E[PLR_{i-1}^{UDP}]$ can be calculated based on the number of UDP packets lost in during substream $i-1$, which is given by

$$E[PLR_{i-1}^{UDP}] = \frac{1}{N_{i-1}^{UDP}} \sum_{k=1}^{N_{i-1}^{UDP}} \mathbf{1}_{\{\lambda_{k_{i-1}} = D_{th}\}}, \tag{6}$$

where $N_{i-1}^{UDP}$ represents the total number of UDP packets in substream $i-1$. $\mathbf{1}_{\{.\}}$ is the indicator function, $\lambda_{k_{i-1}}$ is the average time the $k^{th}$ UDP packet for substream $i-1$ spent in the IP queue, $D_{th}$ is the queue delay threshold, and $\lambda_{k_{i-1}} = D_{th}$ indicates that the packet was lost [15]. $D_{th}$ is the time spent by the last packet in the IP queue when it becomes full for the first time.

### C. An Example

The results of applying Adaptive-BP algorithm for three sample substreams (Case 1, Case 2, and Case 3) is shown in Figure 7, which are derived from the example video clip used in our analysis (see Section V). After obtaining $E[RBT_i^{TCP}]$ and $E[PLR_i^{UDP}]$ for all $BP$ values, the Adaptive-BP algorithm narrows the possible $BP$ values that satisfy both $RBT_{th}$ and $PLR_{th}$ thresholds. For these experiments, $RBT_{th}$ and $PLR_{th}$ are assumed to be 1 sec. and 0.05, respectively (see Section V).

In Case 1, the BP values that satisfy both thresholds are $BP$ = 0% and $BP$ = 10%. $BP$ = 0% results in estimated rebuffing time of 0.38 sec. and estimated UDP PLR of 0.05. On the other hand, $BP$ = 10% results in estimated rebuffing time of 0.57 sec. and estimated UDP PLR of 0.04. Although both BP values can be used, $BP_i$ is chosen to be 10% because in terms of QoE a 1% increase in UDP PLR is more detrimental to video quality than 0.11 sec. increase in rebuffering time. In Case 2, none of the BP values satisfy both thresholds, hence $BP_i$ is set the minimum BP value that satisfies the UDP PLR threshold, i.e., $BP_i$ = 60%. This is because any increase in BP results in significant increase in rebuffering time with minimal improvement in video quality. On the other hand, decreasing BP reduces rebuffering time but it leads to PLR greater than 0.05, which is considered bad video quality [16].

In Case 3, none of the BP values satisfies any of the QoE thresholds indicating bad visual quality [16]. Here, any increase in BP leads to a significant increase in rebuffing time but its improvement in video quality is negligible, thus $BP_i$ is set 0% by default to minimize rebuffering time.

## V. Experimental Setup

Our simulation environment is *Open Evaluation Framework For Multimedia Over Networks* (OEFMON) [17], which is composed of a multimedia framework *DirectShow*, and a network simulator *QualNet*. OEFMON allows a raw video to be encoded and redirected to a simulated network to gather statistics on the received video.

The simulated network is an 802.11g ad-hoc network with a bandwidth of 54 Mbp. Note, the version of the qualnet simulator used for our study only supports the IEEE 802.11g standard. However, the simulation study can easily adopted to 802.11n by having more background traffic to saturate the network. The network scenario used is an 8-node configuration shown in Figure 8. The distance between the source and the destination is set to be 5 m and the distance between the streaming node pairs is set to be 10 m. These distances were chosen to represent the proximity of multiple streaming devices that exist in a modern household. The primary test video is being streamed between nodes 1 and 2, while the remaining nodes are used to generate an aggregate Constant Bit Rate (CBR) background traffic of 50 Mbps to fully saturate the network.

The test video used for our simulation is the video from "The Hobbit" movie trailer, which contains 146 seconds of full HD video (1920×1080 @30fps, 4354 frames). The video is encoded using the x264 encoder with an average bit rate of 4 Mbps and four slices per frame.

The threshold parameters $RBT_{th}$ and $PLR_{th}$ are chosen based on recommendations from industry and literature studies. A recent study done by Conviva, which is a company that monitors Internet video delivery, reports that users react negatively when (re)buffering time exceeds 2% of the total length of the viewing session [18]. However, since FDSP reduces rebuffering by employing both TCP and UDP protocols, a more relaxed $RBT_{th}$ of 1 sec. (10% per substream) is used. On the other hand, a QoE study conducted in [16] showed that a PLR of less than 5% is considered acceptable video, hence $PLR_{th}$ is set to 0.05.

 ISBN: 978-1-61208-439-8

| UDP PLR | BP (%) | TCP Rebuf. (sec) |
|---|---|---|
| 0.05 | 0 | 0.38 |
| 0.04 | 10 | 0.57 |
| 0.04 | 20 | 1.29 |
| 0.04 | 30 | 1.67 |
| 0.04 | 40 | 2.15 |
| 0.04 | 50 | 2.40 |
| 0.04 | 60 | 3.43 |
| 0.04 | 70 | 4.07 |
| 0.03 | 80 | 4.28 |
| 0.03 | 90 | 4.42 |
| 0.03 | 100 | 4.61 |

UDP PLR Threshold (0.05)

TCP Rebuf. Threshold (1 sec.)

(a) Case 1: Both $RBT_{th}$ and $PLR_{th}$ are satisfied.

| UDP PLR | BP (%) | TCP Rebuf. (sec) |
|---|---|---|
| 0.07 | 0 | 0.57 |
| 0.07 | 10 | 1.28 |
| 0.06 | 20 | 1.79 |
| 0.06 | 30 | 2.47 |
| 0.06 | 40 | 3.21 |
| 0.06 | 50 | 3.48 |
| 0.05 | 60 | 4.12 |
| 0.05 | 70 | 4.55 |
| 0.05 | 80 | 5.07 |
| 0.04 | 90 | 5.22 |
| 0.04 | 100 | 5.49 |

TCP Rebuf. Threshold (1 sec.)

UDP PLR Threshold (0.05)

(b) Case 2: Only $PLR_{th}$ is satisfied.

| UDP PLR | BP (%) | TCP Rebuf. (sec) |
|---|---|---|
| 0.29 | 0 | 1.57 |
| 0.27 | 10 | 2.28 |
| 0.25 | 20 | 3.76 |
| 0.23 | 30 | 4.47 |
| 0.21 | 40 | 5.21 |
| 0.20 | 50 | 6.46 |
| 0.18 | 60 | 7.12 |
| 0.17 | 70 | 7.56 |
| 0.15 | 80 | 8.07 |
| 0.12 | 90 | 9.22 |
| 0.10 | 100 | 10.49 |

(c) Case 3: None of the thresholds are satisfied.

Figure 7. Examples of Adaptive-BP selection.

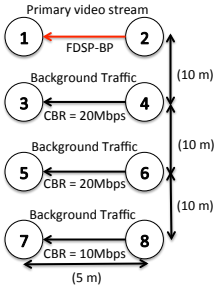
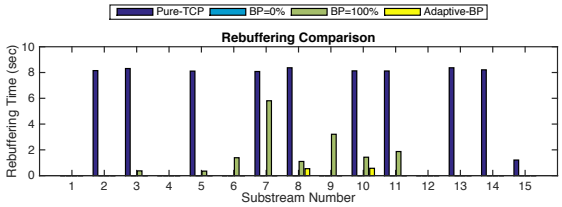
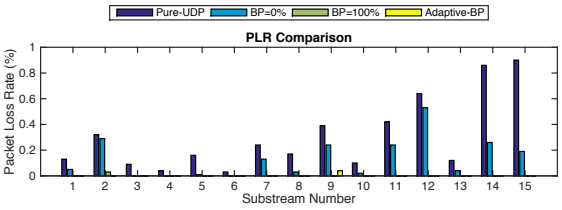



Figure 8. Simulated Network Scenario.




(a) Rebuffering comparison.




(b) PLR comparison.

Figure 9. Comparison of TCP rebuffering time and UDP PLR for pure-TCP, FDSP-BP with BP=0%, FDSP-BP with BP=100% and FDSP-BP with Adaptive-BP for the 146 sec. *Hobbit* video.

## VI. Results

The changes in rebuffering time by adaptively adjusting the BP parameter as compared to pure-TCP, FDSP-BP with BP=0%, and FDSP-BP with BP=100% is shown in Figure 9a. Pure-TCP based video streaming incurs a total of 10 instances of rebuffering with a total rebuffering time of 75.06 sec. FDSP-BP with BP=100% incurs 8 instances of rebuffering with a total rebuffering time of 14 sec. However, FDSP with Adaptive-BP only incurs 2 instances rebuffering with a total rebuffering time of 1.1 sec. In addition, Figure 9a shows that FDSP-BP with BP=0% does not incur any rebuffering, but this is achieved at the cost of increased PLR.

PLR reduction by Adaptive-BP as compared to pure-UDP, FDSP-BP with BP=0%, and FDSP-BP with BP=100% is as shown in Figure 9b. Pure-UDP based video streaming incurs a total loss of 9136 packets with PLR of 0.27. FDSP-BP with BP=0% incurs a loss of 7084 packets with PLR of 0.16. However, FDSP with Adaptive-BP only incurs a loss of 152 packets with PLR of 0.004. Note that FDSP-BP with BP=100% incurs a loss of 97 packets with a PLR of 0.003, but this marginal gain in PLR is achieved at the cost of increased rebuffering time.

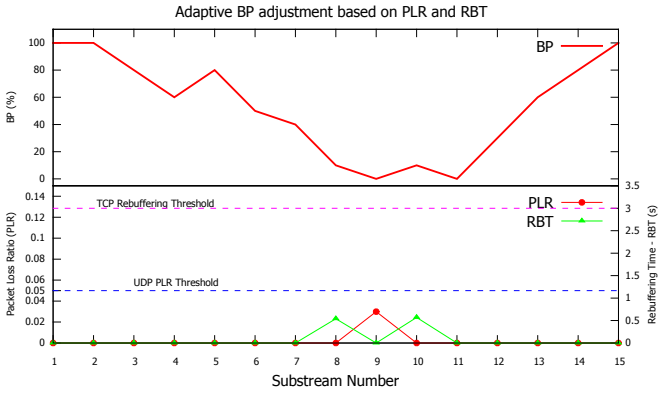
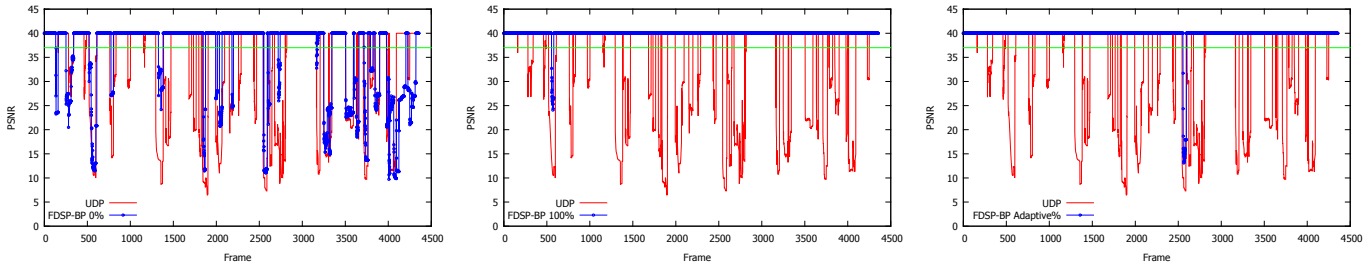



Figure 10. Adaptive BP selection for the 146 sec. *Hobbit* video.

FDSP with Adaptive-BP in action and its impact on packet loss and rebuffering is shown in Figure 10. The graph in the upper portion of the figure shows how $BP_i$ changes, while the graphs in the lower portion of the figure show the actual $RBT_i^{TCP}$ and $PLR_i^{UDP}$ for each substream. The TCP rebuffering ($RBT_{th}$) and the UDP PLR ($PLR_{th}$) thresholds are indicated as a dashed pink line and a dashed blue line, respectively. Initially, the BP value for the first substream starts at 100% by default. Afterwards, the BP values change based on $E[RBT_i^{TCP}]$ and $E[PLR_i^{UDP}]$. For substreams 2 to 8, Adaptive-BP efficiently adjusts the BP value so that no packet loss occurs. For substreams 9, UDP PLR is 0.04, but significantly outperforms pure-UDP and FDSP-BP with BP=0% with PLR of 0.4 and 0.24, respectively, as shown in Figure 9b. The two instances of TCP rebuffing that occur for substream 8 and 10 result in total rebuffering time of 1.1 sec.

### A. PSNR Comparison

The PSNR results of FDSP with Adaptive-BP against pure-UDP, FDSP with BP=0%, and FDSP with BP=100% is shown in Figure 11. Note that PSNR of 37 dB for a given frame




(a) FDSP with BP=0%. (b) FDSP with BP=100%. (c) FDSP with Adaptive-BP.

Figure 11. PSNR plots for the 146 sec. *Hobbit* video. The green line at 37 dB represents the PSNR threshold above which the human eye cannot perceive any quality difference.

is considered excellent quality [19]. Therefore, our results saturate at 40 dB representing a perfect frame reconstruction. Figure 11 shows that the average PSNR values for FDSP with BP=0%, BP=100%, and Adaptive-BP are 35.6 dB, 39.8 dB, and 39.7 dB, respectively. Therefore, the video quality for FDSP with Adaptive-BP is for the most part identical to FDSP with BP=100% except for a dip in PSNR for frames 2566-2587 corresponding to the UDP packet loss shown in Figure 9b at substream 9. Visually this shows up as a glitch during video playback and lasts for 0.4 seconds. In contrast, the proposed Adaptive-BP more than makes up for slight reduction in PSNR by significantly reducing TCP buffering with just 2 instances of rebuffering and a total rebuffering time 1.1 seconds. In comparison, pure-UDP streaming is extremely lossy and yields an average PSNR of just 32.76 dB.

These results clearly show that FDSP with Adaptive-BP results in better QoE compared to FDSP with static BP values, pure-TCP, and pure-UDP based streaming.

## VII. Conclusion

This paper proposed the Adaptive-BP technique that dynamically adjusts the proportion of packets sent over TCP versus UDP for FDSP, as presented in [5][6]. The proposed method adaptively selects the BP parameter for each substream based on the estimated rebuffering time and UDP packet loss rate. For each substream the Adaptive-BP algorithm selects the BP value that satisfies the rebuffering time and the packet loss ratio thresholds. Our results show that the proposed method reduces both rebuffering time and packet loss ratio leading to a more favorable overall video streaming experience. As future work, FDSP with Adaptive-BP will be extended to include real time video streaming and further improve the accuracy of the estimation algorithm.

## References

[1] R. Pantos and W. May, "HTTP Live Streaming," Apr. 2014, iETF Draft, URL: https://developer.apple.com/streaming/ [Accessed: 2015-09-20].

[2] "Microsoft Smooth Streaming," Microsoft, [Accessed: 2015-09-20]. [Online]. Available: http://www.iis.net/downloads/microsoft/smooth-streaming

[3] T. Kim and M. H. Ammar, "Receiver Buffer Requirement for Video Streaming over TCP," in *Proceedings of Visual Communications and Image Processing Conference*, January 2006, pp. 422–431.

[4] X. Shen, A. Wonfor, R. Penty, and I. White, "Receiver playout buffer requirement for tcp video streaming in the presence of burst packet drops," in *London Communications Symposium*, 2009.

[5] J. Zhao, B. Lee, T.-W. Lee, C.-G. Kim, J.-K. Shin, and J. Cho, "Flexible dual tcp/udp streaming for h.264 hd video over wlans," in *Proc. of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC '13)*. New York, NY, USA: ACM, 2013, pp. 34:1–34:9.

[6] M. Sinky, A. Dhamodaran, B. Lee, and J. Zhao, "Analysis of H.264 bitstream prioritization for dual TCP/UDP streaming of HD video over WLANs," in *2015 IEEE 12th Consumer Communications and Networking Conference (CCNC 2015)*, Las Vegas, USA, Jan. 2015, pp. 576–581.

[7] S. Wenger, "H.264/AVC over IP," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 645–656, Jul. 2003.

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

[9] A. Nafaa, T. Taleb, and L. Murphy, "Forward error correction strategies for media streaming over wireless networks," *Communications Magazine, IEEE*, vol. 46, no. 1, pp. 72–79, Jan. 2008.

[10] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 2, pp. 16:1–16:22, May 2008.

[11] L. Borg, K. Streeter, and G. Eguchi, "HTTP Dynamic Streaming Specification Version 3.0 FINAL," Aug. 2013, URL: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/hds/pdfs/adobe-hds-specification.pdf[Accessed: 2015-09-20].

[12] M. van der Schaar and D. Turaga, "Cross-layer packetization and retransmission strategies for delay-sensitive wireless multimedia transmission," *IEEE Transactions on Multimedia*, vol. 9, no. 1, pp. 185–197, Jan. 2007.

[13] I. Ali, M. Fleury, S. Moiron, and M. Ghanbari, "Enhanced prioritization for video streaming over QoS-enabled wireless networks," in *Wireless Advanced (WiAd'11)*, Jun. 2011, pp. 268–272.

[14] Y.-K. Wang, R. Even, T. Kristensen, and R. Jesup, "RTP Payload Format for H.264 Video," RFC 6184 (Proposed Standard), Internet Engineering Task Force, May 2011.

[15] K. Ishibashi, M. Aida, and S.-i. Kuribayashi, "Estimating packet loss-rate by using delay information and combined with change-of-measure framework," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 7, Dec 2003, pp. 3878–3882.

[16] Qin Dai and Lehnert, R., "Impact of Packet Loss on the Perceived Video Quality," in *Evolving Internet (INTERNET), 2010 Second International Conference on*, Sept 2010, pp. 206–209.

[17] C. Lee, M. Kim, S. Hyun, S. Lee, B. Lee, and K. Lee, "OEFMON: An open evaluation framework for multimedia over networks," *Communications Magazine, IEEE*, vol. 49, no. 9, pp. 153–161, Sep. 2011.

[18] Conviva SanMateo California, Tech. Rep. Feb 2013, "The conviva viewer experience report," Conviva, URL: http://www.conviva.com/vxr-home/vxr2013/[Accessed: 2015-09-20].

[19] J. Gross, J. Klaue, H. Karl, and A. Wolisz, "Cross-layer optimization of OFDM transmission systems for mpeg-4 video streaming," *Computer Communications*, vol. 27, no. 11, pp. 1044 – 1055, Jul. 2004.

 ISBN: 978-1-61208-439-8 