Multimedia Systems (2020) 26:479–493
https://doi.org/10.1007/s00530-020-00653-w

REGULAR PAPER

Experimental study of QoE improvements towards adaptive HD video streaming using flexible dual TCP-UDP streaming protocol

Kevin Gatimu[1] · Arul Dhamodaran[1] · Taylor Johnson[1] · Ben Lee[1]

Received: 19 September 2018 / Accepted: 30 April 2020 / Published online: 2 June 2020

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

Keywords Low latency streaming · Hybrid protocol · FDSP · DASH · Adaptive video streaming · QoE

1 Introduction

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

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

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

The HAS model introduces a number of factors that influence viewers' Quality of Experience (QoE). These

Communicated by B. Li.

✉ Kevin Gatimu
gatimuk@eecs.oregonstate.edu

Arul Dhamodaran
dhamodaa@eecs.oregonstate.edu

Taylor Johnson
johnstay@eecs.oregonstate.edu

Ben Lee
benl@eecs.oregonstate.edu

[1] School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, OR 97331, USA

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

Transmission Control Protocol (TCP) is the underlying transport protocol of HAS, and it provides transport services such as reliable in-order delivery, congestion control, and flow control. As a result, applications which rely on TCP often experience high latency, and this adversely affects the HAS model as well. On the other hand, the *User Datagram Protocol* (UDP) is a low-latency alternative without the services provided by TCP. Our hybrid streaming protocol, called *Flexible Dual TCP-UDP Streaming Protocol* (FDSP), combines the reliability of TCP with the low latency of UDP through a simple application-layer combination, thus eliminating special network-layer modifications or additional protocols [6–9]. This is especially important for media content providers who need to deploy videos to heterogenous networks and diverse devices. Our initial simulation studies of FDSP have shown that it is effective in improving direct device-to-device (D2D) streaming in a wireless local area network [6]. The basic FDSP was then improved by adding *Bitstream Prioritization* (BP), where a percentage of more important elements of the H.264 bitstream were prioritized via TCP transmission [7, 8]. This was followed by a study using a client-server VoD testbed, which showed that FDSP-based streaming achieves lower latency and less packet loss than TCP-based and UDP-based streaming, respectively [9].

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

1. FDSP can sustain a particular bitrate version of video longer than TCP in congested networks with less packet loss and rebuffering. This can decrease the frequency of bitrate switches and increase average video bitrate.
2. The FDSP client buffer is more stable than in TCP-based streaming. This provides the client with a more reliable measure for assessing the available bandwidth and developing more accurate buffer-based adaptation algorithms [10]. This is in accordance with the DASH implementation guidelines, which emphasizes the importance of a rate adaptation algorithm to smooth out fluctuations in available bandwidth [11].

The rest of this paper is organized as follows. Section 2 provides a background of HAS, FDSP with BP, and UDP firewall traversal. Section 3 discusses the related work. Section 4 describes the experiment setup using a physical testbed. This is followed by a discussion of the results in Sect. 5. Finally, Sect. 6 concludes the paper and discusses possible future work.

2 Background

2.1 HAS

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

Bitrate adaptation algorithms can be broadly classified into three major categories: client-side, server-side, and network-level [16]. This paper will focus on client-side algorithms, which can be further classified into throughput-based, buffer-based, and hybrid [16]. In general, *throughput-based* methods select video bitrates according to bandwidth estimation while *buffer-based* methods do so based on a target client buffer occupancy. Hybrid methods are a combination of the two. There is a growing consensus on the greater

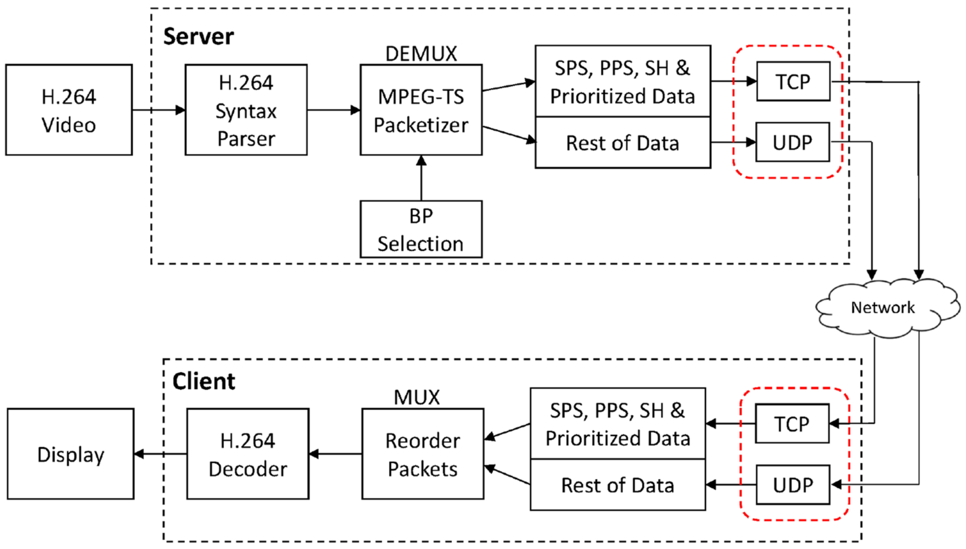


Fig. 1 Flexible dual TCP-UDP streaming protocol (FDSP) architecture [6]

importance of analyzing buffer occupancy compared to bandwidth estimation towards developing bitrate adaptation algorithms. For instance, Huang et al. demonstrated the ineffectiveness of bandwidth estimation [17], especially when there are competing flows, and proposed a buffer-based approach to bitrate adaptation [10]. Similarly, Spiteri et al. ignored bandwidth estimation in favor of buffer occupancy [18]. Furthermore, Yin et al. formulated an optimization model between buffer occupancy and bandwidth estimation, and found that their effectiveness for bitstream adaptation was limited by the latter [19]. Our study on FDSP-based streaming showed that it exhibits a more stable client buffer compared to TCP-based streaming. Therefore, FDSP provides a more reliable reference in the transport layer for designing better buffer-based bitrate adaptation algorithms. This is important, especially given the growing evidence of how unreliable bandwidth estimation can be.

2.2 FDSP overview

This section provides an overview of FDSP, including its architectural features and video streaming using substreams. For more details, see [6, 7] and [8]. FDSP is a hybrid streaming protocol that combines the reliability of TCP with the low latency characteristics of UDP. Figure 1 shows the FDSP architecture consisting of a server and a client.

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

When a stream is initiated, the FDSP server transmits the packets for the first 10-second substream. All the TCP packets for this substream ($T_1 \ldots T_n$) must be received (i.e., buffered) before playback begins. This startup delay (T_{init}) is low since only the TCP portion of the data is sent rather than the whole 10 seconds of video. To minimize rebuffering, the TCP packets for the next substream are sent at the same time as the UDP packets ($U_1 \ldots U_n$) for the current substream through a process called *substream overlapping* as illustrated in the transmission stream section of Fig. 2. In this particular example, note that the TCP packets for substream 2 are all transmitted together with UDP packets of substream 1. This is done before transmitting the UDP packets for substream 2. Substream overlapping is repeated throughout the duration of the stream. However, when playback for a particular substream is complete and the TCP packets for the upcoming substream are not yet all available, the client has to wait, thus

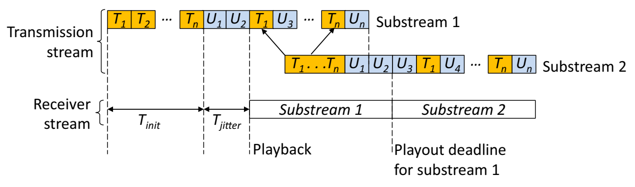

Transmission stream
T_1 T_2 ··· T_n U_1 U_2 T_1 U_3 ··· T_n U_n
Substream 1
$T_1 \ldots T_n$ U_1 U_2 U_3 T_1 U_4 ··· T_n U_n
Substream 2
Receiver stream
T_{init}
T_{jitter}
Substream 1
Substream 2
Playback
Playout deadline for substream 1

Fig. 2 Substream overlapping. Each packet is either UDP (U) or TCP (T), where the subscript represents the packet number within a substream

causing a rebuffering instance. The playout deadline for all subsequent packets is then incremented by the rebuffering time.

2.3 UDP firewall traversal

FDSP's TCP-UDP hybrid form is a useful and relevant technology for current innovations in HAS and video streaming transport in general. Even though HAS is primarily built on HTTP/TCP mainly due to HTTP's ability to traverse network address translators (NATs) or firewalls [20], UDP-based technologies are also critical to supplementing the latest HAS systems, e.g., as in CDN-P2P architectures (see Sect. 3 for more details on UDP-based streaming). As a result, there exist several UDP firewall traversal techniques.

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

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

3 Related work

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

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

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

Swaminathan et al. used HTTP chunked transfer encoding to disrupt the correlation between latency and segment duration, particularly in live streaming [26]. This was done by using partial HTTP responses rather than waiting for complete responses (i.e., full segments) to be generated by the server. Houze et al. also used HTTP chunked encoding, but to supplement an application layer multi-path TCP streaming scheme [27]. Here, video frames were subdivided in proportion to network path speeds and reassembled by the client, thereby reducing latency and rebuffering. However, HTTP transfer chunked encoding can lead to extra overhead due to the increased volume of HTTP transfers. This is especially the case in congested networks, where timeout issues can occur when complete HTTP responses are not assembled punctually [28]. Rather than requiring chunked encoding, FDSP is readily compatible with basic HTTP while still reducing latency and rebuffering. At the same time, HTTP chunked encoding is an orthogonal issue and could optionally be implemented on FDSP.

Alternatively, HTTP/2 provides server push mechanisms that allow the client receives multiple video segments per request instead of just a single segment [29–31]. This reduces the overall time needed for the client-server request-response mechanism, thus reducing latency. However, HTTP/2 is not as widely available as the more established HTTP/1.1. HTTP/2 only has 15% worldwide deployment and, at a current growth rate of 5% additional coverage every year, it has a long way to go before becoming a widely recognized standard [32]. However, FDSP is a simple application-layer combination of UDP and TCP, making it compatible with HTTP/1.1. FDSP's straightforward implementation also makes it compatible with HTTP/2.

UDP is well-suited for low-latency applications as it lacks the extra overhead necessary for TCP's features, such as flow control and reliable delivery. However, UDP's simplicity can result in packet loss, especially in congested networks. In addition, the lack of congestion control can lead to depletion of network resources due to bandwidth over-utilization, placing UDP out of favor compared to TCP in the broader Internet landscape. Nevertheless, UDP's low-latency offers an attractive option for contemporary HAS systems by supplementing CDNs with UDP-based P2P networks [33, 34]. This helps content providers lower deployment and maintenance costs [35]. At the same time, P2P networks improve live streaming latency by decreasing HTTP requests made to CDN servers [36, 37]. In fact, CDN caching can increase live streaming delay by 15-30 seconds [38]. CDN-P2P architectures have been commercialized for some time now by companies such as ChinaCache [33] and Akamai [34]. These hybrid architectures primarily rely on CDNs for HTTP-based retrieval of initial or critical video segments while using P2P networks for bandwidth relief or for retrieving future segments. Similarly, FDSP prioritizes the more important parts of the video bitstream via TCP while offloading the rest to UDP. Therefore, it can be integrated into a CDN-P2P-like framework, where the UDP portion of FDSP, in particular, can be reserved for a P2P network.

The work closest to ours is hybridization efforts at the transport layer to supplement the low latency and low overhead of UDP with TCP-like features [39–42]. Velten et al. initially proposed the *Reliable Data Protocol* [39], which was designed to be a minimal variation of TCP for bulk data transfer with simplified flow control, buffering, and connection management. Bova and Krivoruchka followed this with the improved *Reliable UDP* (RUDP) [40]. RUDP extends UDP mainly by making some features mandatory, e.g., packet retransmissions, in-order delivery, and flow control. However, it is not standardized and is primarily limited to specific tasks, e.g., Microsoft's proprietary version for its TV software, Mediaroom [41]. There are also several RUDP-like protocols, but they are mostly application-specific. For example, Floyd et al. proposed the *Datagram Congestion Control Protocol* (DCCP), which adds congestion control to streaming media but without reliable in-order delivery [42].

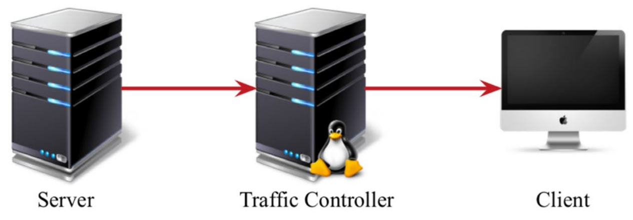


Fig. 3 Experiment testbed. The client consumes video provided by the server, while the traffic controller sets the level of network congestion

FDSP is also a hybrid streaming protocol, in that the services it provides fall somewhere between pure TCP and pure UDP. However, FDSP has an advantage over other hybrid methods because it simply uses the existing TCP and UDP protocols without any modification to either.

More recently, Google has done work on an experimental transport protocol built on UDP called *Quick UDP Internet Connections* (QUIC) [43]. Its main goal is to improve the performance of TCP-based applications by reducing latency and connection time. This is achieved by customizing UDP with encryption support, real-time bandwidth estimation, and bitstream compression. However, QUIC has been shown to have higher protocol overhead than TCP at low video bitrates [44]. On the other hand, FDSP demonstrates good performance across a wide range of video bitrates, including low ones.

4 Experiment setup

As in our previous work [9], the experimental testbed is shown in Fig. 3, which consists of a client-server pair and a traffic controller. The client and the server are each running *VLC Media Player* [45] on Mac OS X. The following modifications were made to integrate FDSP with BP into VLC:

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

The traffic controller, running on CentOS, connects the server to the client via a network bridge across interfaces `eth2` and `eth3`, respectively. The Linux *traffic control* (*tc*) utility is then used to perform traffic control on the network bridge, which, in turn, sets the available bandwidth. The *tc* configures the Linux kernel primarily through *queueing disciplines* (*qdiscs*). A *qdisc* is an interface between the kernel and a network interface, where packets are queued and released according to *tc* settings. These settings are then

Table 1 Network congestion settings for for *tc*

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

used to create different levels of network congestion. For example, a loss setting drops packets from the *qdisc* according to a specified percentage, while a delay setting prolongs the duration the packets spend in the *qdisc*.

This testbed provides a physical platform for two sets of experiments. The first is a fundamental video streaming comparison among the relevant protocols, i.e., FDSP, UDP, and TCP. This is used to corroborate our findings in favor of FDSP from our simulation-only device-to-device studies [6–8]. The second set of experiments are then used to explore the advantages of FDSP over TCP towards future integration into HAS. The rest of this section details each set of experiments, while the corresponding results are discussed in Sect. 5.

4.1 Basic comparison of FDSP, UDP and TCP

A summary of *tc* parameters used to compare the three protocols is shown in Table 1. The values chosen represent an array of Wide Area Network (WAN) scenarios that would typically plague Internet video streaming performance. The *Delay* setting, in conjunction with the other settings, was primarily used to generate five different levels of real-world Internet congestion [46]. The core network RTT latency[1] is about 30 ms within Europe, 45 ms within North America, and 90 ms for Trans-Atlantic routes [47]. However, edge networks introduce additional latency. Therefore, *Delay* ranging from 25 to 125 ms in increments of 25 ms was used for each of the two bridged interfaces (`eth2` and `eth3`), resulting in a total RTT latency of 50–250 ms in increments of 50 ms. This corresponds to five different levels of congestion dictated by RTT latency, i.e., 50 ms, 100 ms, 150 ms, 200 ms, and 250 ms. In addition, for each delay setting, the corresponding random *Jitter* value was set at 20% of the delay, while the *Duplicate* setting generates duplicate packets, e.g., due to retransmissions. The *Loss* setting causes a minimum ratio of packets to be randomly dropped by the network. The *Corrupt* setting introduces a random bit error in a specified percentage of the packets. Finally, the *Reorder* setting simulates multi-hop routing by further delaying a specified percentage of packets according to the *Delay* and *Jitter* settings. For brevity, the *Delay* setting will be used to represent the 6-tuple settings.

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

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

4.2 Comparison of FDSP and TCP in multi-bitrate streaming

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

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

The token bucket filter (`tbf`) is a packet queue model that releases tokens according the ratio between the `rate` parameter, i.e., the desired available bandwidth, and the kernel frequency. The `burst` parameter must be set to at least this ratio in bytes. In this example, the `rate` is 5 Mbps and the kernel frequency is 1000 Hz for our testbed, which yields a `burst` of 625 bytes. A packet at the head of the queue is transmitted once there are enough tokens corresponding to the packet size in bytes.

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

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

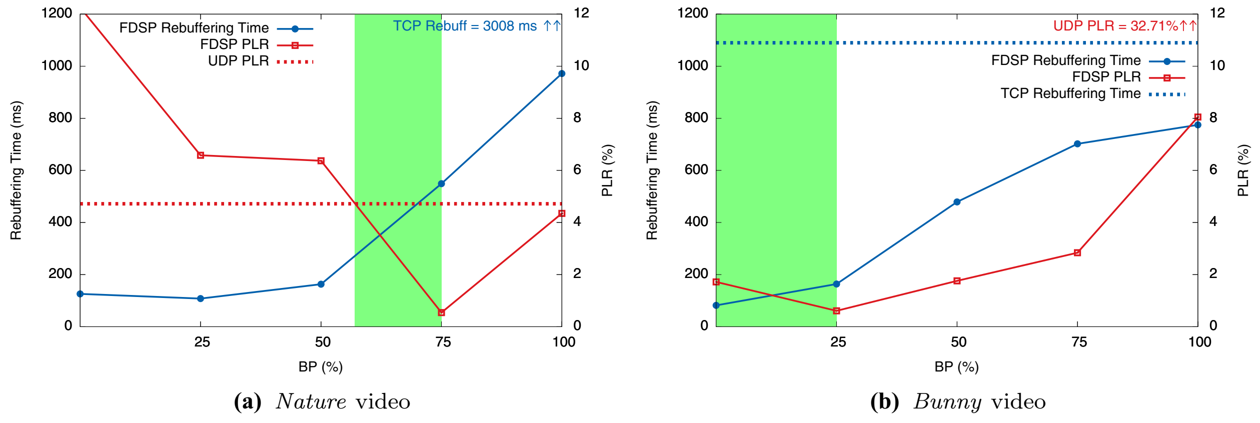


Fig. 4 Rebuffering time and packet loss ratio (PLR) for FDSP, TCP, and UDP at 100 ms delay green colour indicates optimal range (colour figure online)

which is similar to the recommended network profiles provided by the DASH Industry Forum Guidelines [48].

Each category is further subdivided into congested and non-congested settings. Although our results are primarily focused on congested settings, the experiments were also repeated for non-congested settings for completeness. In our prior simulation studies, congestion was set at the average bitrate of the video, which proved sufficient. However, in our physical testbed, this severely hampered streaming, particularly with the addition of the traffic control `latency` parameter, which specifically refers to network latency. Our experiments showed that available bandwidth set at 25% above the average video bitrate provided sufficient congestion for making useful comparisons between FDSP and TCP.

5 Results

This section discusses the results of our experiments. Sect. 5.1 analyzes the improvements of FDSP relative to both TCP and UDP in rebuffering and packet loss ratio (PLR) as BP increases from 0 to 100%. The corresponding tradeoffs are also discussed in detail. In summary, as BP increases, PLR decreases while rebuffering increases. Then, Sect. 5.2 shows why FDSP is more suitable than TCP for multi-bitrate streaming and, consequently, its potential as a transport protocol for improved adaptive streaming.

5.1 FDSP improvement over both UDP and TCP

Figure 4 shows an example of basic video streaming improvements of FDSP over TCP and UDP at 100 ms RTT delay for *Nature* and *Bunny*. Note that this and every other RTT delay setting is accompanied by corresponding *tc* parameters as described in Table 2 and Sect. 4.1. In general, FDSP rebuffering time is significantly lower than TCP rebuffering time even though it increases with BP. At the same time, PLR also decreases within a particular range of BP. The other levels of network congestion chosen for our experiments (i.e., 50 ms, 150 ms, 200 ms and 250 ms) show similar results.

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

5.1.1 FDSP Improvement over TCP in Rebuffering

Reducing both rebuffering time and the number of rebuffering instances is important towards improving the user's QoE. Figure 5 shows the total amount of rebuffering time and instances for the different levels of network congestion. For each congestion level, rebuffering for FDSP is shown with different values of BP as well for TCP. For instance, for *Nature* with 150 ms RTT delay, FDSP rebuffering time ranges from 108 ms to 1616 ms, compared to 9410 ms in TCP. In addition, the number of rebuffering instances ranges from 2 to 3 for FDSP compared to 7 for TCP. Meanwhile, for *Bunny* with 150 ms RTT delay, FDSP rebuffering time ranges from 92 ms to 1441 ms with 2 to 6 instances,

Table 2 Available bandwidth settings (Mbps) for comparing FDSP to TCP

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

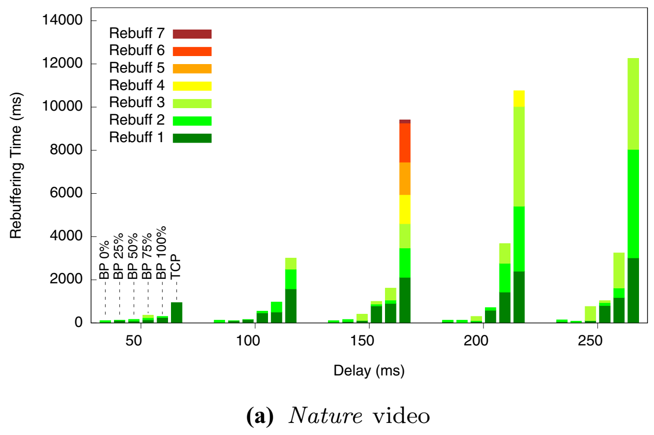


Fig. 5 Rebuffering for different levels of network congestion for FDSP-based streaming at different values of BP and TCP-based streaming

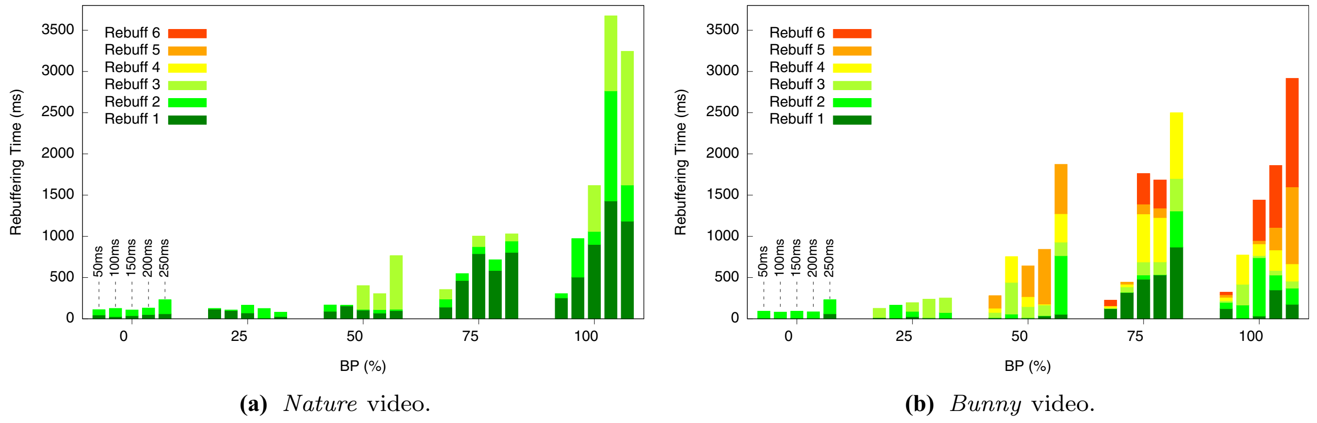


Fig. 6 An in-depth look at rebuffering time for different levels of network congestion at different values of BP. TCP has been omitted here as it has much higher rebuffering than FDSP

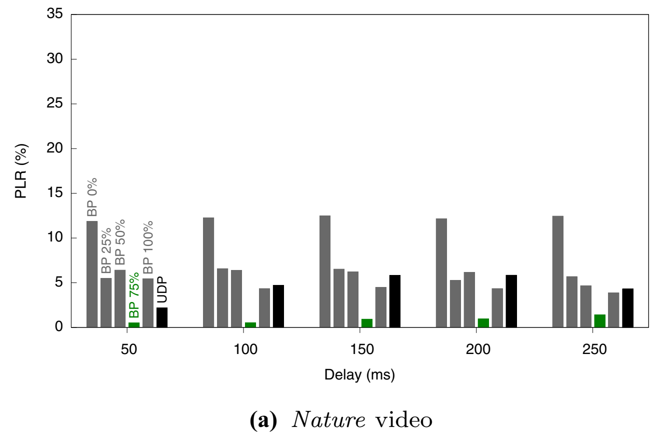
compared to 8764 ms with 5 instances for TCP. The rest of the rebuffering results for the two videos are summarized in Tables 3 and 4. Note that the first rebuffering instance (*Rebuff 1*) is the startup delay. As can be seen, FDSP exhibits lower startup delay than TCP for all BP values.

While FDSP is significantly better than TCP in terms of rebuffering, it is important to note that rebuffering does increase with BP. This is because higher BP values result in more TCP data, which increases the chance of retransmissions and thus rebuffering. A closer look at the behavior of just FDSP is illustrated by Fig. 6, which shows the rebuffering time and instances across the different network congestion levels for each BP value.

5.1.2 FDSP improvement over UDP in PLR

FDSP-based streaming results in not only less rebuffering, but better video quality by reducing PLR. Figure 7 shows the effect of BP on PLR across different levels of network congestion for both *Nature* and *Bunny*. For each congestion level, PLR is shown for FDSP with different values of BP as well as for UDP. PLR can be minimized by increasing BP, which leads to better video quality. For example, in *Nature*, the PLR for 50 ms RTT delay decreases from 9 to 0.32% as BP increases from 0 to 75%. Similarly, in *Bunny*, the PLR decreases from 1.19 to 0.51% as BP increases from 0 to 25%. In addition, this implies that there is an optimal range of BP operation based on the type of video as shown in Fig. 4. As BP increases, more packets are sent via TCP rather than UDP. This protects them from network-induced losses. Since PLR is due to lost UDP packets, the overall PLR decreases as BP increases from 0% through the optimal range. Further details are discussed in Sect. 5.1.3.

Figure 8 shows a sample of the visual improvement of FDSP-based streaming with 0% BP over UDP-based streaming in *Bunny*. The video frame in Fig. 8a is intact while the frame in Fig. 8b shows the effects of packet loss under UDP-based streaming. In such situations, the loss of just



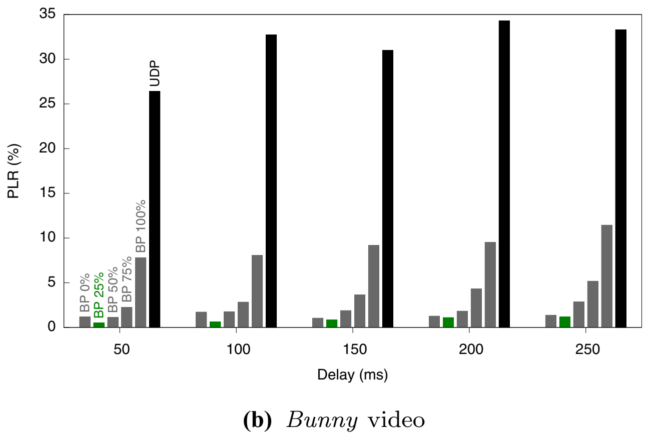
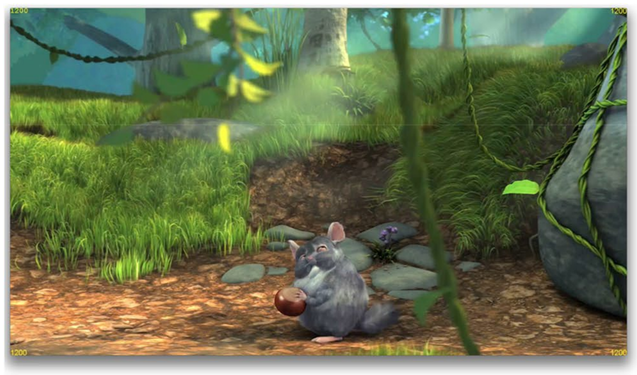


Fig. 7 PLR for FDSP-based streaming at different values of BP and UDP-based streaming for different levels of network congestion


(a) Basic FDSP (0% BP)

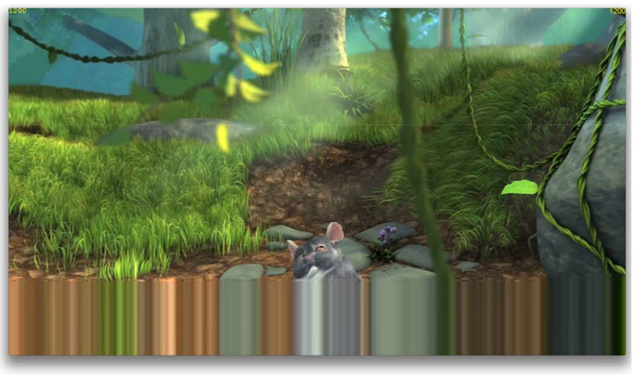

(b) UDP

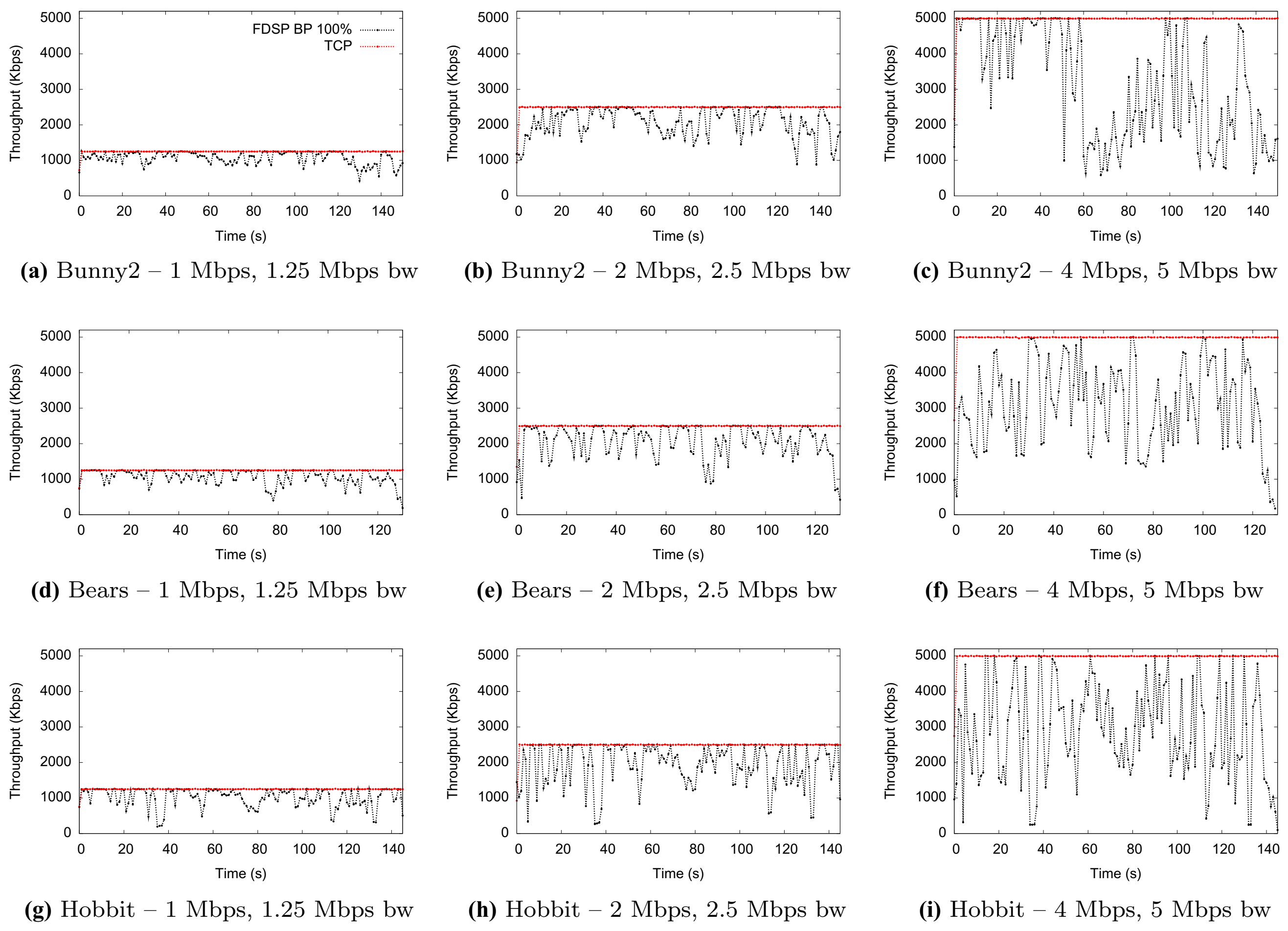
Fig. 8 Visual comparison between FDSP-based streaming and UDP-based streaming for *Bunny*

a slice header or the first few bytes of a slice renders the rest of the slice data useless to the decoder, even if properly received. This results in the decoder performing error concealment on the lost data as shown in slice 4 of Fig. 8b. On the other hand, FDSP-based streaming, even at 0% BP, protects at least the slice headers through TCP transmission, thus making any available slice data useful to the decoder. As a result, FDSP produces better quality video frames as shown in Fig. 8a.

5.1.3 Optimal range of BP

Since overall rebuffering time is significantly lower in FDSP than TCP across all BP values as shown in Fig. 4, there is an optimal range of BP values for decreasing PLR as shown. If BP surpasses the optimal range and becomes too high, the network can become saturated with TCP packets. This will cause more packets to be delayed, reordered, or lost when there is network congestion. The TCP packets are then more prone to retransmissions so as to guarantee in-order, reliable delivery. Meanwhile, the IP queue at the sender is filled with staged TCP and UDP packets. These additional TCP packets in the IP queue then cause subsequent UDP packets to be dropped. This is the cause of most of the PLR when BP becomes too high. Note that additional PLR also occurs when some UDP packets arrive at the client too late, past the decoder's playout deadline, and are thus also considered lost.

The frequency of I-frames can be used to categorize the type of video, and therefore determines the optimal range of BP. Videos such as *Bunny* are characterized by fast motion, many scene changes, and a corresponding higher number of I-frames. In fact, there are 37 I-frames in *Bunny* compared to just 5 in the low-motion *Nature*. UDP-based streaming PLR in low-motion videos is much lower than in high-motion videos. For instance, as shown in Fig. 7, *Nature* has 2.2–4.3% UDP-based streaming PLR across all congestion levels compared to 26.4–33.3% in *Bunny*. This is because *Nature*'s lower I-frame frequency reduces the likelihood of making network saturation worse.



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

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

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

Fig. 9 Throughput for FDSP-based streaming with 100% BP under static congested conditions (*bw* bandwidth)

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

5.2 FDSP improvement over TCP for multi-bitrate streaming

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

5.2.1 Lower bandwidth requirement

Figure 9 shows the throughput results for the three sets of videos streamed via FDSP with 100% BP in static congested scenarios. The throughput profiles for the BP values 0%, 25%, 50%, and 75% are very similar with minor proportional differences. FDSP-streaming required much less bandwidth than TCP-streaming in congested networks. For instance, the average throughput for *Hobbit* encoded at 4 Mbps was 2.87 Mbps at 0% BP and 2.91 Mbps at 100% BP. The average

Table 3 Rebuffering time and instances for *Bunny*

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

The percentage entries under protocol represent BP values for FDSP

throughput was lower than the average encoded bitrate for two reasons: (1) packet loss reduced the overall amount of data; and (2) rebuffering lengthened the playback time and, therefore, increased the time value used in the average throughput calculations.

In contrast, TCP-based streaming utilized practically all of the available bandwidth. For example, the TCP throughput for 4 Mbps *Bunny2* streamed on 5 Mbps of available bandwidth varied between ~ 4.9 and 5 Mbps. This is due to a couple of reasons. First, even when there is no congestion, TCP streaming requires bandwidth that is about twice the average video bitrate as discussed in Sect. 3. Second, TCP transmission requires significantly more bandwidth than FDSP due to retransmissions. For instance, when streaming the 4 Mbps version of *Bunny2* in 5 Mbps of available bandwidth, 94,477 KB of video data was transmitted and received for FDSP-based streaming compared to 155,441 KB for TCP-based streaming. This shows that FDSP would be more suitable for adaptive streaming in congested networks than TCP due to its lower bandwidth requirements. This would result in less bitrate switching, especially in congested networks. Furthermore, less bitrate down-switching would also raise the average bitrate.

FDSP-based streaming also exhibited very similar results for the dynamic congestion scenarios. However, the video completely stalled in the case of TCP-based streaming. This is because pure-TCP throughput is adversely affected by the available bandwidth oscillation due to the inability of TCP slow-start to achieve a sufficiently large congestion window, which results in persistently low throughput and increased retransmissions. In contrast, FDSP's UDP portion utilizes the oscillating bandwidth

Table 4 Rebuffering time and instances for *Nature*

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

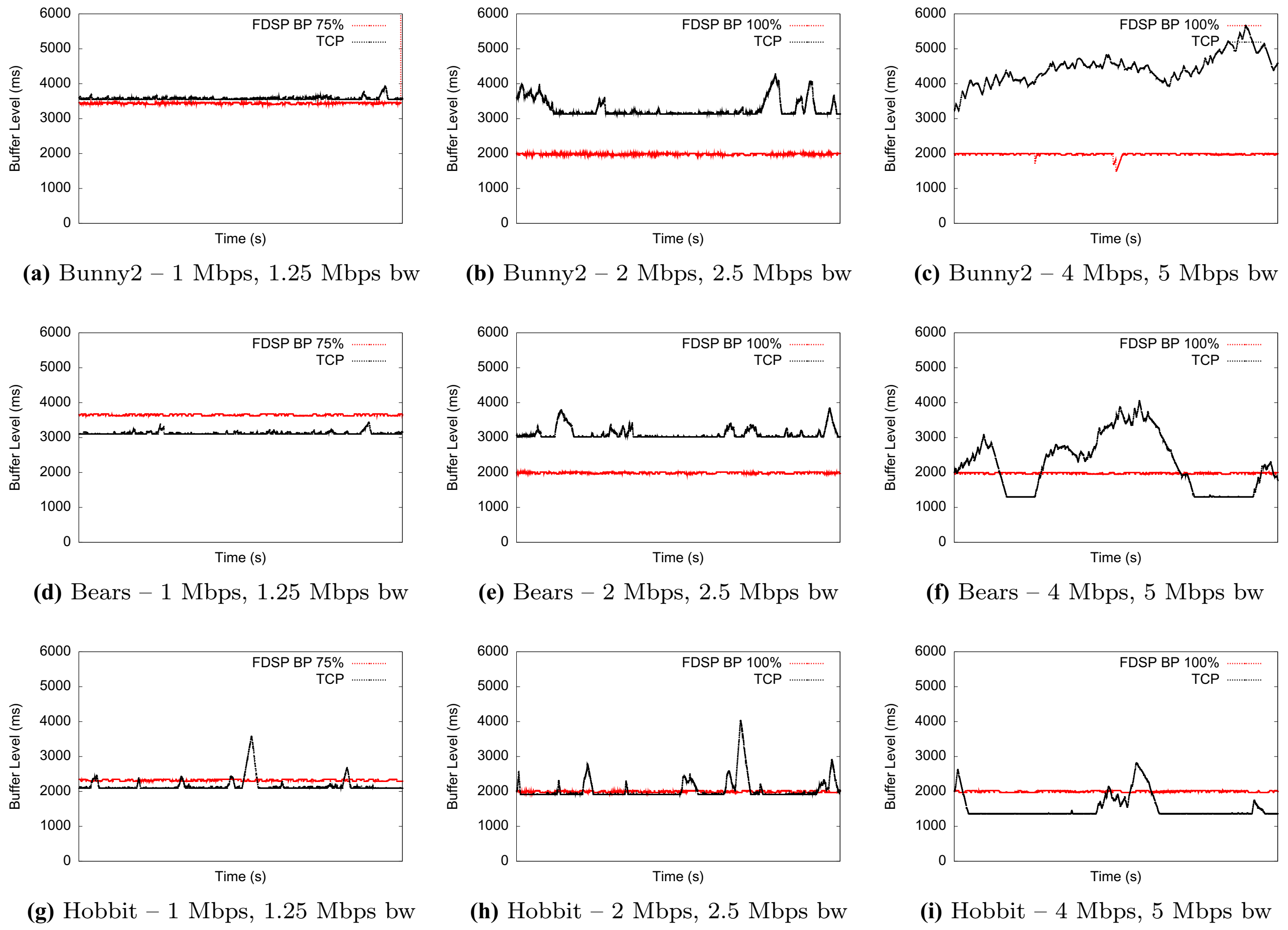
The percentage entries under protocol represent BP values for FDSP

more effectively. Finally, neither FDSP nor TCP was negatively affected by non-congested scenarios.

5.2.2 More stable client buffer occupancy

FDSP-based streaming showed more stable client buffer levels than TCP-based streaming. Figure 10 shows the client buffer occupancy for all the test videos and encoded bitrates streamed via FDSP and TCP at different static congestion levels. In general, FDSP-based streaming shows a consistent client buffer occupancy compared to more erratic results exhibited by TCP. This is the case for 0–100% BP for the videos encoded at 2 or 4 Mbps and 0–75% BP for the 1 Mbps videos. Since the client buffer behaves similarly for these BP ranges, the results for the highest respective BP values are shown in Fig. 10.

Among the three encoded bitrates, 1 Mbps videos exhibit the highest proportion of TCP data when streamed via FDSP. This is because most of the frames are so small that they have very little data beyond the slice headers. Consequently, protecting their slice headers via TCP transmits almost all of the corresponding frame data via TCP. As a result, FDSP-based streaming and the corresponding buffer occupancy at higher BP values (beyond 75%) greatly resembles TCP-based streaming for 1 Mbps videos. Therefore, the recommendation is to use the FDSP client buffer occupancy for buffer-based adaptation at the full BP range only when streaming at higher video bitrates. In this regard, lower video bitrates call for a more limited BP range. A scheme that maps a given video bitrate to the BP range that best leverages the buffer occupancy towards



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

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

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

Fig. 10 Client buffer occupancy for FDSP- and TCP-based streaming at 100% BP for 2 and 4 Mbps videos and 75% BP for 1 Mbps

buffer-based adaptation is beyond the scope of this experimental study and reserved for future work.

6 Conclusion and future work

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

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

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

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. VNI Global Fixed and Mobile Internet Traffic Forecasts. http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html. Accessed 3 Mar 2018
2. 4k Internet TV & Video to be Viewed by 1 in 10 US Residents, August 2016. https://www.juniperresearch.com/press/press-releases/4k-internet-tv-video-content-to-be-viewed-by-1-i. Accessed 3 Mar 2018
3. Jiang, J., Sekar, V., Zhang, H.: Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. IEEE ACM Trans. Netw. **22**(1), 326–340 (2014). https://doi.org/10.1109/TNET.2013.2291681
4. Krishnan, S.S., Sitaraman, R.K.: Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. IEEE ACM Trans. Netw. **21**(6), 2001–2014 (2013). https://doi.org/10.1109/TNET.2013.2281542. (ISSN 1063-6692)
5. What YOU Need to Know About HLS: Pros and Cons, January 2016. http://blog.red5pro.com/what-you-need-to-know-about-hls-pros-and-cons/. Accessed 3 Mar 2018
6. Zhao, J., Lee, B., Lee, T.-W., Kim, C.-G., Shin, J.-K., Cho, J.: Flexible Dual TCP/UDP Streaming for H.264 HD Video over WLANs. In: Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication (ICUIMC 2013), pp. 34:1–34:9. Kota Kinabalu, Malaysia (2013). https://doi.org/10.1145/2448556.2448590**(ISBN 978-1-4503-1958-4)**
7. Sinky, M., Dhamodaran, A., Lee, B., Zhao, J.: Analysis of H.264 bitstream prioritization for dual TCP/UDP streaming of HD video over WLANs. In: IEEE 12th Consumer Communications and Networking Conference (CCNC 2015), pp. 576–581. Las Vegas, USA (2015)
8. Dhamodaran, A., Sinky, M., Lee, B.: Adaptive bitstream prioritization for dual TCP/UDP streaming of HD video. In: The Tenth International Conference on Systems and Networks Communications (ICSNC 2015), pp. 35–40. Barcelona, Spain (2015)
9. Gatimu, K., Dhamodaran, A., Johnson, T., Lee, B.: Experimental study of low-latency HD VOD streaming using flexible dual TCP-UDP streaming protocol. In: 2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC), pp. 1–6 (2018). https://doi.org/10.1109/CCNC.2018.8319234
10. Huang, T.-Y., Johari, R., McKeown, N., Trunnell, M., Watson, M.: A buffer-based approach to rate adaptation: evidence from a large video streaming service. In: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, pp. 187–198. ACM, New York (2014). https://doi.org/10.1145/2619239.2626296**(ISBN 978-1-4503-2836-4)**
11. ISO/IEC TR 23009-3:2015—Information technology—dynamic adaptive streaming over HTTP (DASH)—Part 3: implementation guidelines. https://www.iso.org/standard/63562.html. Accessed 3 Mar 2018
12. Alex Z.: Smooth Streaming Technical Overview. http://www.iis.net/learn/media/on-demand-smooth-streaming/smooth-streaming-technical-overview. Accessed 3 Mar 2018
13. Adobe Systems. HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html. Accessed 3 Mar 2018
14. Apple Inc. HTTP Live Streaming Internet—Draft. https://tools.ietf.org/html/draft-pantos-http-live-streaming-19. Accessed 3 Mar 2018
15. ISO/IEC 23009-1:2012—Information technology—dynamic adaptive streaming over HTTP (DASH)—Part 1: media presentation description and segment formats. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623
16. Kua, J., Armitage, G., Branch, P.: A survey of rate adaptation techniques for dynamic adaptive streaming over http. IEEE Commun. Surv. Tut. **19**(3), 1842–1866 (2017). https://doi.org/10.1109/COMST.2017.2685630. (ISSN 1553-877X)
17. Huang, T.-Y., Handigol, N., Heller, B., McKeown, N., Johari, R.: Confused, timid, and unstable: picking a video streaming rate is hard. In: Proceedings of the 2012 Internet Measurement Conference, IMC '12, pp. 225–238. ACM, New York (2012). https://doi.org/10.1145/2398776.2398800**(ISBN 978-1-4503-1705-4)**
18. Spiteri, K., Urgaonkar, R., Sitaraman, R.K.: Bola: Near-optimal bitrate adaptation for online videos. In: IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016). https://doi.org/10.1109/INFOCOM.2016.7524428
19. Yin, X., Jindal, A., Sekar, V., Sinopoli, B.: A control-theoretic approach for dynamic adaptive video streaming over http. SIGCOMM Comput. Commun. Rev. **45**(4), 325–338 (2015). https://doi.org/10.1145/2829988.2787486. (ISSN 0146-4833)
20. Popa, L., Ghodsi, A., Stoica, I.: Http as the narrow waist of the future internet. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, pp. 6:1–6:6. ACM, New York (2010). https://doi.org/10.1145/1868447.1868453**(ISBN 978-1-4503-0409-2)**
21. Peer5, 2018. https://www.peer5.com/. Accessed 3 Mar 2018
22. Dutton, S.: WebRTC in the real world: STUN, TURN and signaling—HTML5 Rocks. https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
23. Wang, B., Kurose, J., Shenoy, P., Towsley, D.: Multimedia streaming via TCP: an analytic performance study. ACM Trans. Multimed. Comput. Commun. Appl. **4**(2), 16:1–16:22 (2008). https://doi.org/10.1145/1352012.1352020. (ISSN 1551-6857)
24. Aggarwal, A., Savage, S., Anderson, T.: Understanding the performance of TCP pacing. In: Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), vol. 3, pp. 1157–1165 (2000). https://doi.org/10.1109/INFCOM.2000.832483
25. Chakareski, J., Sasson, R., Eleftheriadis, A., Shapiro, O.: System and method for low delay, interactive communication using multiple TCP connections and scalable coding, (2014). http://www.google.com/patents/US8699522. U.S. Classification 370/474, 370/536, 375/240.05, 709/231; International Classification H04J3/24; Cooperative Classification H04L65/607, H04L47/32, H04L47/10, H04L47/193, H04L47/2416, H04L65/4015, H04L47/283, H04L65/80
26. Swaminathan, V., Wei, S.: Low latency live video streaming using HTTP chunked encoding. In: 2011 IEEE 13th International Workshop on Multimedia Signal Processing, pp. 1–6 (2011). https://doi.org/10.1109/MMSP.2011.6093825
27. Houzé, P., Mory, E., Texier, G., Simon, G.: Applicative-layer multipath for low-latency adaptive live streaming. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–7 (2016). https://doi.org/10.1109/ICC.2016.7511550
28. Hoffman, B.: Too chunky: performance and HTTP chunked encoding (2012). https://zoompf.com/blog/2012/05/too-chunky/. Accessed 3 Mar 2018
29. Wei, S., Swaminathan, V.: Low latency live video streaming over HTTP 2.0. In: Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, NOSSDAV '14, pp. 37:37–37:42. ACM, New York (2014). https://doi.org/10.1145/2578260.2578277**(ISBN 978-1-4503-2706-0)**
30. Cherif, W., Fablet, Y., Nassor, E., Taquet, J., Fujimori, Y.: DASH fast start using HTTP/2. In: Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV '15, pp. 25–30. ACM, New

York (2015). https://doi.org/10.1145/2736084.2736088**(ISBN 978-1-4503-3352-8)**
31. Huysegems, R., van der Hooft, J., Bostoen, T., Rondao Alface, P., Petrangeli, S., Wauters, T., De Turck, F.: HTTP/2-based methods to improve the live experience of adaptive streaming. In: Proceedings of the 23rd ACM International Conference on Multimedia, MM '15, pp. 541–550. ACM, New York (2015). https://doi.org/10.1145/2733373.2806264**(ISBN 978-1-4503-3459-4)**
32. Theedom, A.: Tracking HTTP/2 Adoption: Stagnation—DZone Web Dev (2016). https://dzone.com/articles/tracking-http2-adoption-stagnation. Accessed 3 Mar 2018
33. Liu, X., Yin, H., Lin, C.: A novel and high-quality measurement study of commercial CDN-P2p live streaming. In: 2009 WRI International Conference on Communications and Mobile Computing, vol. 3, pp. 325–329 (2009). https://doi.org/10.1109/CMC.2009.152
34. Lu, Z., Wang, Y., Yang, Y.R.: An analysis and comparison of CDN-P2p-hybrid content delivery system and model. J. Commun. (2012). https://doi.org/10.4304/jcm.7.3.232-245. http://www.jocm.us/index.php?m=content&c=index&a=show&catid=39&id=90**(ISSN 1796-2021)**
35. Xu, D., Kulkarni, S.S., Rosenberg, C., Chai, H.-K.: Analysis of a CDN-P2p hybrid architecture for cost-effective streaming media distribution. Multimed. Syst. **11**(4), 383–399 (2006). https://doi.org/10.1007/s00530-006-0015-3. (ISSN 0942-4962, 1432–1882)
36. Seyyedi, S.M.Y., Akbari, B.: Hybrid CDN-P2P architectures for live video streaming: comparative study of connected and unconnected meshes. In: 2011 International Symposium on Computer Networks and Distributed Systems (CNDS), pp. 175–180 (2011). https://doi.org/10.1109/CNDS.2011.5764567
37. Thi, ThuH, Tran, K., Jinsul, N.J.: Design and deployment of low-delay hybrid CDN-P2P architecture for live video streaming over the web. Wirel. Pers. Commun. **94**(3), 513–525 (2017). https://doi.org/10.1007/s11277-015-3144-1
38. Michaels, C.: HLS Latency Sucks, But Here's How to Fix It | Wowza (2017). https://www.wowza.com/blog/hls-latency-sucks-but-heres-how-to-fix-it. Accessed 3 Mar 2018
39. Velten, T., Hinden, R., Sax, J.: Reliable data protocol (1984). https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00. Accessed 3 Mar 2018
40. Bova, T., Krivoruchka, T.: Reliable UDP protocol. https://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00. Accessed 3 Mar 2018
41. Williams, J.: Microsoft TV test (2011). https://www.viavisolutions.com/en-us/literature/microsoft-tv-test-application-notes-en.pdf. Accessed 3 Mar 2018
42. Floyd, S., Handley, M., Kohler, E.: Datagram congestion control protocol (DCCP). https://tools.ietf.org/html/rfc4340. Accessed 3 Mar 2018
43. Wilk, A., Iyengar, J., Swett, I., Hamilton, R.: QUIC: a UDP-based secure and reliable transport for HTTP/2. https://tools.ietf.org/html/draft-hamilton-early-deployment-quic-00. Accessed 3 Mar 2018
44. Timmerer, C., Bertoni, A.: Advanced transport options for the dynamic adaptive streaming over HTTP. arXiv preprint arXiv:1606.00264, (2016)
45. VideoLAN. http://www.videolan.org/. Accessed 3 Mar 2018
46. Network Latency and Packet Loss Emulation @ Calomel.org. https://calomel.org/network_loss_emulation.html. Accessed 3 Mar 2018
47. IP Latency Statistics: (2017). http://www.verizonenterprise.com/about/network/latency/. Accessed 3 Mar 2018
48. DASH Industry Forum. Guidelines for Implementation: DASH-AVC/264 Test cases and Vectors (2014). https://dashif.org/wp-content/uploads/2016/06/DASH-AVC-264-Test-Vectors-v1.0.pdf. Accessed 3 Mar 2018