On the Delay Performance of Browser-based Interactive TCP Free-viewpoint Streaming

Tilak Varisetty, Markus Fidler
Institute of Communications Technology
Leibniz Universität Hannover
Email: firstname.lastname@ikt.uni-hannover.de

Matthias Ueberheide, Marcus Magnor
Computer Graphics Lab
Technische Universität Braunschweig
Email: lastname@cg.cs.tu-bs.de

***Abstract*—In free-viewpoint video arbitrary views of a scene or an object are rendered from a 3-dimensional scene representation that is obtained using multiple cameras or generated by computer graphics. The interactivity that is due to the viewpoint selection is particularly challenging in case of networked applications, where a server renders the scene from a viewpoint that is chosen by a remote client. Relying on widely-used standard browser-based video streaming technology, data transport is performed by the Transmission Control Protocol (TCP), implying an anticipated risk of potentially large delays. The magnitude, frequency, and origin of such delays are the focus of this work. To investigate the tail distribution of the delays, we use a controlled testbed environment and instrument the entire video streaming chain from the server-side renderer to the display at the client using various measurement points. We identify three major sources of delays: the video coders, the protocol stack, and the network. We investigate the causes of these delays and show a strong impact of network parameters, such as round-trip time and packet loss probability, on protocol stack delays. While stack delays can significantly exceed network delays, we find that stack delays can be reduced effectively by adapting the parameters of the video encoder.**

I. Introduction

Multiview video arises in many situations where a scene is captured simultaneously by multiple cameras from different viewpoints. The cameras can be configured as specific camera arrays, or they can be naturally located at various positions, e.g., cameras in a sports stadium or the cameras of mobile phones that may be used to record a public event. Applications of multiview video include immersive telepresence systems, 3-dimensional stereoscopic films, or free-viewpoint television, where the viewer can freely navigate the viewpoint [1]. The selected viewpoint may either coincide with a given camera position or otherwise it may be rendered using the views of nearby cameras. The rendering of a scene or an object from a given viewpoint is also performed in many other application areas of computer graphics such as gaming or Computer Aided Design (CAD).

In a video streaming application a server encodes the video and transmits it to one or more clients for simultaneous reproduction. Due to varying network latencies referred to as delay jitter, the client first stores the received data in a de-jitter buffer, and then displays it from the buffer after a defined playout delay that considers, e.g., a certain quantile of the network latencies.

This work was supported in part by the European Research Council (ERC) under StG 306644.

ISBN 978-3-903176-08-9 © 2018 IFIP

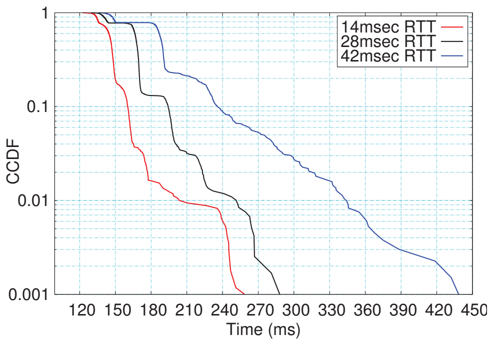


Fig. 1. End-to-end delay of interactive TCP streaming of a free-viewpoint application with server-side rendering. Different network RTTs of 14, 28, and 42 ms are evaluated. The packet loss rate is 1%. In the distribution tail, delays exceed the RTT by an order of magnitude.

Streaming of multiview video is particularly challenging due to the high bandwidth requirements when transmitting multiple video streams [2]. Compression techniques such as Multiview Video Coding (MVC) have been developed that use prediction to take advantage of the temporal correlation within the streams of the individual views (intra-view prediction) as well as the spatial correlation between different views (inter-view prediction).

Providing the entire set of all views to a client enables non-interactive multiview video streaming, where viewpoint selection or rendering of the viewpoint can be performed locally, i.e., by the client. A significant reduction of the data rate can be achieved, however, if only a selected subset of the available views needs to be transmitted. This is done in interactive streaming that uses a control channel from the client to the server to notify the server about the client's viewpoint selection [2]. Using this information, the server may either transmit a range of (potentially) relevant views, so that selection or rendering of the viewpoint remains with the client, or alternatively the server may perform the rendering itself and transmit only the single selected view [3]. Server-side rendering has the additional benefit that only a view but not the source data, for example a CAD model, need to be disclosed.

The reduction in bandwidth that is achieved by interactive streaming entails, however, more stringent latency requirements. Given a subset of the views, the client's viewpoint navigation is constrained by the boundaries of this subset [4], [5]. To prevent the client from selecting a viewpoint outside of this subset, the server has to continuously keep track of the viewpoint or even make a prediction of future viewpoints to adapt the range of views that it provides accordingly [5]–[7]. In case of server-side rendering, client and server have to interact in real-time, as the client's selection of the viewpoint has to be considered immediately by the server.

To explore the feasibility of interactive free-viewpoint streaming we investigate the delay performance of a reference implementation [8] in a controlled network testbed. We implement a server-side application that takes a 3-dimensional model to render selected viewpoints without disclosing the source data. We employ open source video coders and streaming servers and consider a browser-based implementation of the client that uses only standard HTML5 streaming without requiring any additional plugins. As HTML5 streaming is an emerging technology, the availability is still limited by constraints such as codec, browser, operating system, and type of video streaming [9]. The choice of HTML5 streaming also implies the use of the Transmission Control Protocol (TCP).

Fig. 1 shows an example of the Complementary Cumulative Distribution Function (CCDF) of the end-to-end delay of video frames from the server-side renderer to the display at the client. The end-to-end delays comprise the time needed for encoding and decoding of the video as well as the transmission of the video frames over the network. The measurements are performed using different network configurations with Round Trip Times (RTTs) of 14, 28, and 42 ms. Other parameters have not been changed and the packet loss rate is 1% in all experiments. The effect of the RTT is clearly visible in the upper part of Fig. 1, where the curves are spaced 14 ms apart from each other, as expected. A much stronger impact, that exceeds the RTT by an order of magnitude, is visible in the tail distribution of the end-to-end delays.

This paper reports the results of an extensive measurement study. We investigate the occurrence of large tail delays, show their causes and how they may be circumvented. To identify where delays occur, we instrument the video encoding and transmission chain and perform logging at various measurement points. Given that video frames are segmented into several packets, i.e., TCP segments, we implement a frame logging mechanism that identifies video frame boundaries in a packet stream with packet loss and retransmissions to be able to measure the delivery of entire video frames.

The remainder of the paper is structured as follows. In Sec. II we discuss related works on multiview and free-viewpoint streaming and the performance of TCP streaming. We show our experimental setup in Sec. III. Our measurement results obtained from the network experiments are presented in Sec. IV and Sec. V, where we investigate the impact of network and video parameters, respectively. Brief conclusions are provided in Sec. VI.

II. Related Work

We first discuss related works in multiview and free-viewpoint streaming and focus on the streaming performance of TCP afterwards.

A. Multiview and Free-viewpoint Video Streaming

Specific to multiview video streaming systems are multiple cameras, e.g., a camera array, that capture a scene from different viewpoints. The content that is generated by the cameras is encoded and streamed by a server over a network to one or more clients that can choose the viewpoint individually from the set of views that are available. Using techniques from computer graphics, the client may also render arbitrary new views, thus enabling a free-viewpoint navigation. Significant work has been dedicated to the quality and the complexity of the view synthesis. For an introduction to multiview video streaming see [2] and to free-viewpoint video [1].

One of the main challenges in multiview video streaming is the bandwidth that is required to transmit a potentially large set of views to the client. Various techniques have been developed to reduce the amount of data that is needed. In source coding, considerable works have taken advantage of the spatial correlation of the individual views to achieve a higher compression rate. A prominent example is the H.264/MPEG-4 extension MVC [10] that makes efficient use of temporal as well as spatial prediction. Using MVC, all views can be jointly encoded and transmitted, so that the client can choose the viewpoint without further interaction with the server.

Interactive multiview streaming, on the other hand, uses a control channel from the client to the server to report the viewpoint that is chosen by the client [2]. Using this information, the server can transmit only the selected view to the client to save bandwidth. Switching to a different view implies a random access that can be supported by insertion of intracoded frames. Since intracoded frames achieve less compression gain, more efficient frame structures, so-called merge frames, that enable view switching at defined intervals T, are presented in [11].

A concern with interactive multiview streaming is the view switching delay, that occurs if the user switches to a new view that is not streamed currently. In addition to the view switching interval T that can be small, i.e., in the order of a few frames [5], the network RTT of up to hundreds of milliseconds may have a considerable impact on the view switching delay [5], [12]. An approach to avoiding view switching delays is transmitting additional views, possibly with a higher compression [7], that are likely to be requested by the client within one RTT [4], [5], [7]. This creates a general tradeoff between bandwidth and latency [12] and requires a good anticipation of the viewpoint navigation of the user [5], [6]. Similar aspects concern free-viewpoint video streaming, where rendering can be performed at the client or at the server, requiring either sufficient bandwidth to transmit multiple views or small RTTs, respectively [3].

The importance of the RTT for view switching in interactive video streaming is emphasized in [2]–[5], [12]–[14]. While a number of works use an assumption of a deterministic RTT [4], [5], [7], mostly as a constant to determine a range of viewpoints that need to be prefetched, real networks exhibit large delay variations, as observed, e.g., in experiments in [12]. Further, recent works employ Dynamic Adaptive Streaming over HTTP (DASH) and TCP for free-viewpoint streaming with client-side rendering [14] and for multiview streaming [13], where delays are observed that exceed the RTT by orders of magnitude. For an example, [13] implements a number of techniques like buffer control, server push schemes, and parallel streaming to reduce the average view switching delay from 3.2 s to 380 ms given a network RTT of 4 ms.

B. TCP Streaming Performance

Despite the fact that TCP may cause large delays, it has become popular for streaming for other reasons, e.g., to circumvent firewalls. Extensive literature is available on the performance of TCP in multimedia streaming and specifically DASH. Relevant performance measures include the throughput and different types of delays that impact the users' Quality of Experience [15].

A major source of difficulty in TCP streaming is the variability of TCP's throughput. A common approach is to adapt the data rate of the encoded video to react to fluctuations of the available network bandwidth [16]. To reduce delays, [17] improves the adaptation based on TCP throughput predictions. To what extent the achievable TCP throughput can be utilized at all is investigated using an analytical model in [18]. An important conclusion is that good streaming performance is attained when the achievable TCP throughput is at least twice the video data rate.

Regarding TCP delays, different types of delays have to be distinguished. While there exist numerous studies that investigate network delays of TCP packets, fewer works have focused on TCP protocol delays [19]–[25]. TCP protocol delays are defined as the time difference from a write on the sender side socket to the corresponding read on the receiver side socket [19]. They comprise network delays plus potential transport layer queueing delays at the sender and at the receiver [23].

The works [19]–[22] present Cumulative Distribution Functions (CDFs) of protocol delays that for certain relevant network parameters show a long distribution tail due to heavy sender-side buffering. The authors of [19], [20] conclude that large tail delays may be avoided by reducing the sender's socket buffer size. Based on a testbed measurement study, [21] develops a parametric model of the CCDF of TCP protocol delays. The CCDFs show a characteristic exponential tail decay that depends on the relation of the average TCP throughput and the source data rate. The importance of this relation was already discussed above as it is also observed in [18].

An analytical model of TCP delays is derived for Constant Bit Rate (CBR) sources and the TCP version NewReno in [23]. The model gives working regions, i.e., RTTs and packet loss rates, that provide acceptable delay performance for streaming applications. The work [22] estimates service

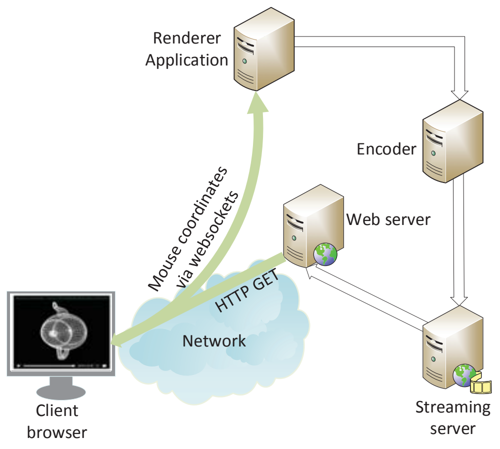


Fig. 2. System overview. The renderer application generates images at a fixed frame rate using the viewpoint that is selected by the mouse coordinates of the client. The images are encoded and streamed to the client browser by a webserver using HTTP and TCP.

curves of different TCP versions and shows how the different algorithms for adaptation of the Congestion Window (CWND) affect the video streaming performance. Queueing models of TCP's finite state machine are used to analyze its performance in [26], [27].

III. Experimental Setup

In today's Internet, TCP became a de facto standard for video streaming that has recently been adopted for more demanding applications such as multi-view [13] and free-viewpoint video streaming [14]. While [13], [14] use DASH with view switching delays of several hundreds of milliseconds up to seconds and client-side free-viewpoint rendering [14], we consider an interactive free-viewpoint video streaming system with server-side rendering that has significantly more demanding delay requirements. Like [13], [14] we implement a browser-based solution that also uses TCP but not DASH for streaming. In this section, we first give a brief overview of our implementation of free-viewpoint video streaming [8]. We then introduce our Emulab testbed configuration that we use for experimentation, and give details on our instrumentation of the application and network that enables us to distinguish different causes of delays.

A. Free-viewpoint Video Application

An overview of the system that we implemented is given in Fig. 2. The setup uses a client-server model where the server executes the necessary software for rendering, encoding, streaming, and hosting of the video. The view that is generated by the server-side renderer application is selected by the client.

The renderer application is programmed using OpenGL libraries. It uses libwebsockets on a specified TCP port to receive the coordinates of the viewpoint, i.e., the mouse coordinates, from the client. The renderer application periodically creates new images based on the selected viewpoint. Rendering is performed at a configurable rate of up to 50 frames per second (fps).

TABLE I
SOFTWARE VERSIONS.

Software	Version No.
Ubuntu	12.04.5 LTS
ffmpeg, ffserver	2.1.git
lighttpd	1.4.28
OpenGL	2.1 Mesa 8.0.4
libwebsocket	1.0.8
libvpx	1-3.0

Whenever the renderer creates a new image, it is encoded by the open source software ffmpeg using the libvpx video codec. The encoded image is streamed by ffserver and made available to the client as an HTML5 file by a lighttpd webserver. The client sends an HTTP GET request to receive the HTML5 stream from the webserver. Streaming is performed via TCP, specifically the TCP version Cubic with selective acknowledgements (SACK) option. The client runs Google's Chrome browser to play the video. As our setup only uses standard HTML5 streaming technology, the client's browser does not require any plugins.

For the purpose of the following measurements, we have configured the video encoder to produce constant-sized frames, specifically intracoded frames, to ease the interpretation of the results by avoiding delay variations that are due to the size of the frames. The size of video frames is approximately 12.6 kByte if not specified otherwise.

Tab. I summarizes the versions of the software used. Further details on our implementation can also be found in [8].

B. Network Testbed Configuration

We use the Emulab installation at our institute to evaluate the performance of our free-viewpoint video streaming application. Emulab[1] is a well-known framework for network emulation that can configure arbitrary network topologies consisting of nodes, i.e., hosts and routers, and links for controlled experimentation. Each node is put into effect by a physical machine that is booted with a defined operating system to act either as a router or as a host that runs the intended network application.

The machines have several network interfaces for experimentation, in our case a minimum of four 1 Gbps Ethernet interfaces, that are connected to a central switch. The switch creates Virtual LANs (VLANs) between the nodes to form the desired topology. Link parameters such as capacity, delay, and packet loss are emulated by the system using additional nodes, e.g., a link with a fixed delay comprises two VLAN links connected to a delay node that forwards packets only after the defined amount of time. The ipfw utility on freebsd is used for this purpose[2]. Capacity limits and packet loss are emulated in the same way. The machines have additional interfaces that are connected to a separate control network that is used to execute and monitor the experiments and to synchronize the clocks of the different machines using the Network Time Protocol (NTP) and the institute's NTP server.

[1]https://www.emulab.net/

[2]https://www.freebsd.org/cgi/man.cgi?ipfw(8)

TABLE II
UTILIZATION WITH RESPECT TO THE GREEDY TCP THROUGHPUT FOR A VIDEO FRAME RATE OF 10 FPS, DIFFERENT RTTS, AND LOSS RATES.

Loss rate (%)	RTT (ms)	Throughput (Mbps)	Utilization (%)
0.5	28	5.03	20.0
1	14	7.29	13.8
1	28	3.57	28.2
1	42	2.58	39.0
2	28	2.57	39.1

In our experiments, we use 1 Gbps Ethernet links to connect the video client and server. The Maximum Transmission Unit (MTU) of the Ethernet is 1500 Byte such that the TCP Maximum Segment Size (MSS) is 1460 Byte and video frames of 12.6 kByte size result in 9 TCP segments. The central link is configured to emulate a wide area network with RTTs of 14, 28, and 42 ms, respectively. In addition, the link has independent Bernoulli random packet losses of 0.5, 1, and 2%, respectively. Given the link parameters, the throughput of a TCP connection is limited by TCP's congestion control algorithm. We measured the throughput that is achieved by a greedy TCP Cubic source for these network parameters using iperf[3]. The results are detailed in Tab. II. We define the utilization of the TCP connection by the video source as the quotient of the video data rate and the greedy TCP throughput. The greedy TCP throughput is the maximal throughput of the TCP connection, that is achieved in case of a greedy data source. Tab. II gives the utilizations for a frame rate of 10 fps. The utilization for other rates of x fps follows by multiplication with $x/10$.

C. Measurement Points and Delays

To identify and evaluate the causes of large end-to-end delays, as observed in Fig. 1, we instrument the video application and the network using a number of Measurement Points (MP) as shown in Fig. 3. The trace files that are recorded contain tuples of timestamp and value, where the value is either a unique video frame identifier or an entire TCP segment, depending on whether the MP is in the application or in the network.

In detail, MP-A denotes the timestamp after an image has been created by the renderer application glrender. The image is written to a pipe file and MP-B denotes the time when ffmpeg fetches the image from the pipe. After encoding, ffmpeg writes the corresponding video frame to a localhost socket, that is MP-C. The frame is picked up from the localhost by ffserver at MP-D and written to the TCP socket where MP-E is the timestamp before the TCP send call. MP-F denotes the time when the first TCP segment of the video frame is transmitted on the network and MP-G is the time when the first TCP segment of the video frame arrived at the client. Once all TCP segments of the frame are received, the frame is delivered to the browser for reproduction at MP-H. Since TCP delivers data in order, the delivery of a complete frame is delayed if one of the preceding frames is not yet complete.

[3]http://software.es.net/iperf/

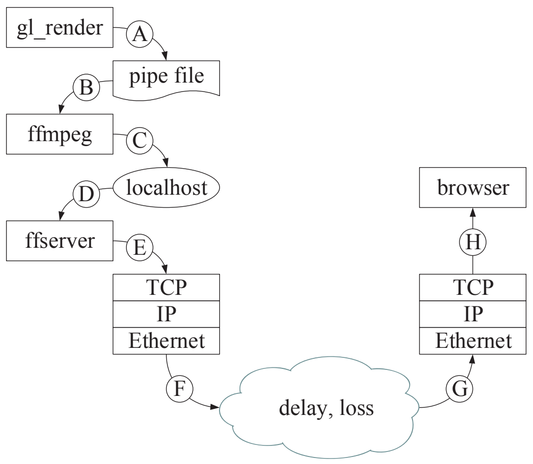


Fig. 3. The streaming pipeline is instrumented using measurement points A to H. We distinguish coding delays (A→C and D→E), stack delays (C→D and E→F), and transfer delays (F→H).

The logging framework is implemented in the source code of the software of the glrender application, ffmpeg, and ffserver. The glrender application generates a video frame number which is unique for every frame. It is mapped to the ffmpeg input and used for logging from MP-A up to MP-E. After MP-D, the ffserver assigns a unique Presentation Time Stamp (PTS) to the header of each video frame that is transmitted to the client. We note the correspondence of frame number and PTS at MP-E and use the PTS for identification of frames up to MP-H. Since TCP divides the video frames into smaller segments, the TCP segments are captured at MP-F and MP-G in the network using libpcap, respectively, the wireshark software[4]. We postprocess the network trace files to identify the video frame boundaries using the TCP sequence number and the PTS from the recorded TCP segments.

The different types of delays that each video frame incurs in the streaming pipeline from the rendering application to the client browser can be classified as follows:

Coding delays: The coding delays comprise the loading time of the image from the pipe (A→B), FFM encoder delays (B→C), and delays due to packaging into the FFM container format (D→E). In our measurements, the average delay between A→B is 18 ms, and B→C is 14 ms, respectively. The delay between D→E is dependent on the frame rate as we observed that ffserver generally buffers one video frame to send the current frame on the TCP socket. Hence, the delay is 100, 50, and 33 ms for 10, 20, and 30 fps, respectively. The coding delays observed in the experiments are not dependent on the network conditions.

Stack delays: Delays in the sender's protocol stack occur whenever the transmission is throttled by TCP congestion control so that frames have to wait for transmission in the sender's TCP stack (E→F), specifically in the socket buffer. In our experiments, the socket buffer is configured to have a size of 3 MByte. While the authors of [19], [20] argued that delays in the sender's TCP stack can be avoided by reducing the socket buffer size, we note that this approach shifts the problem to the next higher entity that would have to adapt accordingly. Precisely, if the socket buffer is full, the write call to the TCP socket by ffserver (E) blocks so that ffserver cannot fetch further frames from the localhost interface causing additional delays there (C→D). In the following evaluation we will generally show the combined stack delays of each frame (C→D plus E→F).

Transfer delays: The transfer delay of a video frame comprises the time to transmit all packets of the frame via the network (F→G). In case of packet loss, retransmissions are required in addition that can cause additional waiting times at the receiver until all packets are received in sequence and the frame can be delivered to the browser (G→H). In the evaluation we will show the entire transfer delay until the last TCP segment of the current video frame is delivered. (F→H).

IV. Impact of Network Delays and Loss

We conducted a large number of experiments with different network configurations and parameter sets. For clarity of exposition, we report experiments with selected parameter sets that enable us to separate and identify certain effects most clearly. In our evaluation, we first consider the impact of the network delay, specifically different RTTs of 14, 28, and 42 ms, on the transfer and stack delays experienced by the video frames. The network discards packets randomly to realize a loss rate of 1%. Different packet loss rates of 0.5% and 2% are considered afterwards. The frame rate is 10 fps and the frame size of 12.6 kByte corresponds to 9 TCP segments. The video bit rate at 10 fps is about 1 Mbps, i.e., compared to the greedy TCP throughput reported in Tab. II the utilization is moderate in all cases with a maximum of 39.1%.

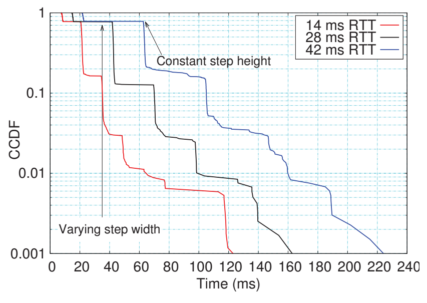
A. RTT-induced Delays

The network RTT has an obvious impact on the transfer delay as it takes at least RTT/2 to deliver a video frame to the client. Frequently, the transfer delay is, however, larger as not all packets of a frame may be transmitted at once due to TCP congestion control. Further, packets may require retransmission in the case of packet loss. In addition, the RTT may result in stack delays and blocking of the sender's TCP socket.

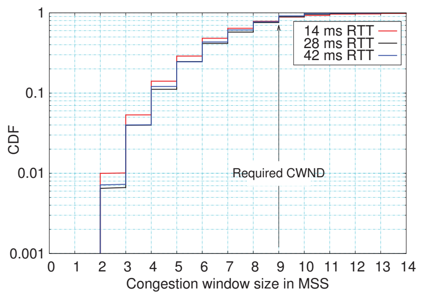
1) Per-frame Transfer Delays: In Fig. 4(a), we show the dependence of the CCDF of the transfer delay (F→H) on the RTT. We notice that all curves show the same stepped trend, where the first step at 7, 14, and 21 ms, respectively, corresponds to RTT/2 and the following steps have a width of 14, 28, and 42 ms, respectively, corresponding to the RTT. The step height, on the other hand, shows little influence of the RTT and is almost identical for the first steps.

The behavior is explained by the CDF of the CWND that is shown in Fig. 4(b). First, we notice that the CDF of the CWND exhibits no significant influence of the RTT. As the CDF indicates the probability that the CWND does not exceed a given value, we see that a CWND of less than 9 MSS, that

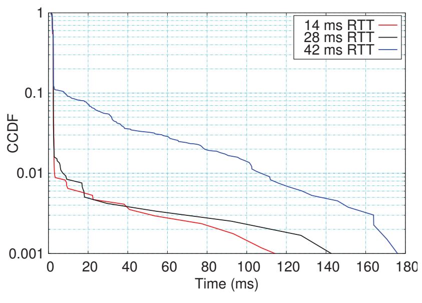
[4]https://www.wireshark.org/



(a) Transfer delay



(b) CWND



(c) Stack delay

Fig. 4. Impact of the RTT. The packet loss rate is 1%. The CCDF of the transfer delay exhibits first a minimum delay of RTT/2 followed by steps of a width of RTT. The CDF of the CWND shows only a marginal influence of the RTT. A CWND of at least 9 MSS permits sending a video frame at once. The probability that the CWND falls below 9 MSS corresponds to the probability to see transfer delays of more than RTT/2. Stack delays are observed when the transfer of a frame takes longer than 100 ms so that the next frame, at a frame rate of 10 fps, has to wait for transmission.

is the minimum CWND that is required to be able to transmit a video frame at once without waiting for acknowledgements, occurs with probability 0.77. In this case only a part of the video frame, i.e., CWND packets, can be transmitted before the transmission is paused. The transmission is resumed after one RTT when the first acknowledgements appear at the sender. We observe that the probability of 0.77 corresponds to the first step of a width of RTT in Fig. 4(a). The following steps can be explained in a similar way, e.g., in case of a CWND of less than 5 MSS it takes another round of one RTT. In addition, retransmissions that also take one RTT start to have an influence. In the given case of 9 TCP segments per frame and 1% packet loss, the transfer of a frame requires retransmission of one or more packets with a probability of almost 0.09.

We conclude that the RTT has only a minor effect on the CDF of the CWND in our experiments. Hence, the probabilities of observing transfer delays of several RTTs are similar irrespective of the RTT. In contrast, the magnitude of the transfer delays grows linearly with the RTT.

2) Stack Delays and Blocking: If the transfer of one or more video frames is not completed when the next video frame is ready for transmission, additional buffering applies at the sender resulting in stack delays and possibly blocking of the sender's TCP socket. Given the frame rate of 10 fps, this applies if video frames do not complete transfer within 100 ms. From Fig. 4(a), we find that transfer delays exceed 100 ms with a probability of slightly less or more than 0.01 in case of an RTT of 14 and 28 ms, and about 0.1 in case of an RTT of 42 ms. The CCDFs of the stack delays (C$\rightarrow$D plus E$\rightarrow$F) presented in Fig. 4(c) confirm these probabilities. Again, the effect is due to the probability that the CWND falls below a critical value. This critical value depends, however, on the RTT, so that the probability of incurring stack delays is also RTT dependent. For example, assume that the CWND is small, e.g., less than 5 MSS, so that it takes 3 rounds to deliver a video frame. In case of an RTT of 42 ms but not in case of 28 or 14 ms the transfer delay exceeds 100 ms and the next frame has to wait in the protocol stack.

The stack delays may also be interpreted as queueing delays at a system with random service [22] and the magnitude of the delays can be related to the utilization. Since the greedy TCP throughput depends on the RTT, see Tab. II, we have different utilizations of 13.8%, 28.2%, and 39.0% in case of an RTT of 14, 28, and 42 ms, respectively. We note that the stack delays are significant already at a moderate utilization of 39.0%.

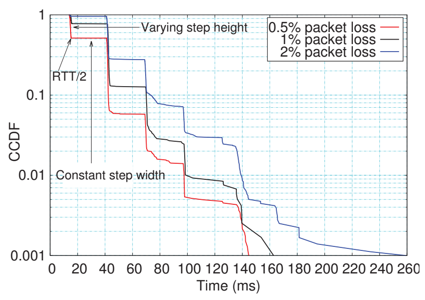
B. Loss-dependent Delay Probabilities

Next, we evaluate the impact of the packet loss probability. For the experiments, we set the loss rate to 0.5, 1, and 2%, respectively. The RTT is fixed to 28 ms.

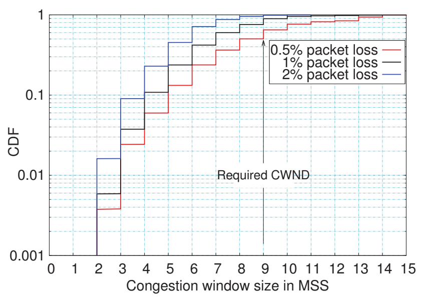
1) Role of the CWND Distribution: Fig. 5(a) displays the CCDFs of the transfer delays. All curves show the same characteristic stepped shape as in Fig. 4(a) with a step width that is determined by the RTT of 28 ms. The curves differ, however, with respect to their step height that decreases in case of a larger packet loss rate, i.e., transfer delays that exceed the minimal transfer delay of RTT/2 by one or more RTTs become more frequent if the packet loss rate is larger.

The effect is caused by the CWND that is stochastically decreasing in the packet loss rate. The CWND falls with a higher probability below certain critical values if the packet loss rate is increased. The CDFs of the CWND are presented in Fig. 5(b). As before, a CWND of at least 9 MSS is required to be able to transmit an entire video frame at once, whereas it takes at least one additional round of one RTT if the CWND falls below 9 MSS. This happens with probability 0.5, 0.77, and 0.95 in case of a packet loss rate of 0.5, 1, and 2%, respectively. The probabilities correspond to the step heights of the first step of the transfer delay CCDFs in Fig. 5(a). The following steps are explained similarly.

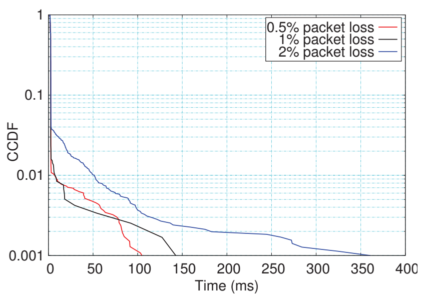
2) Effect on Stack Delays: The CWND distribution also affects the probability and the magnitude of stack delays. With increasing packet loss rate, the CWND falls more frequently below the critical value that is required to deliver a frame within the frame generation interval of 100 ms. Likewise, the utilization increases with the loss rate, i.e., in the experiments the utilization is 20.0%, 28.2%, and 39.1% given the packet



(a) Transfer delay



(b) CWND



(c) Stack delay

Fig. 5. Impact of the packet loss rate. The RTT is 28 ms. The packet loss rate determines the height of the steps of the transfer delay CCDF. The effect is via the CWND, that is stochastically decreasing with the packet loss rate. For increasing packet loss rates, large tail delays are observed in the stack.

loss rates of 0.5%, 1%, and 2%, respectively, causing larger queueing delays in the stack.

V. Adaptation of Video Parameters

The RTT and the packet loss rate determine the TCP throughput as well as transfer and stack delays, as observed for a defined video traffic profile in Sec. IV. In order to mitigate delays, the video traffic can be adapted using a variety of encoding parameters that determine the compression gain and hence the data rate. Common options are adaptation of the temporal resolution, the spatial resolution, or the quantizer, i.e., the quality. While the first option determines the frame rate, the other two options influence the frame size. Generally, the goal of adapting the video data rate is to control the utilization to avoid load-dependent delays. In our evaluation, we discover that the interaction with the TCP protocol stack causes a number of relevant other effects.

A. Frame Rate

To investigate the impact of the frame rate, we adapt the renderer application to generate 10, 20, and 30 fps, respectively. First, we consider the effect of the utilization on delays before we turn to an artifact that is caused by TCP's fast retransmit algorithm.

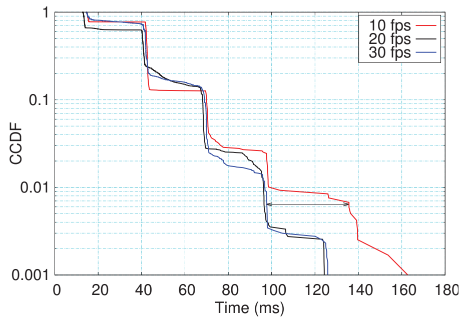
1) Effect of the Utilization: In Figs. 6(a) and 6(b) we show the impact of the frame rate on the transfer and the stack delay, respectively. The RTT is 28 ms and the loss rate 1%. In relation to the greedy TCP throughput the utilization is 28.2%, 56.4%, and 84.6% for 10, 20, and 30 fps, respectively. While we notice little change in the transfer delay in Fig. 6(a), a significant impact of the frame rate on the stack delay is observed in Fig. 6(b). As before, the stepped transfer delay curves in Fig. 6(a) are caused by the size of the CWND, e.g., if the CWND falls below the frame size of 9 MSS it takes one or more additional RTT-sized rounds to transfer a frame, see Fig. 4(b) and the explanation in Sec. IV-A1. This effect is independent of the frame rate.

If the transfer of a frame exceeds the time until the next frame is ready for transmission, that is the reciprocal of the frame rate, stack delays occur. Fig. 6(b) confirms that the probability of non-zero stack delays corresponds to the probability to see transfer delays of more than 100, 50, and 33 ms for 10, 20, and 30 fps, respectively. The magnitude of the stack delays in Fig. 6(b) demonstrates the effectiveness of rate adaptation and supports earlier observations [18], [22] that good delay performance is achieved only if the video data rate is smaller than the greedy TCP throughput by a factor of about two or more.

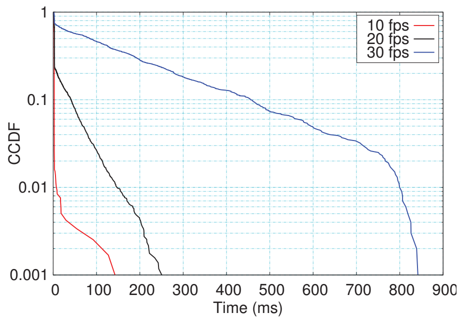
2) Issue of Last Segment Lost: In Fig. 6(a), the tail delay at a probability of 0.01 shows an opposing effect: the transfer delay is larger by almost 50 ms in case of a smaller frame rate of 10 fps. The effect persists if the RTT is reduced from 28 ms in Fig. 6(a) to 14 ms in Fig. 7, where additionally a difference of about 17 ms is noticed between the curves obtained for frame rates of 30 and 20 fps.

The reason for this is packet loss, specifically loss of the last TCP segment of a video frame. In general, TCP's fast retransmit algorithm, that is triggered by three duplicate acknowledgements, as well as selective repeat of missing segments indicated by SACKs deal effectively with loss. An exception is the loss of the last TCP segment of a video frame after which the transmission pauses. It resumes when the next video frame is available and eventually duplicate acknowledgements or a SACK indicate the missing TCP segment and trigger the retransmission. In the experiments, the loss rate is 1% and the frame generation period is 100, 50, and 33 ms in case of a frame rate of 10, 20, and 30 fps, respectively. Consequently, if the frame rate is smaller, it takes longer until the next frame resumes transmission and the retransmission is triggered. The time differences $100-50 = 50$ and $50 - 33 = 17$ ms are observed in Fig. 7 roughly at the loss probability of 0.01.

In more detail, if the last TCP segment of a frame is lost, it takes up to one frame generation period plus at least one RTT until duplicate acknowledgements or a SACK appear at the sender. Once there are three duplicate acknowledgements or a SACK, the retransmission is triggered. It arrives earliest after another RTT/2 at the receiver. In case of an RTT of 14 ms the numbers add up to 54, 71, and 121 ms for a frame rate of 30, 20, and 10 fps, respectively. The arrows in Fig. 7 mark these numbers approximately. For Fig. 6(a), where the RTT is 28 ms, the numbers are 75, 92, and 142 ms. The reason why



(a) Transfer delay



(b) Stack delay

Fig. 6. Impact of the frame rate. The RTT is 28 ms and the packet loss rate 1%. The frame rate determines the utilization that is 28.2%, 56.4%, and 84.6% for 10, 20, and 30 fps, respectively. The CCDF of the transfer delay shows little influence of the frame rate with significant deviations only in the tail. In contrast, the frame rate has a major impact on the stack delays. The probability to see non-zero stack delays corresponds to the probability that the transfer delay exceeds the frame generation interval of 100, 50, and 33 ms, respectively.

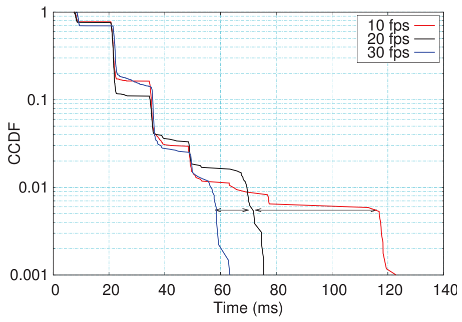


Fig. 7. Impact of the frame rate on the transfer delay. The RTT is 14 ms and the packet loss rate 1%. The tail transfer delays increase with decreasing frame rate. The effect is due to packet loss of the last TCP segment of a video frame. In this case duplicate acknowledgements or SACKs trigger TCP fast retransmit or selective repeat, respectively, only when the next frame is transmitted. The arrows indicate the differences between the frame generation intervals of 100, 50, and 33 ms.

the transfer delays in Fig. 6(a) show no difference between 30 and 20 fps is that due to the small CWND at probability 0.01, the tail delay to deliver a frame is 3.5 RTT, i.e., 98 ms, and hence larger than the estimated delays of 75 and 92 ms for retransmission of the last TCP segment.

B. Frame Size

Given that a reduction of the video frame rate can lead to larger transfer delays, we now investigate whether an adaptation of the frame size is more effective. We modify the quantizer to generate video streams with different frame sizes, where we specify the frame size relative to the one that we used in the previous experiments. We show results for frame sizes of 80%, 100%, and 120%. The frame rate is 10 fps and the network has an RTT of 28 ms and a loss rate of 1%.

1) Impact on the Transfer Delays: In Fig. 8(a), we consider the CCDF of the transfer delay. As before, we notice that the curves have distinct steps. The width of the steps is independent of the frame size and determined by the RTT. The height of the steps increases with decreasing frame size. This means that reducing the frame size effectively improves the transfer delay.

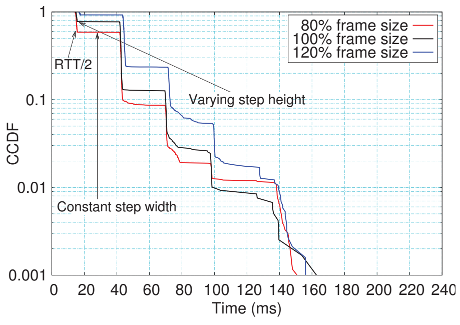
2) Relation with the CWND: The way in which the frame size helps reduce the transfer delay is via its relation to the CWND. First, we notice that the frame size has little effect on the CDF of the CWND, that is presented in Fig. 8(b), where larger frame sizes tend to result in slightly larger CWNDs. A possible reason for this is TCP's Congestion Window Validation algorithm [28], that freezes the CWND if it is not fully utilized. Hence, the CWND only rarely grows to large values if the frame size is small.

Given the similarity of the CWND CDFs in Fig. 8(b), the improvement of the transfer delay is due to the different CWND requirements given frames of different size. In the above case, the frame sizes correspond to 7, 9, and 11 TCP segments, respectively. If the CWND falls below any of these values, frames of the respective size cannot be transmitted in one round, resulting in one or more additional RTTs of transfer delay. The required CWNDs are marked in Fig. 8(b) and the probabilities that the CWND falls below any of these values are clearly visible as the height of the first step of the different transfer delay CCDFs in Fig. 8(a).

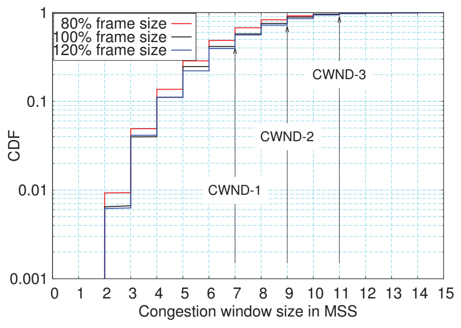
Regarding the stack delays, we observe relatively moderate values that do not show strong differences for the frame sizes above. The reason is the low utilization of 22.7%, 28.2%, and 33.8%, respectively. We omit showing the results and note, however, that the adaptation of the frame size can effectively reduce stack delays if the utilization is high.

VI. Conclusions

We investigated interactive TCP streaming of a free-viewpoint rendering application. We instrumented the entire streaming chain to identify delays and performed a comprehensive measurement study in a controlled network testbed to analyze the impact of relevant network and encoding parameters. A finding is that the utilization of the TCP connection has to be kept low as it has a major impact on delays in the sender's TCP socket buffer. Further, a pronounced influence of the distribution of the CWND on the transfer



(a) Transfer delay



(b) CWND

Fig. 8. Impact of the frame size. The RTT is 28 ms, the packet loss rate 1%, and the frame rate 10 fps. The frame size effectively controls the CWND requirements of the video stream, where a CWND of 7, 9, and 11 MSS, respectively, permits sending a video frame at once. The probabilities that the CWND falls below these critical values correspond to the probabilities to see transfer delays of more than RTT/2.

delay is noticed. The insights guide the design of adaptive applications and suggest to include feedback about the size of the CWND and the socket buffer filling to the application. Our measurements of the end-to-end delay showed a high variability of a few 100 ms. Hence, the receiver of a video streaming application has to provision an adequate de-jitter buffer to accomodate the observed tail delays. In case of our rendering application, frames are displayed immediately when they arrive at the receiver. For moderate RTT and packet loss, stalling is sporadic and the user perception of the time lag is mostly acceptable.

References

[1] S. Aljoscha, "3d video and free viewpoint video – from capture to display," *Pattern Recognition*, vol. 44, no. 9, pp. 1958–1968, Sep. 2011.

[2] J. Chakareski, "Adaptive multiview video streaming: challenges and opportunities," *IEEE Communications Magazine*, vol. 51, no. 5, pp. 94–100, May. 2013.

[3] A. Hamza and M. Hefeeda, "Adaptive streaming of interactive free viewpoint videos to heterogeneous clients," in *Procs. of the 7th International Conference on Multimedia Systems*, May. 2016, pp. 10–22.

[4] X. Xiu, G. Cheung, and J. Liang, "Frame structure optimization for interactive multiview video streaming with bounded network delay," in *Procs. of IEEE International Conference on Image Processing*, Sep. 2011, pp. 593–596.

[5] ——, "Delay-cognizant interactive streaming of multiview video with free viewpoint synthesis," *IEEE Trans. on Multimedia*, vol. 14, no. 4, pp. 1109–1126, Aug. 2012.

[6] T. Maugey and P. Frossard, "Interactive multiview video system with low complexity 2d look around at decoder," *IEEE Trans. on Multimedia*, vol. 15, no. 5, pp. 1070–1082, Aug. 2013.

[7] E. Kurutepe, M. R. Civanlar, and A. M. Tekalp, "Client-driven selective streaming of multiview video for interactive 3DTV," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 17, no. 11, pp. 1558–1565, Nov. 2007.

[8] M. Ueberheide, F. Klose, T. Varisetty, M. Fidler, and M. Magnor, "Web-based interactive free-viewpoint streaming: A framework for high quality interactive free viewpoint navigation," in *Procs. of the ACM International Conference on Multimedia*, Oct 2015, pp. 1031–1034, Short Paper.

[9] X. Yan, L. Yang, S. Lan, and X. Tong, "Application of HTML5 multimedia," in *Procs. of International Conference on Computer Science and Information Processing (CSIP)*, Aug. 2012, pp. 871–874.

[10] A. Vetro, T. Wiegand, and G. J. Sullivan, "Overview of the stereo and multiview video coding extensions of the H.264/MPEG-4 AVC standard," *Procs. of the IEEE*, no. 4, pp. 626–642, Apr. 2011.

[11] G. Cheung, A. Ortega, and N. M. Cheung, "Interactive streaming of stored multiview video using redundant frame structures," *IEEE Trans. on Image Processing*, vol. 20, no. 3, pp. 744–761, Mar. 2011.

[12] J.-G. Lou, H. Cai, and J. Li, "A real-time interactive multi-view video system," in *Procs. of the ACM International Conference on Multimedia*, Nov. 2005, pp. 161–170.

[13] D. Yun and K. Chung, "Dash-based multi-view video streaming system," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 99, no. 99, pp. 1–1, Apr. 2017.

[14] A. Hamza and M. Hefeeda, "A DASH-based free viewpoint video streaming system," in *Procs. of Network and Operating System Support on Digital Audio and Video Workshop*, Mar. 2014, pp. 55–60.

[15] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hofeld, and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 469–492, Mar. 2015.

[16] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Procs. of the ACM Conference on Multimedia Systems*, Feb. 2011, pp. 157–168.

[17] K. Miller, A.-K. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, pp. 41–44, Oct. 2016.

[18] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," in *Procs. of the ACM International Conference on Multimedia*, Oct. 2004, pp. 908–915.

[19] A. Goel, C. Krasic, K. Li, and J. Walpole, "Supporting low latency TCP-based media streams," in *IEEE International Workshop on Quality of Service*, Aug. 2002, pp. 193–203.

[20] A. Goel, C. Krasic, and J. Walpole, "Low-latency adaptive streaming over TCP," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 3, pp. 21–40, sep. 2008.

[21] R. Lübben and M. Fidler, "On characteristic features of the application level delay distribution of TCP congestion avoidance," in *Procs. of IEEE International Conference on Communications (ICC)*, May 2016.

[22] ——, "Service curve estimation-based characterization and evaluation of closed-loop flow control," *IEEE Trans. on Network and Service Management*, vol. 14, no. 1, pp. 161–175, Mar. 2017.

[23] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The delay-friendliness of TCP for real-time traffic," *IEEE/ACM Trans. on Networking*, vol. 18, no. 5, pp. 1478–1491, Oct. 2010.

[24] J. Wu, C. Yuen, and J. Chen, "Leveraging the delay-friendliness of TCP with FEC coding in real-time video communication," *IEEE Trans. on Communications*, vol. 63, no. 10, pp. 3584–3599, Oct. 2015.

[25] Y. Xiong, M. Wu, and W. Jia, "Rate adaptive real-time video transmission scheme over TCP using multi-buffer scheduling," in *Procs. of the International Conference for Young Computer Scientists*, Nov. 2008, pp. 354–361.

[26] R. L. Cigno and M. Gerla, "Modeling window based congestion control protocols with many flows," *Performance Evaluation*, vol. 36-37, pp. 289–306, Aug. 1999.

[27] M. Garetto, R. L. Cigno, M. Meo, and M. A. Marsan, "Modeling short-lived TCP connections with open multiclass queuing networks," *Computer Networks*, vol. 44, no. 2, pp. 153–176, Feb. 2004.

[28] G. Fairhurst, A. Sathiaseelan, and R. Secchi, "Updating TCP to support rate-limited traffic," RFC 7661, Oct. 2015.