An Evaluation of TCP-based Rate-Control Algorithms for Adaptive Internet Streaming of H.264/SVC

Robert Kuschnig, Ingo Kofler, Hermann Hellwagner
Institute of Information Technology (ITEC)
Klagenfurt University, Austria
{firstname.lastname}@uni-klu.ac.at

ABSTRACT

Recent work [18] indicates that multimedia streaming via TCP provides satisfactory performance when the achievable TCP throughput is approx. twice the media bit rate. However, these conditions may not be achievable in the Internet, e.g., when the delivery path offers insufficient bandwidth or becomes congested due to competing traffic. Therefore, adaptive streaming for videos over TCP is required and a number of rate-control algorithms for video streaming have been proposed and evaluated in the literature.

In this paper, we evaluate and compare three different rate-control algorithms for TCP in terms of the (PSNR) quality of the delivered video and in terms of the timeliness of delivery. The contribution of the paper is that, to the best of our knowledge, this is the first evaluation of TCP-based streaming in an Internet-like setting making use of the scalability features of H.264/SVC video. Two simple bandwidth estimation algorithms and a priority-/deadline-driven approach are described to adapt the bit rates of, and transmit, the H.264/SVC GOPs in a rate-distortion optimal manner. The results indicate that the three algorithms perform robustly in terms of video quality and timely delivery, both on under-provisioned links and in case of competing TCP flows. The priority-/deadline-driven technique is even more stable in terms of packet delays and jitter; thus, client buffers can be dimensioned more easily.

Keywords

H.264/SVC, TCP video streaming, adaptive video streaming, rate control, TCP fairness

1. INTRODUCTION

Video streaming over the Internet based on the TCP transport protocol became widespread in the recent past. The reason for this is the popularity of video portals like YouTube or social networks that allow to share and distribute user generated content [16]. Although, from a technical point of view, the use of TCP for real-time multimedia applications is considered controversial in the literature [10, 5], there are good arguments for TCP. First, it offers a reliable end-to-end transport mechanism that makes additional provisions like forward error correction unnecessary. The connection-oriented design of the protocol further allows for easy traversal of firewalls and NAT devices that may exist in the network [1]. As a consequence, the development and deployment of TCP-based streaming applications turns out to be much less complex than approaches based on the connectionless UDP protocol. Finally, TCP also offers congestion control that, at the same time, tries to maximize the throughput while preventing the network from collapsing. This assures that only the fair share of the network bandwidth is consumed by a single connection, which is considered crucial for the stability of the Internet [4].

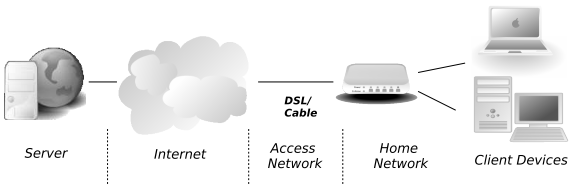

Figure 1: Use case

As the throughput of the TCP connection depends on both, the link capacity and the amount of congestion, the throughput can vary significantly over time. A typical use case is illustrated in Figure 1, where a home network consisting of several computers or consumer electronic products is connected to the Internet via a DSL or cable-based access network. In this common deployment scenario, the capacity of the link and the congestion caused by parallel TCP connections influence the available bandwidth for a video streaming application. Variations in the available bandwidth will result in jerky playback and disruption of the video playback if the throughput falls below the bit rate requirement of the video. Adaptive video streaming [1] tries to overcome this, by adjusting the video bit rate to the available network bandwidth. In adaptive TCP streaming, TCP's adaptability to fluctuating network conditions is used to steer the rate control. TCP-based video streaming and especially adaptive streaming are extensively covered in the literature [17, 11, 8, 6, 18, 15]. An approach which uses adaptive video transcoding based on the TCP transmission rate, is presented in [15]. Other approaches like the TCP-based priority/priority-progress streaming [17, 11] send the more important parts

of the video before the low-priority parts, in order to deliver a rate-distortion optimal video.

Adaptive video streaming requires that the bit rate of the video can be adjusted on-the-fly on the server. However, bit rate adaptation based on transcoding or transrating comes along with high computational cost and scalability issues [1]. An alternative is the use of a scalable video coding format that offers simple video adaptation at the cost of a more sophisticated encoder/decoder design. Although scalability features were included in both standardized and proprietary codecs in the past, their success was rather limited due to coding inefficiencies and decoding complexity. The Scalable Video Coding (SVC) extension [21] of the well-known H.264/AVC standard, however, aims at overcoming the shortcomings of the past codecs. The performance and merits of H.264/SVC have been successfully demonstrated in various use cases [22]. However, to the best of our knowledge, no evaluation w.r.t. adaptive Internet streaming based on TCP and H.264/SVC has been conducted so far.

Therefore, we provide in this paper an in-depth evaluation of three different adaptive TCP-based video streaming approaches that utilize scalability features of H.264/SVC videos. The paper is organized as follows. Section 2 briefly introduces the H.264/SVC video coding standard and its scalability features. An overview of TCP-based video streaming, its challenges and existing approaches are provided in Section 3. The use of H.264/SVC in combination with the adaptive streaming approaches and a detailed description of the algorithms under evaluation are given in Section 4. Section 5 describes the evaluation setup and methodology. The results of our extensive evaluation are presented and discussed in Section 6. Section 7 concludes the paper.

2. H.264/SVC

The H.264/SVC video coding standard [21] was recently standardized as an extension to the well-known H.264/AVC standard. In this context, a video stream is considered scalable, if the stream can be adapted by the removal of parts of the encoded bit stream. Since H.264/SVC extends H.264/AVC, its design is also based on the distinction between the Video Coding Layer (VCL) and a Network Abstraction Layer (NAL). While the VCL deals with the compression of the video data, the NAL is used for organizing the compressed video for storage and transmission [23]. H.264/AVC introduced the concept of a NAL unit which consists of a one byte header and the payload data. NAL units can be classified into VCL NAL units, which contain coded video slices, and non-VCL NAL units which provide other decoder-relevant information like picture and sequence parameter sets. The type of NAL unit is signaled in the one byte header.

H.264/SVC introduces scalability in three dimensions. *Temporal scalability* is the property of a bit stream that video sequences with different frame rates can be extracted. Generally, temporal scalability can be achieved by restricting the motion-compensated prediction to hierarchical prediction structures. Hierarchical B-pictures or non-dyadic structures were already possible with coding tools standardized in H.264/AVC. The only changes introduced by H.264/SVC are related to signaling the different temporal layers.

Spatial scalability, which allows the extraction of video sequences with different spatial resolutions, is also realized in a layered fashion. In each spatial layer, motion-compensation prediction and intra-prediction can be employed. Additionally, SVC introduces inter-layer prediction which means that a picture at a higher spatial layer can be predicted by upsampling the signal from its corresponding lower layer picture.

For *quality scalability*, the H.264/SVC standard offers two approaches. Coarse-grain quality scalable coding (CGS) is similar to spatial scalability, with the difference that the pictures at each layer have the same spatial dimensions. However, this concept comes along with a lower coding efficiency for small relative rate differences between successive CGS layers. An efficient encoding of video bit streams that cover a variety of bit rates can be realized by using medium-grain quality scalable coding (MGS). The MGS concept enables the removal of any NAL unit belonging to a quality enhancement layer. This can be used to perform a finer-grained adaptation of the bit rate and with a higher flexibility. While CGS-based quality adaptation can only be switched at defined points in the bit stream, MGS allows to switch the quality and bit rate at every picture of the video stream. Furthermore, it is possible to partition the coefficients of the transformation and distribute the information among several NAL units.

In order to signal the layer information in the bit stream, SVC extends the NAL unit concept of H.264/AVC [19]. Newly introduced, SVC-specific NAL unit types required a three-byte header extension. In the extended header, the layer information is signaled by the fields *quality id (QID)*, *dependency id (DID)*, and *temporal id (TID)*. In addition, the header contains a *priority id (PRID)* field which can be used to define a suggested adaptation path. This adaptation path specifies in which order the NAL units should be discarded in case of adaptation. The assignment of the priority id to NAL units is not further specified in the standard and can be allocated based on the needs of a certain application or use case. The Quality Level Assigner tool that is included in the JSVM [9] reference software, can be used to assign priority values to NAL units contained in the bit stream. The assignment is done in a way that the extraction based on the PRID is optimal w.r.t. rate-distortion.

3. TCP-BASED VIDEO STREAMING

Video streaming based on TCP has become very popular because of its easy deployment and congestion awareness. For these reasons, a lot of content providers use TCP for streaming multimedia content over the Internet [16, 10]. While TCP performs very well in networks with small round-trip times (RTTs), it becomes more problematic in networks with high bandwidth-delay products, like the Internet. TCPs congestion control is based on the additive-increase/multiplicative-decrease (AIMD) algorithm. The MD step reduces the TCP window (and therefore the throughput) in case of packet loss drastically, before the AI step tries to increase the window once again. This leads to a notable variation of the throughput in networks with large RTTs. An extensive analysis on TCP streaming was conducted in [18]. The results of the analysis indicate that streaming shows good results when the available bandwidth is twice the media bit rate. This kind of overprovi-

sioning is feasible for streaming media at low bit rates, but high-definition media demands significantly higher throughput. While a large number of last-mile networks offer downlink bandwidths greater than 4 Mbps [3], having twice the bit rate of an HD stream is rather an exception. In addition, TCP features no implicit error recovery on connection abort or stalls; this has to be handled on the application layer.
New TCP implementations like TCP Cubic [7] are tackling these problems by introducing mechanisms for fast recovery on packet losses. In contrast to this, unreliable but congestion-aware transport protocols, like the Datagram Congestion Control Protocol (DCCP) [5], were proposed. While they are able to reduce end-to-end latencies by omitting reliability, they suffer from deployment problems in existing network infrastructures.
Apart from the systematic problems of TCP, also performance considerations have to be made regarding the TCP implementation. The TCP stack is in general implemented in the kernel space of the operating system because of performance and security issues. Therefore, an interface between the user space and the kernel space is needed, which is represented by the TCP socket API and a socket buffer. The TCP socket buffer has a direct impact on the TCP performance, because it limits the maximum throughput by restricting the maximum size of the TCP window [24]. So the TCP socket buffer size corresponds directly to the maximum TCP throughput. In general, it is recommended to set the TCP socket buffer size to twice the value of the bandwidth-delay product (BDP) [24]. For adaptive real-time streaming it is not the goal to fully utilize the network link, but to transmit the video data in real-time. Therefore the TCP socket buffer is dimensioned regarding the maximum video bit rate and not the maximum available bandwidth.

In the literature different approaches for adaptive TCP streaming exist. We selected the approaches for our evaluations, which only observe the TCP channel and use the information in an adaptive streaming process. So the streaming systems do not change the behavior of the TCP transmission system. In the following three different approaches for adaptive TCP streaming are introduced; their realisation by using H.264/SVC will be further explained in Section 4.

Application-layer Bandwidth Estimation: The first approach measures the available bandwidth based on the time spent for the transmission of a specific media block [15]. The delivery module writes the media data into a blocking TCP socket and estimates the actual delivery bit rate by simply counting the bytes for a time slot of one second. After retrieving an estimation value for the available bandwidth, the adaptation process uses this information to configure the media transcoder. This approach to adaptive TCP streaming is very straight forward and does not use operating system specific information. As shown in [15] it can adapt to the available bandwidth, but is very implementation specific because it relies on the blocking socket API.

TCP Stack-based Bandwidth Estimation: The current congestion window and the estimated RTT of the TCP connection can be used to estimate the available bandwidth [12]. In [2] for example a simple model for the expected latency of packets sent with TCP Reno is presented. The model also estimates the available channel rate and the expected

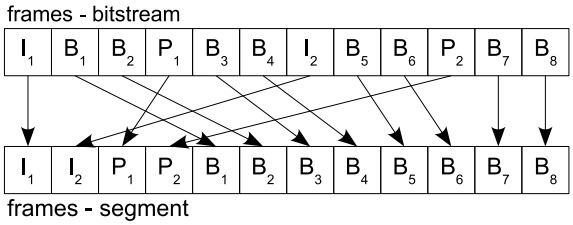

Figure 2: Priority-order within a *video segment*

distortion. The TCP parameters are directly used in the encoding process of the video, which allows the streaming system to adapt to the available network conditions.

Priority Streaming: In contrast to traditional buffering at the receiver, which only tries to overcome bandwidth shortages, *priority streaming* [17, 11] tries to improve the quality of the video over a period of time. A video sequence is split into fragments, each comprising the same play-out duration (GOP). The idea is to rearrange the video syntax elements (e.g., slices, frames, layers) in a video fragment depending on its priority. For example, using a non-scalable video codec, the I-frames would precede the P-frames and the B-frames (see Figure 2). The reordered video fragment is called a video segment. As a result of this reordering, it is not necessary to retrieve the whole video segment; it can be truncated at virtually any point. The quality of the video depends only on how much of the video segments was retrieved. The size of the GOP directly influences the quality variations between the video segments, which favors larger GOPs. On the other hand, the play-out delay increases with the size of the GOP. In most cases, the size of the GOP is restricted by the use case, which dictates maximum play-out delay. A TCP-based priority streaming system uses a single TCP connection for transmission. From the server, each video segment is streamed to the client. When reaching the timeout of a video segment, the server stops transmitting the segment and switches to the next one. From the received video segment, the video can be reconstructed and displayed to the user.

A different but somehow related paradigm to the approaches under investigation is the stream-switching technique. In case of stream switching, the content is available at the server at different bit rates; depending on the actual network conditions the bit rate can be switched. This technique is already deployed in commercial applications like the HTTP-based streaming used in Apple products [14] or the HD Internet streaming solution provided by Move Networks [13]. While the products mentioned above rely on the client to decide when to switch to a higher or lower bit rate version of the content, it is also possible to implement server-side stream switching. The most important difference to the approaches evaluated in this paper is that the stream-switching solution does not scale very well. First, the content has to be encoded using different bit rates which causes an additional effort especially when considering streaming of live content. This would require to encode the content in parallel with multiple encoders. Second, all the different bit rate versions of the same video have to be stored

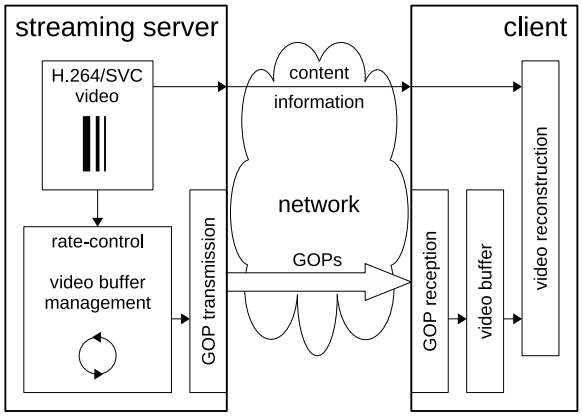

Figure 3: Server-side adaptive streaming system

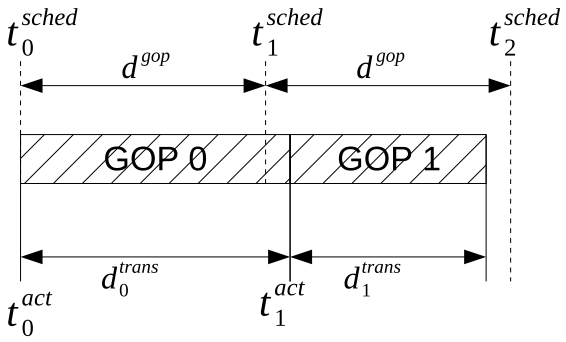

Figure 4: GOP timing

at the server which causes additional storage demand that increases with the number of bit rates offered. Both the implications on the encoder and storage demand lead to the fact that stream-switching solutions only offer a handful of different video bit rates for economical reasons. In contrast, the approaches based on H.264/SVC allow to have a significant higher number of different bit rate versions of the same content. When using MGS quality scalability in combination with bit stream extraction based on the priority id, the bit rate can be adapted in up to 64 discrete steps. This leads to a much higher dynamic range of available bit rates and the advantage of having a single bit stream stored at the server. Additionally, only a single encoder is required in the case of live content.

In the following section, we will show how these approaches can be used in conjunction with H.264/SVC and describe the implementations used for the evaluation.

4. ADAPTIVE STREAMING OF H.264/SVC

For our evaluation of rate-adaptive streaming algorithms, we adapted the three approaches mentioned above for using H.264/SVC. We intentionally focused on adaptation mechanisms that can be characterized as server-side (server-based) algorithms. The rationale behind that is that this allows an easier deployment of the adaptive streaming solutions. An architectural figure of such a server-side streaming solution is given in Figure 3. At the server the video is stored as scalable H.264/SVC bit stream. A rate-adaptive component adapts the bit stream accordingly and transmits the bit stream to the client. The transmission is performed on a per-GOP basis. At the client the the GOP is received, buffered and decoded for the final playback. Additionally, all three approaches do not rely on explicit feedback from the client like notifying the current buffer status or similar. The only kind of feedback that is used at the server is the implicit feedback via the TCP protocol. For instance, blocking send operations on the socket on the server side are used as indication of the connection status.

Another characteristic that is shared by all three approaches is that the timely transmission of the video is controlled at the server side. On the one hand, this means that the server takes care that not too much of the video content is streamed out in advance. The transmission of the content takes place at the same speed as the video is consumed at the client side. This real-time behavior clearly distinguishes the approaches from other paradigms (download and play) that can be found in other streaming solutions in the Internet. The advantage of the real-time playout is that a single client with a fast network connection cannot consume more than its fair share, which is at most the maximum bit rate of the video, and cannot overload the streaming server. On the other hand, this server-side mechanism is further necessary because otherwise the scheduled transmission times of video segments and the actual transmission times can differ in case of congestion. In case of persistent congestion, this difference slightly increases over time and causes the video segments to arrive too late at the client. Since this behavior drains the buffer at the client and causes the video playback to stop at the player, all three approaches employ countermeasures to prevent this drift. In the following the three implementations of the approaches are described in detail.

Application-layer Bandwidth Estimation

The first rate-adaptive approach is based on bandwidth estimation that is done at the application layer (APP-BE). Basically, the server estimates the bandwidth of the TCP connection by considering the number of bytes sent through the socket API during a time interval. In our implementation, the estimation is performed on a per-GOP basis. The number of bytes of the GOP to transmit is known in advance by the server. By measuring the time that is required to send the GOP via the socket API the server calculates the current throughput of the TCP connection. For calculating the bandwidth estimate for the next GOP to send, the measured throughput values of the last five GOPs are averaged. Based on the estimate, the next GOP is adapted by using H.264/SVC-based adaptation so that the total bit rate of the GOP is below the bandwidth estimate. Additionally, it should be noted that the estimate is further adjusted by the buffer control algorithm to prevent a drift of the actual transmission time of each GOP in case of congestion.

More formally, the algorithm can be described as follows. Let d^{gop} be the playback duration of a GOP in seconds and t_i^{sch} the scheduled start time for transmitting the i-th GOP at the server. Without loss of generality, we define $t_0^{sch} = 0$ and assume the scheduled start of any other GOP i as $t_i^{sch} = t_0^{sch} + d^{gop} * i$. In contrast to the scheduled start time we measure the actual start time of each GOP (t_i^{act})

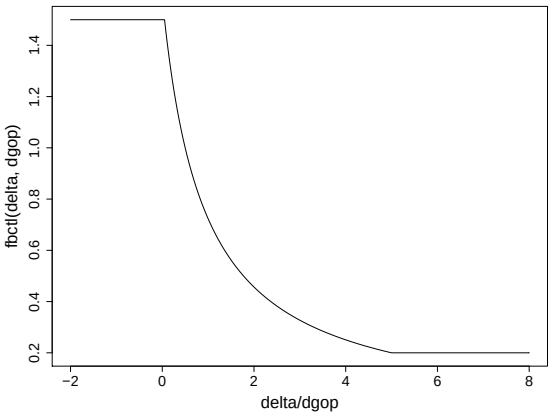

Figure 5: Buffer control function

at the moment before the first byte of a GOP is sent via the socket API. After successfully transmitting the GOP, the actual time is sampled again and the transmission duration (d_i^{trans}) of GOP i is determined. The introduced timestamps and durations are illustrated in Figure 4. Based on the known length of the GOP in bytes (l_i^{gop}), the measured throughput during transmission of GOP i can be simply calculated as $th_i = l_i^{gop}/d_i^{trans}$. The estimated bandwidth for the next GOP $i+1$ is then calculated by averaging over the last five throughput measurements and adjusting the average by a multiplicative term. This adjustment is required for the buffer control and prevents that the scheduled start times t_i^{sched} and the actual start times t_i^{act} begin to diverge. The multiplicative term is determined by the buffer control function $f_{bctl}(delta, d^{gop})$ that depends on the difference between both timestamps and the duration of one GOP. The function is illustrated in Figure 5 and described in more detail in the Appendix A. Finally, the bandwidth estimation (be) for GOP $i+1$ is calculated as follows.

$$be_{i+1} = \frac{1}{5} \sum_{j=i-4}^{i} th_j \cdot f_{bctl}(t_i^{act} + d_i^{trans} - t_{i+1}^{sch}, d^{gop}) \quad (1)$$

The bandwidth estimation for the next GOP is then used for adapting the GOP in a way that it matches the targeted bandwidth. For that purpose, the NAL units are optionally discarded by using the adaptation path defined by the priority id. As already mentioned above, the approach prevents from streaming out faster than real-time by imposing the constraint that the transmission of a GOP may not start earlier than scheduled, i.e., $\forall i : t_i^{act} \geq t_i^{sched}$.

A crucial requirement for this approach is that the size of the GOP is larger than the TCP socket buffer on the server. Otherwise, the socket implementation simply copies the data to transmit into the TCP socket buffer and immediately returns from the send operation. This implies that in order to achieve good performance, the TCP socket buffer on the server should be adjusted according the bit rate of the video and the envisaged RTT range.

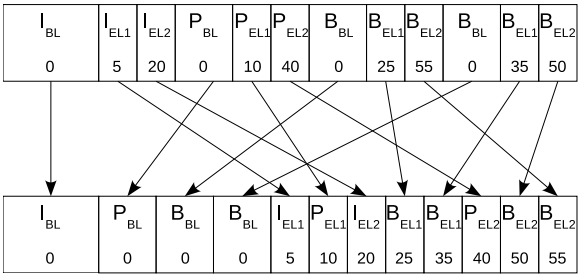

Figure 6: PRID-based NAL unit reordering

TCP Stack-based Bandwidth Estimation

The second approach under investigation is the TCP Stack-based Bandwidth Estimation (TCP-BE). This algorithm has very much in common with the application layer mechanism introduced above. Also in this approach, an estimate of the bandwidth is obtained and the next GOP to transmit is adapted based on that estimate. Furthermore, the estimate is again adjusted by a multiplicative term in order to prevent drift between the scheduled and actual transmission time of each GOP. However, the main difference is the way how the bandwidth estimation is calculated. Instead of measuring the TCP throughput at the application layer, the information about the TCP connection is obtained directly from the network stack. This is done by using the `getsockopt` API call in combination with the option `TCP_INFO`, which is supplied by most *nix operating systems. The information relevant for the bandwidth estimation consists of the current congestion window size $cWnd$, the maximum segment size MSS and the estimate of the round trip time RTT. The throughput estimate for the i-th GOP can then be calculated as [12]:

$$th_i = \frac{cWnd \cdot MSS}{RTT} \quad (2)$$

The bandwidth estimation (be) for GOP $i+1$ is calculated based on Equation 1 by taking into consideration the buffer control function. The rest of the approach is identical to Application-layer Bandwidth Estimation.

Deadline-driven Adaptive Streaming

In contrast to the previous approaches that rely on explicitly estimating the bandwidth, the Deadline-driven Adaptive Streaming (DEADLINE) algorithm is based on the priority streaming paradigm as introduced in Section 3. For our evaluation, the video is split up into video segments, where one segment corresponds to exactly one GOP. In a nutshell, this approach rearranges the NAL units of each GOP according to their priority before transmitting them. On the server a certain time interval is scheduled for the transmission of the rearranged GOP. The server tries to transmit as much data of the GOP as possible during that time interval. After the scheduled time interval has passed, i.e., the deadline for the GOP has been reached, the server starts transmitting the next GOP.

Typically, each GOP of the H.264/SVC scalable bit stream consists of H.264/AVC-compliant NAL units that form the base layer and H.264/SVC-specific NAL units that represent enhancement layers. Normally, these units are transmitted in decoding order which means that both kinds of NAL units

are interleaved. Following the priority streaming paradigm, they become rearranged so that the NAL units are transmitted in order of their priority id. NAL units with identical values of their priority id are arranged in decoding order. In this context, a lower numerical value of the priority id signals a higher priority of the NAL unit according to [20]. The basic principles of this reordering are illustrated in Figure 6. For the sake of simplicity the figure shows a GOP consisting of only four pictures (I, P, B, B) in decoding order. Each of the pictures is represented by one NAL unit carrying the base layer (BL) and two NAL units representing enhancement layers (EL1, EL2). The numerical values in the boxes represent the priority id value of each NAL unit; in our configuration all NAL units of the base layer have the highest priority. For transmission, the NAL units are ordered according to their priority. As a result of the scalable property of H.264/SVC, the quality of the decoded bit stream at the client monotonically increases with the amount of NAL units transmitted by the server.

As the decoder at the client side requires the NAL units in the proper decoding order, this reordering at the server has to be reversed at the client. For supporting the reordering, we embedded proprietary reordering information in the transmitted bit stream. Although this reconstruction at the client can also be achieved using standard-compliant methods, we intentionally used a proprietary approach for the sake of simplicity.

The transmission of the rearranged GOPs is based on the transmission schedule as already defined for the previous two approaches. The scheduled start time for the transmission of the i-th GOP is denoted as t_i^{sched}. Obviously, the deadline for the i-th GOP is the start time of the next GOP t_{i+1}^{sched}. During the time period $[t_i^{sched}, t_{i+1}^{sched})$, the server sends the NAL units via the socket API and checks after each send operation if the deadline has been reached. However, the send operation itself is never preempted. In case of an overprovisioned link, the transmission will be finished before the deadline is reached. The transmission of the next GOP however will be delayed until the scheduled start time of the next GOP. If the bandwidth of the link is insufficient to transmit the whole GOP in the given time interval, the server will skip the remaining NAL units and proceed with the next GOP. It should be noted that this approach does not rely on an explicit H.264/SVC adaptation process, the actual adaptation is performed implicitly by the deadline-driven transmission.

Remarks

Each of three approaches relies on H.264/SVC adaptation based on the priority id mechanism. For our actual research, we used the Quality Level Assigner tool of the JSVM [9] software to obtain the information about the importance of each NAL unit. This tool determines the priority information for MGS NAL units in a rate-distortion optimal way. However, it uses a semantic of the priority id that is different from the one specified in [20]. A post-processing step for assigning the proper priority id values based on the output of the Quality Level Assigner is therefore necessary. As a consequence, our further evaluation is limited to adaptation in the SNR domain, although the approaches themselves are more generic. This means that they are also applicable for temporal or spatial adaptation by assigning the priority ids appropriately. We considered MGS scalability as most suitable for the envisaged use case in the context of Internet streaming. MGS scalability allows a smooth playback at the client (e.g., in contrast to temporal adaptation) and also does not introduce too severe quality changes that would arise when switching between different spatial resolutions. Temporal and spatial scalability could be taken into account as a fallback solution in case of harder congestion situations and network conditions.

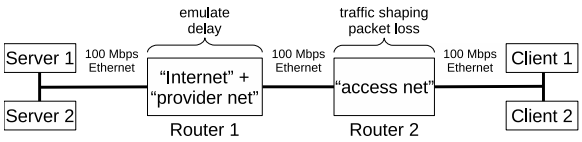

Figure 7: Evaluation setup

5. EVALUATION

We evaluated the behavior and performance of the proposed algorithms under conditions typical for Internet streaming. These include under-provisioned network links and congestion due to competing TCP traffic. A further criterion for assessing applications that are intended to be deployed in the Internet is their TCP friendliness. This is regarded as a necessity for the stability of the Internet and implies that an application/protocol does not consume more than its fair bandwidth share compared to competing TCP connections. Since all of the proposed approaches only make use of a single TCP connection, a TCP friendly behavior can be assumed for all systems.

Evaluation Setup

The evaluation setup consists of six Linux boxes representing two servers, two routers and two clients as illustrated in Figure 7. Each of them runs the Ubuntu Linux operating system (kernel 2.6.27). All client and server computers are configured to use the TCP Reno implementation. The routers are using the Netem[1] kernel component for performing emulation of network characteristics like delay, jitter and packet loss. The symmetric end-to-end delay of the Internet and the provider network is emulated by Router 1. The packet delay is normally distributed with 10% standard deviation and is applied to all packets. Router 2 emulates the asymmetric access network to the clients and limits the up- and downstream bandwidth (BW), while allowing a maximum queuing delay of 200 ms. The asymmetry of the network was emulated by setting the upstream bandwidth to 1/8 of the downstream bandwidth, which is common for DSL/cable access networks. The implementation of the three adaptive TCP streaming approaches is based on Python. For the evaluation, the video streaming server component is deployed on Server 1 while the client component is located at Client 1. To emulated a congested link, Client 2 issues parallel HTTP downloads from Server 2, that compete for their network share. The Apache[2] HTTP Server is used for serving the HTTP downloads.

[1] http://www.linuxfoundation.org/en/Net:Netem
[2] http://httpd.apache.org

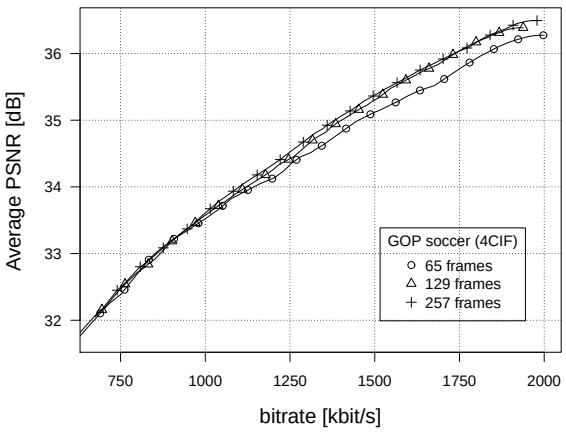

Figure 8: Rate-distortion curves of the test sequences

Network Scenarios

We evaluated the adaptive streaming approaches under three representative network conditions.

The first scenario is an *overprovisioned network link*, where the available network bandwidth substantially exceeds the video bit rate. Because TCP's congestion avoidance technique is based on the AIMD principle, throughput variations may occur in networks with high bandwidth delay products. This scenario is used to evaluate the ability of an approach to fully utilize the bandwidth for video transmission and to cope with the throughput variations. At the other extreme, the behavior of the approaches is further analyzed in a scenario of an *underprovisioned network link*. This should demonstrate the capability of adapting the video to a given limited network bandwidth while maximizing the video quality at the client. In the third scenario, we introduce cross traffic on the network link with one, two and three concurrent infinite-source TCP streams. On such a *congested network link*, an adaptive streaming system has to cope with high bandwidth fluctuations, because of the competition between the concurrent TCP streams. Further, it should provide a TCP-friendly behavior.

Content

The test sequences used for evaluation were created with the H.264/SVC codec provided by the Joint Scalable Video Model (JSVM) [9] 9.15 software. The video *soccer* in 4CIF resolution was encoded with different GOP sizes. The first *65*, *129* and *257* frames of the video *soccer* form a test sequence, each comprising 2.16, 4.30 and 8.57 seconds of video at 30 fps, respectively. Each sequence features an H.264/AVC backward compatible base layer and one MGS quality enhancement layer. Within the MGS quality enhancement layer, the transform coefficients are uniformly partitioned into four NAL units. The Quality Level Assigner tool (JSVM) was used to assign 64 different priority ids to the NAL units based on rate-distortion values. In Figure 8 the rate-distortion values of the test sequences are shown, ranging from the lowest bit rate (highest priority) to the full bit rate (lowest priority). It can be observed that the sequences with smaller GOP sizes have a slightly worse rate-distortion performance, because the prediction structures are restricted to the GOP length.

Methodology

Although each test sequence comprises a different number of frames, it can be observed in Figure 8 that the average PSNR values of the sequences are similar. The *soccer65* sequence is streamed 400 times in a loop, which results in approx. 900 seconds play-out duration. Because each test sequence represents a video segment with a different duration, we are streaming the *soccer129* and *soccer257* sequences 200 and 100 times in a loop, respectively. This results in a similar play-out duration, which enables a fair comparison of the evaluation results for the different GOP sizes. In addition to the different network scenarios, also the RTT is varied, ranging from 50 ms to 200 ms. Each experiment is repeated three times to reduce the influence of the runtime environment on the results. In all evaluation runs, no frames or GOPs are lost, because the streaming systems are based on the reliable transmission of TCP.

The TCP socket buffer size was determined by taking the considerations of Section 3 into account. The streaming systems should be able to utilize overprovisioning to enhance their streaming performance, so we allow 50 % higher TCP throughputs. So the TCP socket buffer size l_{sock} is a result of the maximum bit rate of the video br_{max} ($\approx$ 2048 $kbps$) and the maximum round trip time RTT_{max} (200 ms), resulting in $l_{sock} = 1.5 \cdot 2 \cdot br_{max} \cdot RTT_{max} = 153600$ $bytes$.

The metrics for comparing the adaptive streaming systems are the reconstructed video quality in terms of PSNR and the deviation between the scheduled and the actual arrival time of a GOP on the client, $deltaC = t_i^{act} - t_i^{sched}$. For the DEADLINE approach the $deltaC$ has to be interpreted with care, because the approach has to wait for a GOP to be completely received before the decoding of the GOP on the client can be started. The reason for this is the reordering of the GOP and can be interpreted as additional buffering. In contrast to this, the APP-BE and TCP-BE approach can start the decoding of the GOP immediately after receiving the first frame of the GOP.
In our evaluation, we focus on the dependability of the adaptation algorithms in terms of timeliness of delivery. Thus, in the startup phase we transmit a single GOP in full quality to get a good starting point for the bandwidth estimation.

6. RESULTS

In our evaluation on adaptive Internet streaming, we conduct measurements for RTTs ranging from 50 to 200 ms. Because the streaming performance of TCP at low RTTs is very good, we decided to show the evaluation results for an RTT of 200 ms. In our opinion, this would represent the worst case scenario for our use case. In the following, a discussion of the evaluation results for each network scenario is presented.

The performance of the streaming systems in an *overprovisioned network* is shown in Figure 9. All approaches perform well and can stabilize the video quality by utilizing the additional bandwidth. This is supported by the analysis in [18], which states that good streaming performance can be expected when available bandwidth is about twice

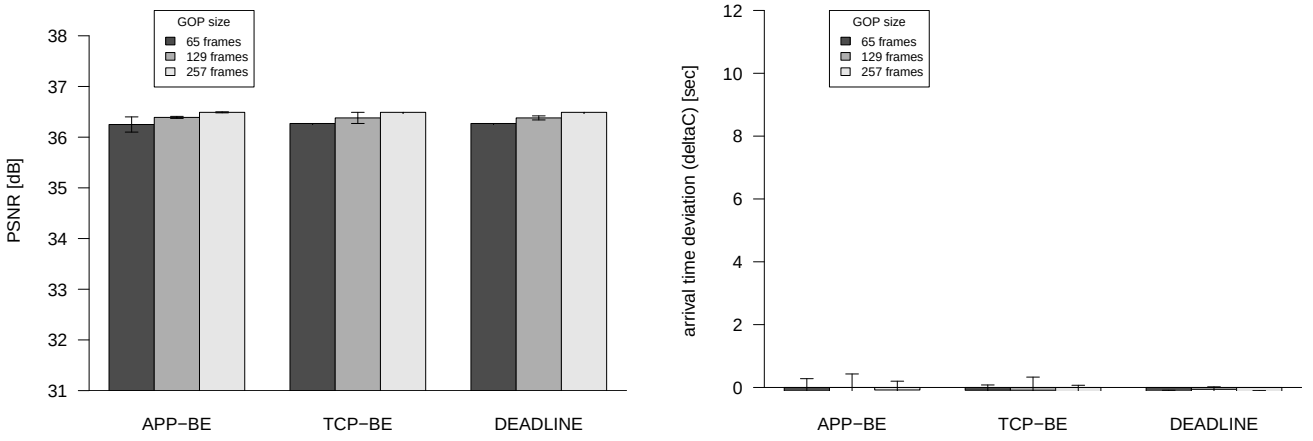

Figure 9: Overprovisioned network: $BW = 4096$ $kbps$. PSNR and arrival time deviation on the client ($deltaC$) for the different algorithms and GOP sizes at 200 ms RTT.

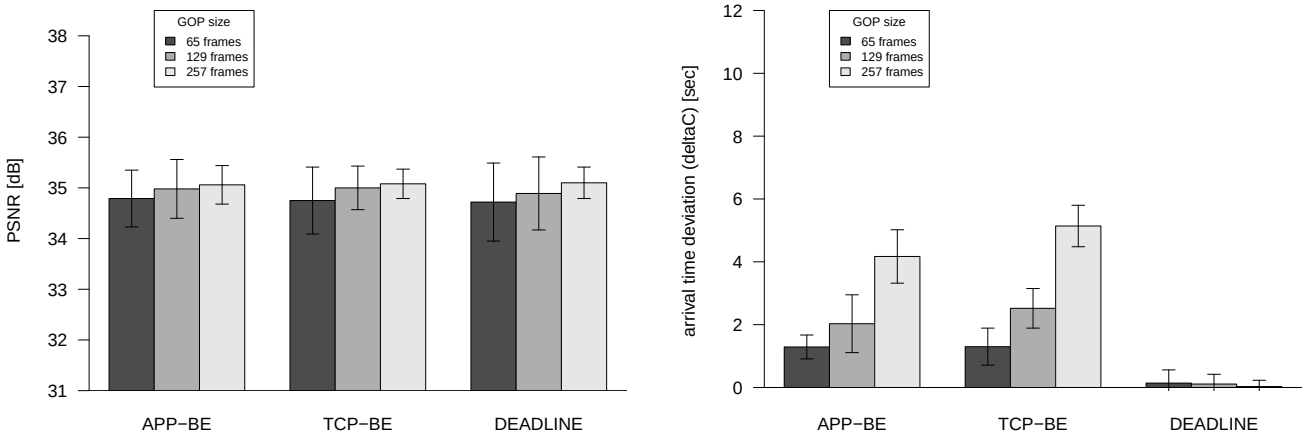

Figure 10: Underprovisioned network: $BW = 1536$ $kbps$. PSNR and arrival time deviation on the client ($deltaC$) for the different algorithms and GOP sizes at 200 ms RTT.

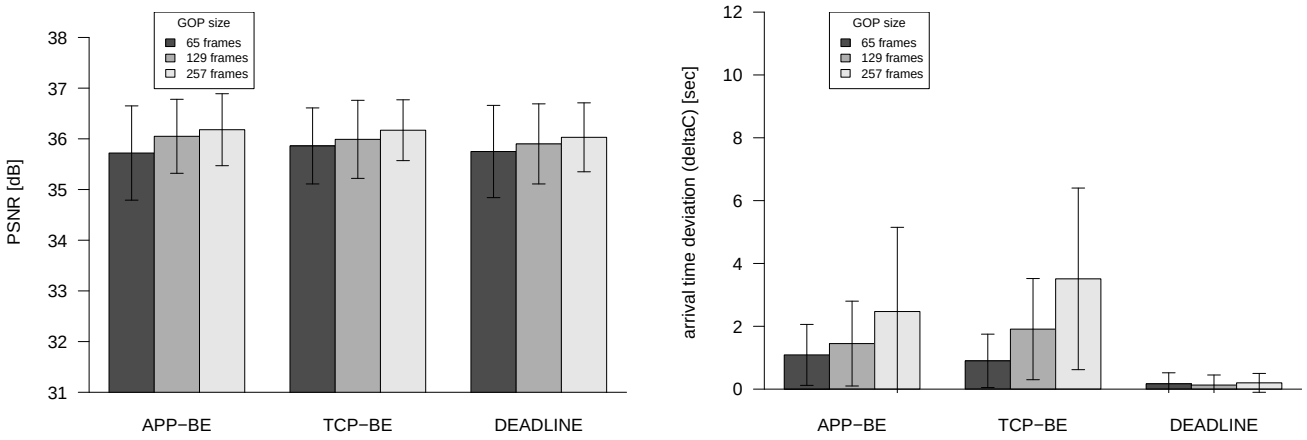

Figure 11: Congested network: $BW = 4096$ $kbps$ and one concurrent TCP stream. PSNR and arrival time deviation on the client ($deltaC$) for the different algorithms and GOP sizes at 200 ms RTT.

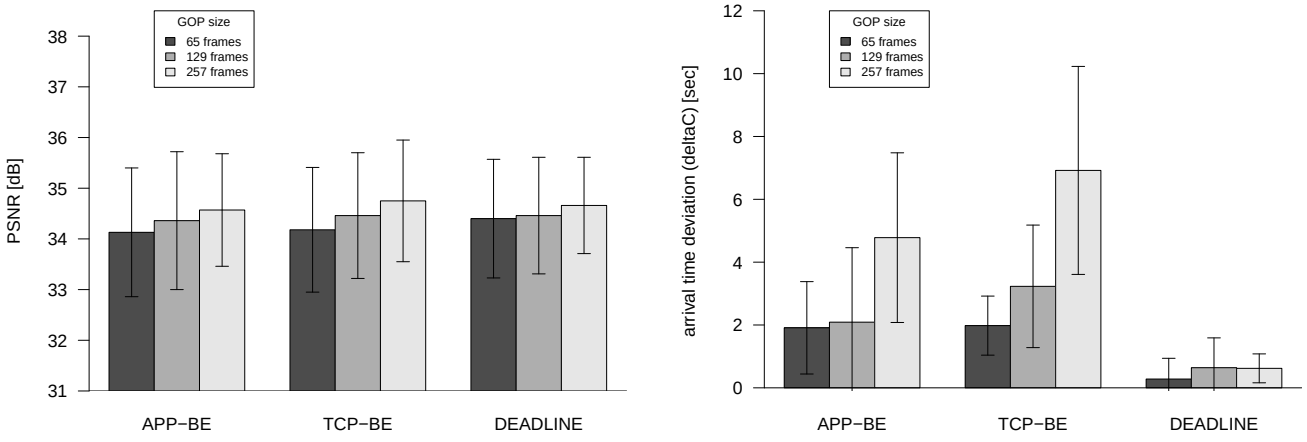

Figure 12: Congested network: $BW = 4096$ $kbps$ and two concurrent TCP streams. PSNR and arrival time deviation on the client ($deltaC$) for the different algorithms and GOP sizes at 200 ms RTT.

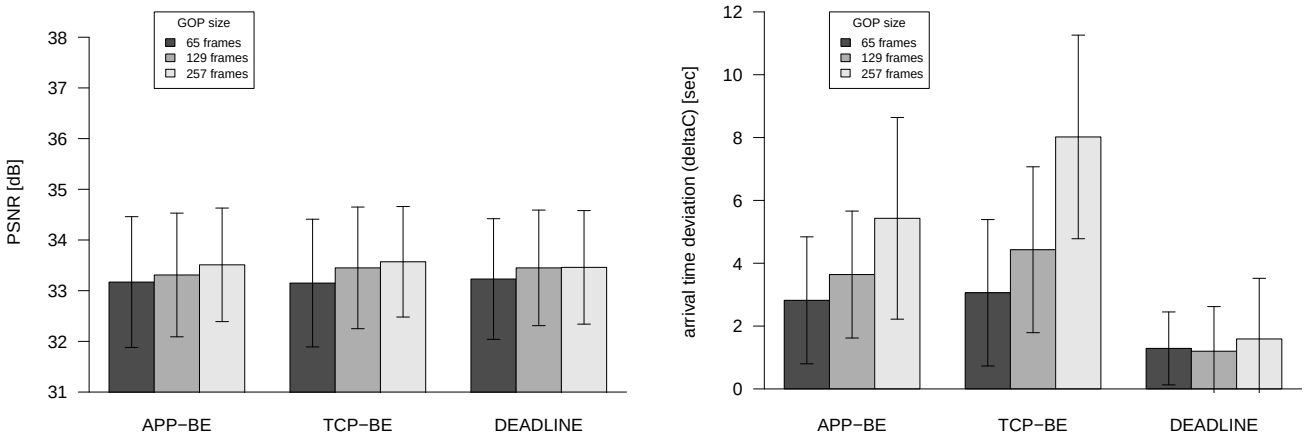

Figure 13: Congested network: $BW = 4096$ *kbps* and three concurrent TCP streams. PSNR and arrival time deviation on the client (*deltaC*) for the different algorithms and GOP sizes at 200 ms RTT.

the media bit rate. The small differences in the average PSNR for the different GOP sizes are mainly caused by the better rate-distortion performance of larger GOPs (see Figure 8). The variation of the emulated packet delay leads to TCP throughput variations, which result in the adaptation of some GOPs even in the case of overprovisioning. For the evaluation on the timeliness of delivery of the GOPs, we calculate the arrival time deviation of the GOPs on the client (*deltaC*) for the different algorithms and GOP sizes at 200 ms RTT. If the available bandwidth exceeds the bit rate of the adapted video, the delivery of a single GOP may take less time than the GOP playout duration. This can be observed in Figure 9, where GOPs arrive a little earlier than expected. The arrival time deviation is near zero, which means that all GOPs arrive in time and only little buffering will be needed at the client. The standard deviation of the PSNR and arrival time deviation is shown as error bars in the plots.

The adaptability of the streaming systems to a static bottleneck link with 1536 kbps bandwidth (*underprovisioned network*) is shown in Figure 10. The PSNR values in Figure 10 are approx. 35 dB, which corresponds to a bit rate of ≈ 1400 *kbps* (see Figure 8). This indicates that all streaming approaches can adjust their streamout rate to the available bandwidth. Also the larger GOP sizes help to stabilize the PSNR values, as can be seen from the lower standard deviation values. In bandwidth limited networks, the correctness of the bandwidth estimation is crucial. The figure shows that larger GOP sizes tend to increase *deltaC* for APP-BE and TCP-BE. This is mainly because for larger GOP sizes a wrong bandwidth estimate results in a larger deviation of the delivery duration, which directly influences *deltaC*. Although higher, the *deltaC* values for APP-BE and TCP-BE are in an acceptable range because of the buffer control at the server. The DEADLINE approach does not involve bandwidth estimation, so timely delivery can be achieved.

The evaluation results for congested network links are shown in Figures 11, 12 and 13. The average PSNR decreases the more congestion on the network link occurs. Also the quality changes between the GOPs increase (standard deviation) with the congestion level. The PSNR values for Figures 11, 12 and 13 are approx. 36, 34.5 and 33.5 dB, respectively. The PSNR values correspond to bit rates of 1800, 1250, 1000 kbps, respectively (see Figure 8). So for all congestion scenarios the approaches only use their fair share of the available link bandwidth. While all of the approaches are able to adapt to available/fair bandwidth, it can be observed that the DEADLINE approach can use larger GOPs to stabilize the overall quality of the video. The findings are similar to the results in the underprovisioned network scenario. DEADLINE can achieve a stable performance for all GOP sizes, while APP-BE and TCP-BE suffer from high variations at large GOP sizes.

The evaluation results indicate that the optimal GOP size for APP-BE and TCP-BE in our evaluation setup is 65 frames, because of the high arrival time deviation (*deltaC*) for large GOPs in case of congestion. In contrast to these approaches, DEADLINE is able to enhance its performance with larger GOP sizes, so a GOP size of 257 frames is considered to be optimal. Figures 14, 15 and 16 show the average PSNR of the adaptive streaming systems with respect to the RTT. In case of no congestion it can be observed that the quality is near constant. In the congested network scenarios, higher RTTs lead to lower PSNR values, which is mainly because of tough competition on the network link.

7. CONCLUSION

We evaluated and compared three approaches to adaptive TCP streaming (rate control) of H.264/SVC video in an Internet-like setting. To the best of our knowledge, this is the first study making use of H.264/SVC MGS scalability in such settings. We found that adaptive TCP streaming, despite the simplicity of the rate-control algorithms deployed, can effectively cope with bandwidth limitations and congestion, even under high RTTs. By utilizing H.264/SVC MGS-based scalability, the bit rate and quality of the delivered video data can be a adapted in a fine-grained man-

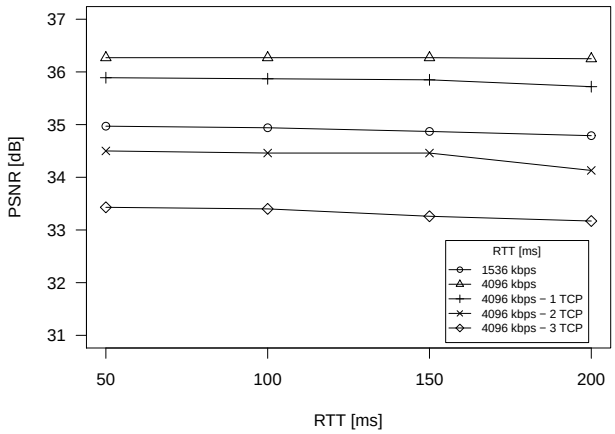

Figure 14: Average PSNR for APP-BE with GOP size 65 with respect to the RTT.

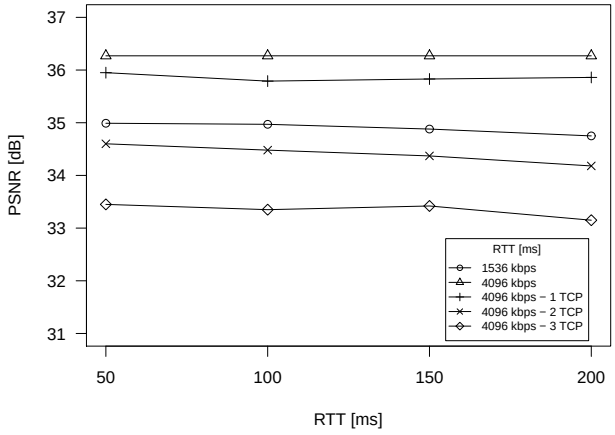

Figure 15: Average PSNR for TCP-BE with GOP size 65 with respect to the RTT.

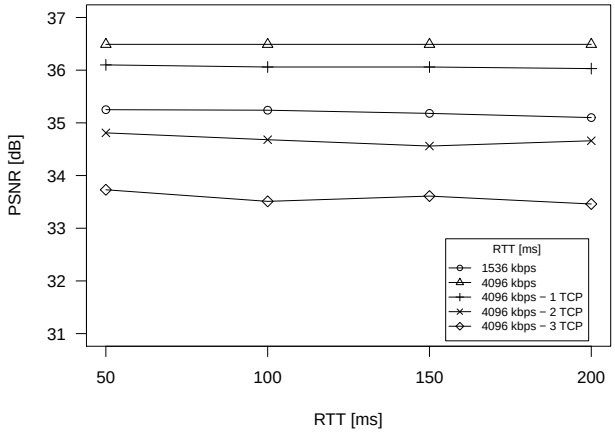

Figure 16: Average PSNR for DEADLINE with GOP size 257 with respect to the RTT.

ner. In order to stabilize TCP throughput over a period of time and decrease video quality fluctuations and packet jitter, our adaptive streaming approach utilizes rather large GOPs. We found, however, that the approaches using bandwidth estimation suffer from large GOP sizes, since inaccurate bandwidth estimates for GOP delivery may impact quality and increase jitter; client buffer dimensioning thus may become difficult. In contrast, the priority-/deadline-driven streaming approach that reorders and transfers – up to a pre-determined deadline – GOP picture data according to their importance, can cope with large GOP sizes and stabilize the quality in a rate-distortion optimal manner. Packet delay jitter is more predictable and client buffer sizes can be determined more easily. However, the client buffer must provide room for hosting and re-ordering a full GOP before playout can start. Thus, also startup delay increases by an additional GOP time. Another finding of our study was that bandwidth estimation on the application level can work quite well, comparably to making use of detailed information from the TCP stack. In case an operating system does not provide access to such information, the streaming server may resort to the application-level method.

8. ACKNOWLEDGMENT

This work was supported in part by the Austrian Science Fund (FWF) under project "Adaptive Streaming of Secure Scalable Wavelet-based Video (P19159)" and by the EC in the context of the P2P-Next project (FP7-ICT-216217).

9. REFERENCES

[1] J. G. Apostolopoulos, W.-T. Tan, and S. J. Wee. Video streaming: Concepts, algorithms, and systems. Technical report, HP Laboratories, 2002.

[2] A. Argyriou. Real-time and Rate-distortion Optimized Video Streaming with TCP. *Elsevier Journal on Signal Processing: Image Communication*, 22(4):374–388, 2007.

[3] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC '07)*, pages 43–56, 2007.

[4] S. Floyd and K. Fall. Promoting the Use of End-to-end Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[5] S. Floyd, M. Handley, and E. Kohler. Problem Statement for the Datagram Congestion Control Protocol (DCCP). RFC 4336 (Informational), 2006.

[6] A. Goel, C. Krasic, K. Li, and J. Walpole. Supporting Low Latency TCP-Based Media Streams. Technical report, Oregon Graduate Institute School of Science and Engineering, 2002.

[7] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[8] P.-H. Hsiao, H. T. Kung, and K.-S. Tan. Video over TCP with Receiver-based Delay Control. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '01)*, pages 199–208, 2001.

[9] Joint Video Team (JVT) of ISO/IEC MPEG and

ITU-T VCEG. Joint Scalable Video Model. *Doc. JVT-X202*, 2007.

[10] C. Krasic, K. Li, and J. Walpole. The Case for Streaming Multimedia with TCP. In *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS '01)*, pages 213–218, 2001.

[11] C. Krasic, J. Walpole, and W.-C. Feng. Quality-adaptive Media Streaming by Priority Drop. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '03)*, pages 112–121, 2003.

[12] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM - Computer Communication Review*, 27(3):67–82, 1997.

[13] Move Networks. Move Adaptive Stream - Product Sheet. http://www.movenetworks.com. Last accessed on 2009-09-22.

[14] R. Pantos. HTTP Live Streaming. Internet Draft draft-pantos-http-live-streaming-01, 2009.

[15] M. Prangl, I. Kofler, and H. Hellwagner. Towards QoS Improvements of TCP-Based Media Delivery. In *Proceedings of the Fourth International Conference on Networking and Services (ICNS '08)*, pages 188–193, 2008.

[16] M. Saxena, U. Sharan, and S. Fahmy. Analyzing Video Services in Web 2.0: A Global Perspective. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '08)*, pages 39–44, 2008.

[17] W. Feng, M. Liu, B. Krishnaswam, A. Prabhudev. A Priority-Based Technique for the Best-Effort Delivery of Stored Video. In *Proceedings of the SPIE/IST Multimedia Computing and Networking 1999 (MMCN'99)*, 1999.

[18] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. *ACM Transactions on Multimedia Computing, Communications and Applications*, 4(2):16:1–16:22, 2008.

[19] Y. Wang, M. Hannuksela, S. Pateux, A. Eleftheriadis, and S. Wenger. System and Transport Interface of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1149–1163, 2007.

[20] S. Wenger, Y.-K. Wang, T. Schierl, and A. Eleftheriadis. RTP Payload Format for SVC Video. Internet Draft draft-ietf-avt-rtp-svc-19, 2009.

[21] T. Wiegand, G. Sullivan, H. Schwarz, and M. Wien, editors. *ISO/IEC 14496-10:2005/Amd3: Scalable Video Coding*. International Standardization Organization, 2007.

[22] M. Wien, H. Schwarz, and T. Oelbaum. Performance Analysis of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1194–1203, 2007.

[23] W. Ye-Kui, M. M. Hannuksela, S. Pateux, A. Eleftheriadis, and S. Wenger. System and Transport Interface of SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1149–1163, 2007.

[24] E. Yildirim, D. Yin, and T. Kosar. Balancing TCP Buffer vs Parallel Streams in Application Level Throughput Optimization. In *Proceedings of the Second International Workshop on Data-aware Distributed Computing (DADC '09)*, pages 21–30, 2009.

APPENDIX

A. BUFFER CONTROL FUNCTION

The buffer control function is used for controlling the buffer by adjusting the estimated bandwidth. The rationale behind this is that without adjustment the scheduled start time and the actual start time of the transmission of each GOP tend to diverge. This is a consequence of the fact that the estimation of the available bandwidth is not perfect and over- or underestimates the actual value. On the other hand, if a TCP connection does not try to increase the throughput over time and does not participate in the competition of possible concurrent TCP connections, it will suffer from starvation.

The function that is given in Equations 3 and 4 is designed by considering the following situations. If the difference between scheduled and actual start time (*delta*) is less than 5 percent of the GOP duration in seconds, the bandwidth estimation is increased by 50 percent. This allows the algorithms to be more aggressive and competitive in case of low *delta* values. However, when *delta* exceeds five times the GOP duration, the actual transmission of the GOP lags significantly behind the scheduled transmission. In that case the actual rate for video transmission is chosen to be 1/5 of the estimated bandwidth. No adjustment takes place if *delta* is exactly 0.5 times of the GOP duration, i.e., the function value is 1. Based on these three operating points, a curve with the general form $f(x) = \frac{a}{x+b} + c$ is fitted to match the operating points described above. The parameters of the resulting function $\psi(x)$ are given in Equation 4.

$$f_{bctl}(delta, d^{gop}) := \begin{cases} 1.5 & \text{if } delta/d^{gop} < 0.05 \\ 1/5 & \text{if } delta/d^{gop} >= 5 \\ \psi(\frac{delta}{d^{gop}}) & \text{otherwise} \end{cases} \quad (3)$$

$$\psi(x) := \frac{1.46}{x + 0.893} + 0.0476 \quad (4)$$