Evaluation of HTTP-based Request-Response Streams for Internet Video Streaming

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

ABSTRACT

Adaptive video streaming based on TCP/HTTP is becoming popular because of its ability to adapt to changing network conditions. We present an in-depth experimental analysis of the use of HTTP-based request-response streams for video streaming. In this scheme, video fragments are fetched by a client from the server, in smaller units called chunks, potentially via multiple parallel HTTP requests (TCP connections). A model for the achievable throughput is formulated. The model is validated by a broad range of streaming experiments, including an evaluation of TCP-friendliness.

Our findings include that request-response streams are able to scale with the available bandwidth by increasing the chunk size or the number of concurrent streams. Several combinations of system parameters exhibiting TCP-friendliness are presented. We also evaluate the video streaming performance in terms of video quality in the presence of packet loss. Multiple request-response streams are able to maintain satisfactory performance, while a single TCP connection deteriorates rapidly with increasing packet loss. The results provide experimental evidence that HTTP-based request-response streams are a good alternative to classical TCP streaming.

Categories and Subject Descriptors

H.5.1 [**Multimedia Information Systems**]: Video

General Terms

Design, Experimentation, Performance

Keywords

HTTP video streaming, TCP-friendliness, H.264/SVC, adaptive video streaming

1. INTRODUCTION

The Internet and its applications are starting to play a main role in consumer electronics. To be able to connect a growing number and variety of devices, the network infrastructure has to be enhanced. As a result of ongoing investments, last-mile bandwidths increased considerably in recent years [4]. This also gave Internet video streaming, which heavily relies on network bandwidth, a lot of momentum. Video-on-Demand (e.g., online video rental), Live Video (TV broadcasts) or social video sharing are only some of the popular video streaming applications.

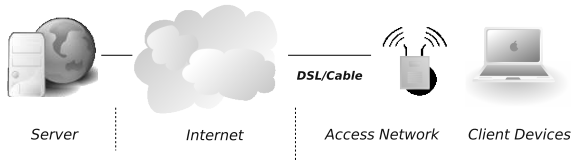

Figure 1: Use case

The Internet imposes new challenges on video streaming since it offers best-effort service only and lacks admission control and quality of service (QoS) mechanisms. Although the Internet does not provide admission control, some mechanisms have to be deployed to avoid network congestion. The transport protocol TCP is responsible for avoiding congestion by continuously adjusting the transmission rate at the sender based on feedback from the network. An example for TCP-based video streaming in the Internet is the video portal YouTube, which streams the content to consumers all over the world. The delivery of YouTube videos is almost centralized [15], which leads to round trip times (RTTs) of up to 200 ms and more. In this context, coping with packet losses which may occur due to congestion or packet corruption, becomes a challenge [8].

In our use case (see Figure 1), we focus on access networks which are assumed to form the bottleneck links. Currently the adaptation to different network conditions is done by the user who chooses from different qualities of the content, ranging from low resolution video to HD (e.g., 720p). Having the user select the video quality based on his/her experience may be cumbersome and error-prone, because the available bandwidth is in general not constant. *Adaptive* video streaming aims to simplify this process by continuously adapting the content bit rate to the available bandwidth of the network. Consequently, adaptive video streaming over TCP gained a lot of attention in recent years [19, 7, 23, 1].

In this paper, we analyze TCP/HTTP-based adaptive video streaming for use in the Internet (see Section 2). In particular, we investigate the performance of multiple paral-

MMSys'11, February 23–25, 2011, San Jose, California, USA.

lel HTTP-based request-response streams for video streaming. Previous work [13] showed that multiple streams show favorable characteristics in case of packet loss and mitigate the effects of TCP connection timeouts or stalls, while maintaining TCP-friendliness. In Section 3 we will present a refined analytical model (as compared to [13]) for the achievable throughput of a request-response streaming system. After giving a short introduction of H.264/SVC and priority streaming in Section 4, we present an HTTP-based request-response H.264/SVC video streaming system in Section 5. We will investigate how the system parameters of the request-response streaming system influence the system performance under diverse network conditions. The evaluation methodology, the investigated system, network parameters, and the evaluation setup will be described in Section 6. In Section 7, our findings on the throughput performance and TCP-friendliness are presented. A comparison in terms of video quality of using multiple HTTP request-response streams vs. a single TCP connection will show the actual benefit of the request-response streams. Section 8 concludes the paper.

2. TCP-/HTTP-BASED VIDEO STREAMING

While there are dedicated protocols for video streaming in IPTV networks (like RTP/UDP [16]), TCP (and HTTP over TCP) gained a lot of attention in the area of Internet video streaming due to its reliable end-to-end transport, the ability to adapt to changing network conditions, and easy deployment. Since TCP's reliability and adaptive behavior are based on acknowledgments and retransmissions, the performance of TCP depends a lot on the network conditions. Dynamically changing RTTs or packet losses lead to a degradation of the performance of TCP.

The throughput rate of TCP depends on the maximum segment size (MSS) and the round trip time (RTT). If we consider a packet loss pattern such that after the successful transmission of $1/p$ packets (of size MSS) one packet is lost, the *estimated TCP throughput rate* r_{tcp} for TCP Reno would be [9]:

$$r_{tcp} = \frac{MSS}{\sqrt{p}} \cdot \frac{1}{RTT} \quad (1)$$

With Equation 1 it becomes evident that the maximum throughput of TCP is limited by the packet loss rate p for a given RTT. The additive-increase/multiplicative-decrease (AIMD) behavior of the congestion control algorithm leads to an additional variation of the throughput. Thus TCP has been considered unsuitable for video streaming for many years. The streaming performance of TCP for constant-bitrate content was investigated in [19], which stated that the TCP throughput should be at least twice the media bit rate in order to avoid jerky playback of the video. With increasing bandwidth of the last-mile networks (a significant number of users own DSL/cable connections with downlink bandwidth greater than 4 Mbps [4]), this kind of over-provisioning is now feasible for low resolution videos. However, high-definition videos have in general higher bandwidth requirements ($>$ 2 Mbps), so over-provisioning may not be sufficient to prevent transmission stalls and therefore jerky playback. In this case, adapting the video content may be required to be able to cope with changing network conditions.

In the last couple of years, adaptive video streaming using TCP/HTTP has become quite popular. Microsoft introduced HTTP-based adaptive streaming with their Smooth Streaming System [23]. The idea is to use small HTTP progressive downloads (of so-called fragments) instead of a single one. Each fragment comprises several seconds of video data and is usually aligned with the GOP boundaries of the video [23]. In general, the fragments are downloaded consecutively and the video can be decoded and displayed on the client. With this approach, it is possible to switch the media bit rate (and hence the quality) after each download and adapt to the current network conditions. Move Networks [10] also uses similar mechanisms like Microsoft Smooth Streaming [23] for the media transport, and so does Apple's HTTP Live Streaming [11]. The 3GPP Adaptive HTTP Streaming standard [1] defines methods for adaptive media streaming. The Akamai HD Network [3] also features adaptive streaming of media content.

The adaptation system of all mentioned approaches works in a similar manner. Different video fragments are supplied which represent various qualities of the media. The video fragments are downloaded sequentially. If the available bandwidth does not allow to download the fragments of high quality, lower quality fragments are selected for download. The main difference between the approaches lies in the metadata for describing the media content, the container format for the media [14], and the adaptation decision taking algorithm, which decides when to switch from one media quality to another. Most of the streaming systems restrict their adaptiveness to stream switching (like shown in [17] for H.264/AVC), without taking advantage of the most recent scalable video codec H.264/SVC [22] which enables fine-grained adaptation. The transport of the video fragments is mainly based on HTTP's request-response paradigm, because using HTTP allows easy deployment in existing network infrastructures.

In our work we call a single HTTP connection transporting video fragments an *HTTP-based request-response stream*. While the streaming systems mentioned above are mainly based on a single request-response stream, we investigate the behavior of *multiple* concurrent HTTP request-response streams. In the next section, the HTTP-based request-response streams will be described in detail.

3. HTTP-BASED REQUEST-RESPONSE STREAMS

In classical TCP streaming, the media data is continuously streamed from the server to the client using a long-lived TCP connection. Because HTTP is based on the request-response (rr) paradigm, *HTTP-based request-response streams* have to request the media data to initiate their transmission. Request-response streams behave differently also in that the responses are in general small chunks of media data. While it is possible to emulate the behavior of a long-lived TCP connection in a request-response streaming setting by using very large chunks, the use of small chunks leads to short responses, which may experience unfairness in congested networks [8].

In general, TCP connections share the available bandwidth in a fair manner [8]. Aggregating multiple TCP streams for a single use is potentially unfair to concurrent single connections. We observed that it is possible to con-

trol the TCP-friendliness of request-response streams by introducing temporal gaps between requests (inter-request gap t_{gap}), which emulate an increased RTT [13]. Because TCP features no throughput fairness between connections with different RTTs [8], it is possible to aggregate multiple submissive request-response streams for a single purpose, while still providing TCP-friendliness. In addition, multiple request-response streams are not as prone to packet losses as a single TCP connection [13].

To get an of idea of the achievable throughput of such a request-response streaming system, we created a simple model describing the upper bound of the throughput. Assuming that a chunk (of size l_{ch}) is transferred within a single RTT and n_c concurrent streams are used for transmitting the media data, the upper bound for the throughput without packet loss $r_{rrsimple}$ can be calculated as follows [13]:

$$r_{rrsimple} = n_c \left(\frac{l_{ch}}{RTT + t_{gap}} \right) \tag{2}$$

Equation 2 shows that we are able to control the throughput by means of n_c, l_{ch} and t_{gap}. In addition, TCP-friendliness has to be supplied, so a trade-off has to be found between the throughput and the TCP-friendliness. The goal is to stabilize and enhance the overall throughput and to be more robust to changing network conditions than a single TCP connection, but also to be fair to other concurrent TCP connections in case of congestion.

Because the assumption that a chunk can be transferred within a single RTT is not valid for large chunk sizes, we extend our model by taking the network configuration into account. This extension allows us to explore the limitation of the request-response streams under certain network conditions. For that reason, we assume to know the bottleneck bandwidth BW and the maximum queueing delay t_q of the bottleneck router. Using BW and t_q we can calculate the queue size $l_q = BW * t_q$. Because the n_c concurrent TCP connections are competing on the network link, we assume that the router queue is shared between all request-response streams ($l_{rr} = l_q/n_c$). Taking into account that each request-response stream can transfer at most l_{rr} bytes per RTT, we can define the number of round trips needed to transmit the chunk as $n_{rt} = l_{ch}/l_{rr}$.

Because the underlying TCP connection tries to maximize the throughput, we can assume that the router queue will be fully utilized. For a single request-response stream, the AIMD algorithm of TCP leads to a saw tooth shaped queue utilization (see Figure 2), which results in a queuing delay of $t_q/2$ on average. Because multiple TCP streams tend to self-synchronize [2], we assume that the average queuing delay is also valid for multiple TCP streams. A request-response stream may not be able to fully utilize the network queue, if small chunk sizes and a low number of concurrent streams are used. In this case, we reduce the estimated queuing delay in our model, if n_{rt} is below one. So the average queuing delay of a request-response stream can be defined as $t_{qav} = min(n_{rt}, 1) \cdot t_q/2$. Using this information the *estimated transfer duration of one chunk* t_{ch} can be defined as:

$$t_{ch} = \lceil n_{rt} \rceil (RTT + t_{qav}) \tag{3}$$

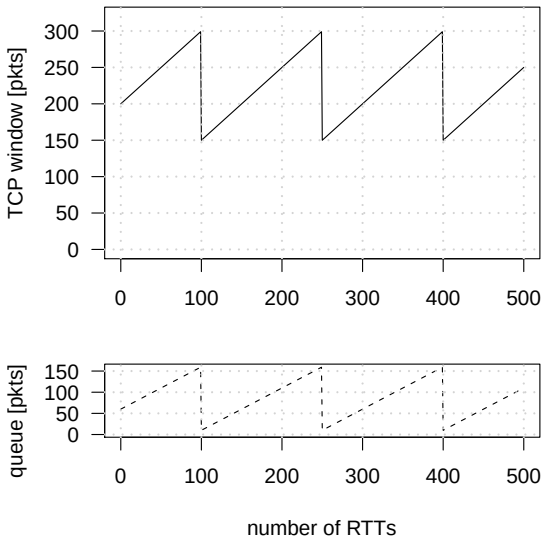

Figure 2: TCP window size and router queue utilization for a TCP flow through a router with a queue size of $l_q = BW * MAXRTT$ [2].

The *average achieved throughput* r_{rr} of the request-response streams can be defined as:

$$r_{rr} = n_c \left(\frac{l_{ch}}{t_{ch} + t_{gap}} \right) \tag{4}$$

Like in the initial model, we are able to tune the throughput by means of n_c, l_{ch}, and t_{gap} (see Equation 4). The main difference is now that this model takes the characteristics of the network into account (except packet loss).

A direct result of Equation 1 is that the data which TCP can transport within a single RTT is limited by the packet loss rate. So in case of packet loss, we have to define l_{rrloss}, which is the amount of data which can be transported by a request-response stream within one RTT. This is now either restricted by the router queue or the behavior of TCP in case of packet loss. Note that we are now using the same packet loss pattern as used in the TCP throughput estimation (see Equation 1).

$$l_{rrloss} = min(l_{rr}, \frac{MSS}{\sqrt{p}}) \tag{5}$$

The queuing delay is also affected by the packet loss, because the packet loss restricts TCP's window size and therefore the router queue utilization. Our model makes the simplifying assumption that if the utilized queue size in the router sinks below a certain level, the queuing delay begins to decrease. For that reason, we define the *average queuing delay under packet loss* $t_{qavloss}$ as:

$$t_{qavloss} = \begin{cases} \frac{l_{rrloss}}{l_{rr}} \cdot t_q/2 & \text{if } l_{rrloss} < \frac{l_{rr}}{2} \\ min(n_{rt}, 1) \cdot t_q/2 & \text{otherwise} \end{cases} \tag{6}$$

Because the number of round trips needed to transmit the chunk is also affected by the packet loss, we redefine it as $n_{rtloss} = l_{ch}/l_{rrloss}$. As a result, we can calculate the estimated transfer duration of one chunk under packet loss t_{chloss}:

$$t_{chloss} = \lceil n_{rtloss} \rceil (RTT + t_{qavloss}) \tag{7}$$

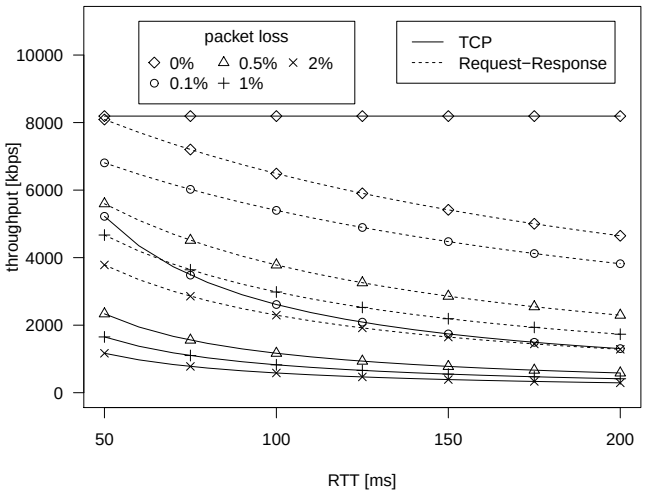

Figure 3: *Model* for throughput performance of a single TCP connection (TCP) r_{tcp} and for the request-response streams r_{rr} ($MSS = 1460\ bytes$, $l_{ch} = 160\ kB$, $n_c = 5$, $t_{gap} = 210\ ms$) at a fixed bottleneck bandwidth $BW = 8192\ kbps$ and packet loss rate p.

The throughput under packet loss for the request-response streaming system r_{rrloss} can be defined as follows:

$$r_{rrloss} = n_c \left(\frac{l_{ch}}{t_{chloss} + t_{gap}} \right) \tag{8}$$

This refined model for request-response streams jointly takes the packet loss and the queuing delay on the bottleneck router into account, by using the throughput estimation for TCP and the knowledge about the bottleneck router. In general, increasing the number of request-response streams or the chunk size leads to an increased throughput, but obviously also affects the TCP fairness and the computational effort needed to manage the multiple streams. In Figure 3, the upper bounds for a single TCP connection and the request-response streams (RR) according to Equations 1 and 8, respectively, are shown. While the performance of the single TCP connection highly depends on the packet loss and the RTT, the decline of the throughput of the request-response streams with increasing RTT and packet loss is significantly reduced. In Section 6, we will discuss the pros and cons of the request-response system parameters in detail and present a comparison of our model with real measurements. In the next section, we will briefly introduce the scalable video codec H.264/SVC and priority streaming, which will be used in our HTTP-based request-response streaming system (see Section 5).

4. H.264/SVC AND PRIORITY STREAMING

To investigate the video streaming performance of request-response streams in terms of video quality, we choose priority streaming based on H.264/SVC scalable content because this does not require an estimation of the available bandwidth and therefore makes additional buffer management at the server obsolete [12]. This simplifies the interpretation of the evaluation results on the streaming performance.

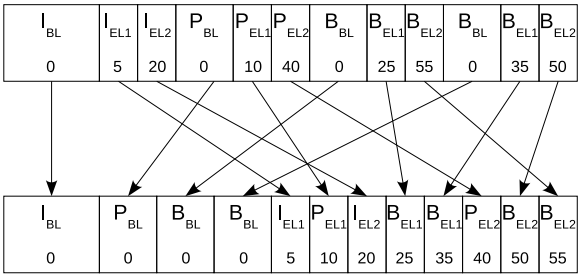

Figure 4: PRID-based NAL unit reordering

The H.264/SVC video coding standard [22] was recently standardized as an extension to the well-known H.264/AVC standard. 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. *Spatial scalability*, which allows the extraction of video sequences with different spatial resolutions, is also realized in a layered fashion. Finally, *quality scalability* is supported, which enables the adaptation to certain quality levels or bit rates. An H.264/AVC bitstream is structured into NAL (Network Abstraction Layer) units, which start with a one byte header. In order to signal the scalability information in the bit stream, SVC extends the NAL unit concept of H.264/AVC [20] (NAL unit header and types). Among others, the SVC 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 PRID 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 [6] 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. Using the PRID, we are able to extract up to 64 different qualities of the video content.

Unlike traditional buffering at the receiver, which tries to overcome bandwidth shortages, *priority streaming* [18, 7] levels the quality of the video over a period of time. For that reason, the video is split into fragments, each comprising the same play-out duration. In a next step, the video syntax elements (e.g., slices, frames, layers) are rearranged in order of priority. For example, using a non-scalable video codec, the I-frames would precede the P-frames and the B-frames. A reordered video fragment is called *video segment*. In priority streaming with H.264/SVC, the NAL units of each video fragment are rearranged according to their priority. In our case, a lower numerical value of the PRID signals a higher priority of the NAL unit according to [21]. The basic principles of this reordering are illustrated in Figure 4. 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 PRID 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,

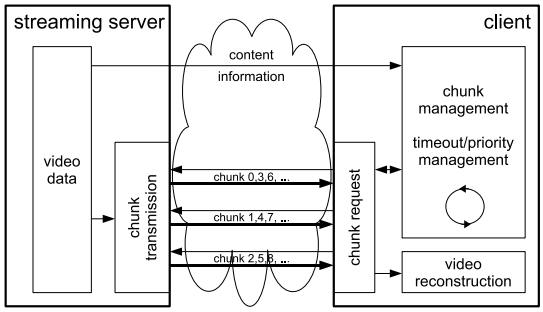

Figure 5: Request-response-based client-driven streaming system

the quality of the decoded bit stream at the client monotonically increases with the amount of data (NAL units) received by the client.

In the next section our HTTP-based request-response streaming system based on priority streaming and H.264/SVC will be explained in detail.

5. HTTP-BASED REQUEST-RESPONSE STREAMING SYSTEM

To investigate the video streaming performance of request-response streams, we designed a streaming system based on HTTP. The ability of the request-response streams to avoid transmission stalls and therefore jerky playback makes them a good candidate for video streaming. Multiple request-response streams are able to reduce quality fluctuations, while providing fairness to a single TCP connection. In addition, priority streaming is used to ensure timeliness of delivery.

The architecture of the *HTTP-based request-response streaming system* can be seen in Figure 5. Since the system uses HTTP, easy deployment, reuse of existing infrastructure (HTTP server, client, encryption, etc.) and application-layer multicast through HTTP proxies are possible. Persistent connections (as defined in HTTP/1.1 [5]) are used for establishing the TCP connections, in order to reduce the overhead of connection setup. Before streaming, the video is split into fragments, each comprising the same play-out duration. Each video fragment is rearranged according to the video quality / priority, resulting in a *video segment* (see Section 4).

The streaming system is characterized by three different parameters, namely the *chunk size* l_{ch}, the *number of concurrent streams* n_c and the *inter-request gap* t_{gap}. A video segment is split into *chunks* of size l_{ch}, which are served by a standard HTTP server. The download of the video chunks is coordinated by the client. For that purpose, the client maintains n_c HTTP-based request-response streams and schedules the downloads of the different chunks by using a separate queue for each stream as shown in Figure 6. Each chunk is retrieved by the client according to the order within the queue. Between consecutive chunk download requests, an inter-request gap t_{gap} is inserted to provide TCP-friendliness. On the client, the time used for downloading a segment is monitored. If the maximum time allocated for downloading a segment is reached (normally the the play-out duration of the segment), the client stops downloading chunks of the segment and switches to chunks of the next

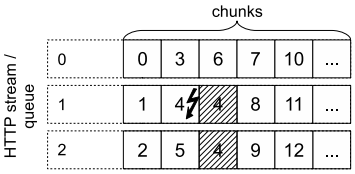

Figure 6: Three HTTP streams/queues on the client with priority management

segment. The client reconstructs the video from the received (and possibly truncated) video segment.

The streaming client coordinates the in-order transmission of the chunks and attempts to maximize the in-order throughput. To prevent transmission stalls, *timeout management* is used. A chunk is considered stalled if it is not retrieved within a timeout duration which can vary between 1000 ms and 5000 ms. Assuming that the maximum RTT is about 200 ms, which is common in Internet video streaming [15], at most 25 "attempts" ($25 * 200\ ms \approx 5000\ ms$) should be needed to transmit the video chunk. Very large chunks may exceed this limit, but we regard using such chunk sizes as unreasonable for video streaming. The lower bound of the transfer timeout is set to 1000 ms, to allow retransmissions on congested network links. The timeout is initialized as 3000 ms. The *transfer duration*, i.e., the time needed to download a chunk, is monitored for each chunk. A moving average over the last 20 transfer durations plus a tolerance of 30% is used for the calculation of the current transfer timeout. Expired transfers are considered (with a penalty) in the moving average as well, in order to supply the timeout management with early feedback on stalled transfers.

Priority management tries to improve the timeliness of the delivery by prioritizing video chunks required in the near future. In our case, the chunks will be needed by the client in priority order (as shown in Figure 6). If the transmission of a chunk is stalled (see original chunk 4 in Figure 6), it will be re-inserted into two queues (see hatched chunks), in order to increase the probability of a successful download. The priority management does not change the TCP-friendliness of the streaming system, because the number of concurrent HTTP streams is kept constant. Only the queues at the client used for fetching the chunks are updated, while the congestion control is ensured by the underlying TCP implementation of each single HTTP stream.

6. EVALUATION

In Section 5, we presented an HTTP-based video streaming system which makes use of request-response streams. The system offers many parameters to configure the HTTP-based streaming process, so it allows for an in-depth analysis of the streaming performance. Therefore, we will use this request-response streaming system as a basis for the evaluation of the streaming performance regarding the achievable throughput and the TCP fairness in different congestion scenarios.

Content

The test sequences used for evaluation were created by means of the H.264/SVC codec provided by the Joint Scalable Video Model (JSVM) [6] 9.18 software. The *soccer* sequence was encoded in 4CIF resolution at 30 fps and the

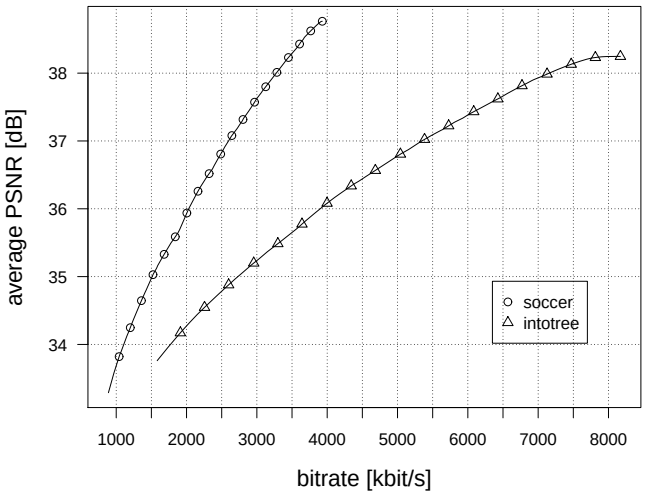

Figure 7: Rate-distortion curves of the test sequences

in to tree sequence in 720p resolution at 50 fps. 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 PRIDs to the NAL units based on rate-distortion values. In Figure 7, 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). The video quality is evaluated at a fixed bottleneck bandwidth of $BW = 8192kbps$ and two different test sequences. To show the influence of over-provisioning on the video quality, the sequences have a different maximum bit rate (see Figure 7). It can be observed that the maximum bit rate of the *in to tree* sequence is roughly twice the maximum bit rate of the *soccer* sequence. Each test sequence has a play-out duration of 10 seconds, which is also considered as the video fragment size. For the streaming experiments, the content is streamed 50 times in a loop.

Evaluation Setup

For the emulation of the Internet and the last-mile network, we use a test setup consisting of six Linux boxes. As shown in Figure 8, two servers are streaming the media data to two clients. Two routers are responsible for network emulation. The operating system is Ubuntu Linux (kernel 2.6.27) and on all computers the *TCP Reno* variant is used. On the routers, *Netem*[1] is used for the 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. Router 2 acts as the access network and limits the up- and downstream bandwidths (BW), allowing a maximum queuing delay of 200 ms. In addition, the packets are dropped in a random fashion to emulate packet loss on the transmission channel. Our implementation of the streaming system is based on Python and the HTTP library *libcurl*[2]. In the evaluation, Server 1 streams the media data

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

[2] http://curl.haxx.se

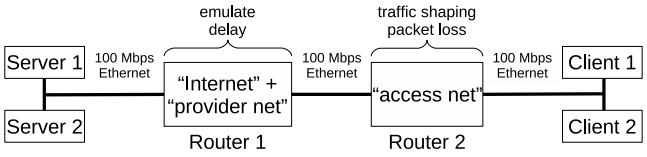

Figure 8: Evaluation setup

to Client 1. Congestion is emulated by concurrent HTTP downloads from Server 2 initiated by Client 2. The Apache HTTP Server[3] was used for serving the video chunks and the concurrent downloads.

System Parameters

The performance of the system is measured for the different system parameters (see Equation 4) and network scenarios. Different chunk sizes l_{ch} are used for the evaluation, namely 20, 40, 80, 160 and 320 kBytes (kB). The number of concurrent request-response streams n_c used is varied in a range of 1 to 5. The fairness of the request-response streaming system can be controlled via the inter-request gap parameter t_{gap}, as shown in [13]. On the other hand, the inter-request gap increases the transmission latency, therefore we choose it to be 220 ms at maximum. Each streaming evaluation run lasts for 500 seconds.

Network Scenarios

Because HTTP-based request-response streaming targets the Internet streaming use case, two different network scenarios are considered. First, we start the evaluation with an *uncongested network link*, where the streaming system should be able to make use of all the bandwidth. Second, we look at *congested network links* with different congestion levels, which are emulated by 1 to 4 concurrent TCP downloads. Two different bottleneck network bandwidths (4096 and 8192 kbps) and four different RTTs (ranging from 50 to 200 ms) were investigated. Given all the possible network and system parameters, a single evaluation run varying all parameters consists of 8000 streaming experiments. Each streaming experiment was repeated three times and the average performance values were used for the presentation of the results.

Performance Metrics

For the characterization of the streaming system, we use the *average throughput* which is measured for each video fragment (in our case every 10 seconds) and the *average download duration of the chunks*. We decided to use the average performance value of the different RTTs for presentation, because we want to show a single performance value for each system parameter set.

$$\frac{RR}{TCP} = \frac{r_{rr}}{\frac{1}{n_{tcp}} \sum_{i=1}^{n_{tcp}} r_{tcpi}} \tag{9}$$

The *TCP fairness ratio* RR/TCP of the request-response streams (see Equation 9) in a congested network should give us a good idea of how to choose the system parameters to achieve TCP fairness for a given bottleneck bandwidth of

[3] http://httpd.apache.org

the last-mile link. A fairness ratio value $RR/TCP = 1$ indicates that the request-response streams only use their fair share of the available bandwidth. The request-response streams are potentially unfair to concurrent TCP streams if the ratio is larger than one ($RR/TCP > 1$), while a ratio of $RR/TCP < 1$ indicates that the request-response streams are only using less than their fair share. For the presentation of the fairness ratio, we average the values for the different congestion levels (number of competing downloads), in order to get a single value for a specific system parameter set. In addition, we will show how our model of Section 2 correlates with the measured results for the averaged performance values and a specific system parameter set ($l_{ch} = 160\ kB$, $n_c = 5$, $t_{gap} = 210\ ms$ at a fixed bottleneck bandwidth $BW = 8192\ kbps$), which provides TCP fairness.

In the next section, the results of our evaluation are presented.

7. RESULTS

In our evaluation of video streaming with HTTP-based request-response streams, we conducted a broad range of experiments. We measured the average throughput, the average download duration of the chunks, and evaluated the fairness of the streaming solution. To show our findings in a concise manner, we choose to average the results over the different RTTs.

In the first part of this section, we discuss the influence of the three system parameters, namely the chunk size l_{ch}, the number of concurrent request-response streams n_c and the inter-request gap t_{gap}. Figures 9 and 10 show the average throughput performance and the average download duration of a chunk for the bottleneck bandwidths $BW = 4096\ kbps$ and $BW = 8192\ kbps$, respectively. The evaluation took place in an uncongested network. In general, it can be noticed that a single request-response stream cannot make full use of the available bandwidth, because of the nature of the request-response paradigm and the inter-request gap. The influence of the system parameters under consideration is discussed as follows.

Chunk size: Larger chunk sizes use the available network bandwidth more effectively. This can be noticed in Figure 10, when comparing the chunk sizes 20 kB and 160 kB. While the 20 kB chunks cannot fully utilize the network link with 5 streams, the 160 kB chunks are able to make good use of the available bandwidth with even 3 streams. Request-response streams with very large chunks behave similarly to single TCP connections. This leads to an increased throughput performance, but also to congestion in case of multiple streams. Because of the fixed bandwidth limit BW, in uncongested networks the download duration increases linearly with the chunk size (e.g., with a single stream $t_{dur} = l_{ch}/BW$). Yet, if the available bandwidth is exhausted, enlarging chunks will not lead to an increased throughput anymore.

Number of streams: The number of request-response streams shows a behavior similar to the chunk size. In Figure 9, at a chunk size of 20 kB, a higher number of streams leads to higher throughput. Multiple streams are also more error resilient (see Section 3), but tend to generate self-congestion, because each request-response stream is based on TCP which tries to maximize the throughput. If increasing the number of streams does not lead to a substantial throughput increase, we can assume that self-congestion takes place (see chunk size 320 kB in Figure 9). As a result of this, also the download duration increases with the number of streams. We observed that one can safely increase the number of streams as long as the download duration does not increase in a linear fashion.

Temporal gap: The inter-request gap parameter has obviously no positive effect on the throughput or download durations. Because of the artificial gap between the requests, the time the request-response streams are not transporting data is increased. Therefore, smaller inter-request gaps lead to higher throughput. In general, the inter-request gap has no influence on the download duration, because the inter-request gap is applied between the downloads. In addition, Figure 9 shows that the inter-request gap has a higher influence on small chunks, because the inter-request gap is applied between consecutive downloads of chunks and larger chunks need longer to be transferred.

In Figures 9 and 10 we also show the values of our model for the estimated throughput. The model shows a good correlation with the measured values and can give a good hint on the achievable throughput. At large chunk sizes and heavy self-congestion, the model seems to increase its error, but can predict the upper bound of the throughput. The same behavior can be noticed for the download duration.

For the evaluation of TCP fairness, we start 1 to 4 concurrent HTTP downloads to the request-response streaming system. By changing the number of concurrent downloads, we are able to adjust the network congestion to different levels. For the HTTP downloads and the streaming system, the throughput values are recorded and the *TCP fairness ratio* RR/TCP is calculated (see Equation 9). We choose to average the fairness ratios for the different RTTs and congestion levels, to get a single value for the fairness of a specific parameter set. In Figure 11, the TCP fairness of the request-response streaming system for the three system parameters is shown. The value $RR/TCP = 1$ is marked, because it shows under which conditions TCP fairness can be achieved. It can be seen that all three parameters have an influence on the TCP fairness. While the fairness decreases (RR/TCP increases) with the number of streams and the chunk size, an increasing inter-request gap is able to increase the fairness. For very large chunks (e.g., 320 kB at $BW = 4096\ kbps$), it becomes increasingly difficult to find a fair parameter set with multiple streams. This is because the inter-request gap has in general an upper bound defined by the use case envisioned (in our case 220 ms).

Another interesting finding can be noticed when directly comparing the fairness ratio plots for $BW = 4096\ kbps$ and $BW = 8192\ kbps$. The fairness ratio values of a certain chunk size in $BW = 4096\ kbps$ behave similarly to the fairness ratio values of the doubled chunk size in $BW = 8192\ kbps$. In our opinion, this may be an indicator for the scalability of the request-response streaming system. If more bandwidth is available, using larger chunk sizes may lead to a better link utilization without hurting TCP fairness. Further work will look into that presumption in more detail to explore the scalability of request-response streams.

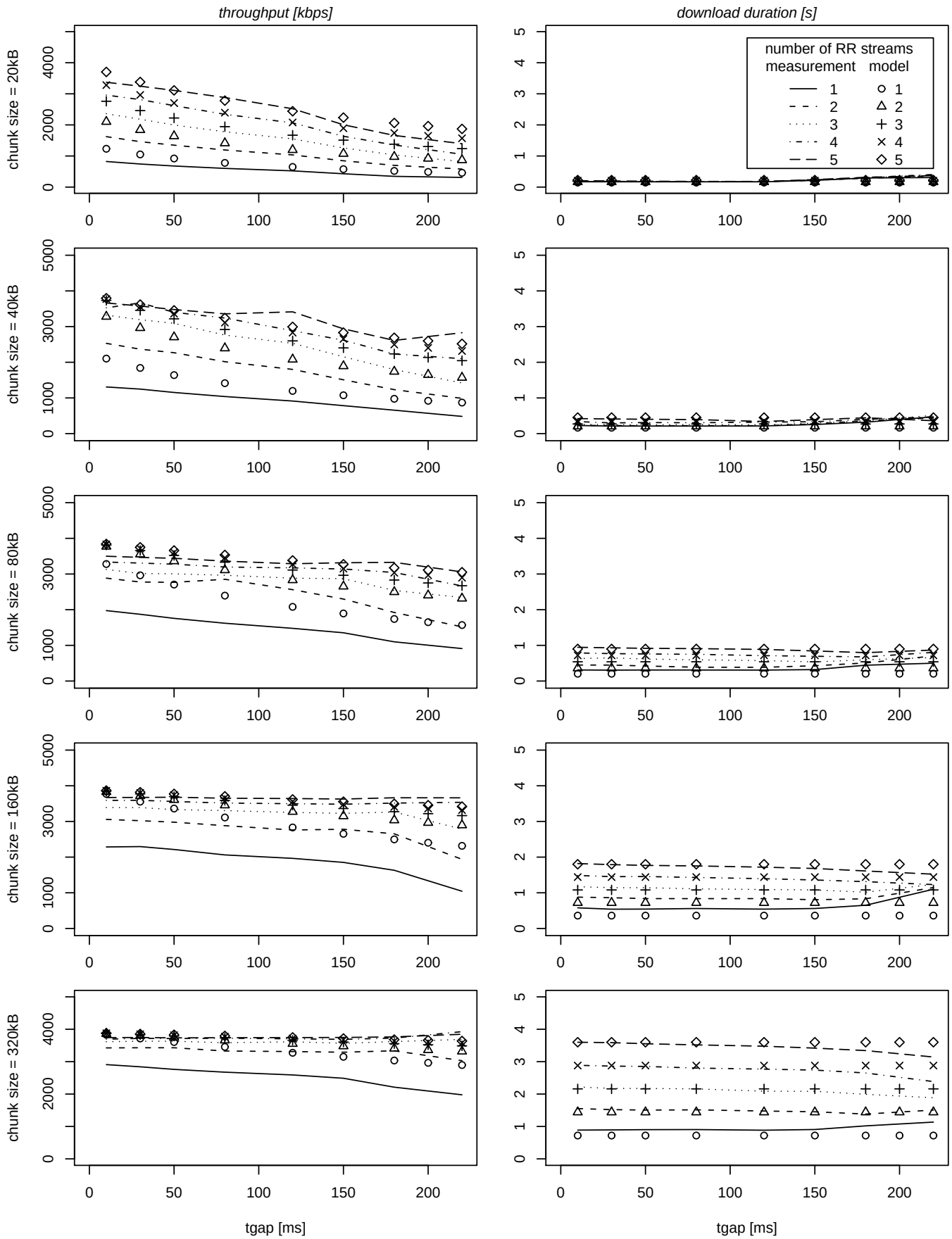

Figure 9: Measured and modeled throughput performance r_{rr} and download duration of a chunk t_{ch} for the request-response streams. The results are shown for a fixed bottleneck bandwidth of $BW = 4096\ kbps$, a maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs to provide a single performance value.

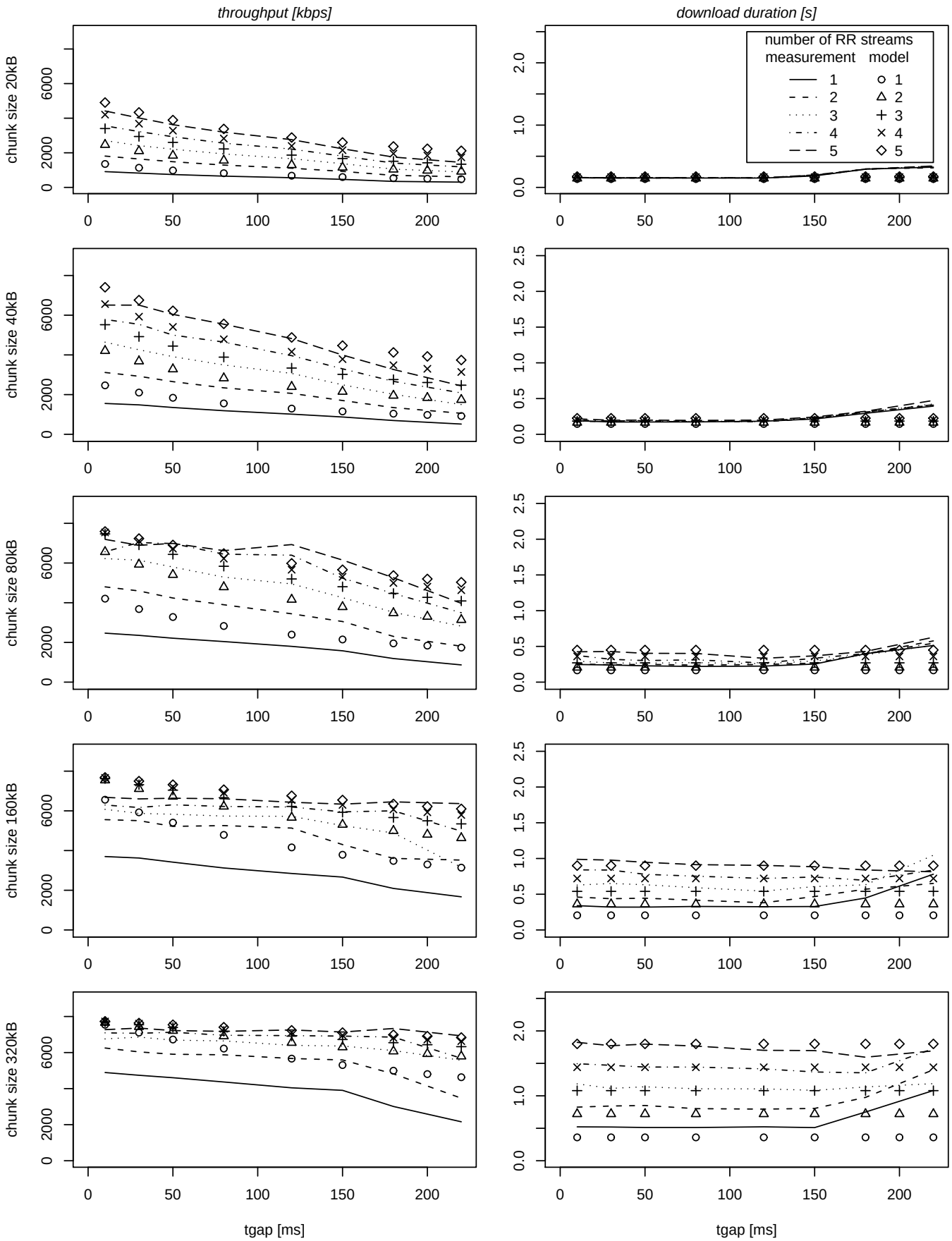

Figure 10: Measured and modeled throughput performance r_{rr} and download duration of a chunk t_{ch} for the request-response streams. The results are shown for a fixed bottleneck bandwidth of $BW = 8192\ kbps$, a maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs to provide a single performance value.

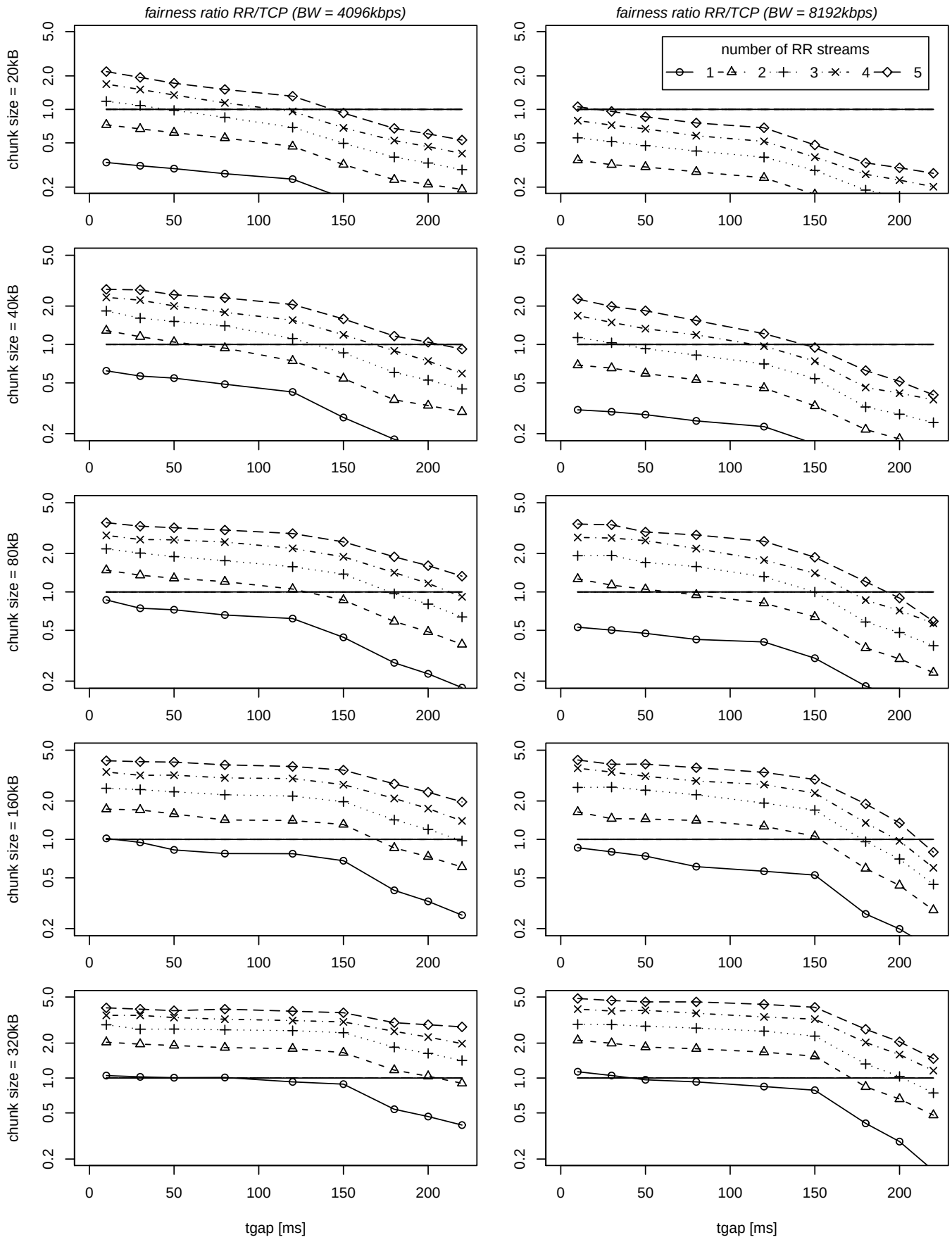

Figure 11: Measured TCP fairness ratio RR/TCP of the request-response streams compared to 1 . . . 4 concurrent HTTP downloads. The results are shown for two bottleneck bandwidths ($BW = 4096\ kbps$ and $8192\ kbps$ and maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs and different congestion levels to provide a single performance value.

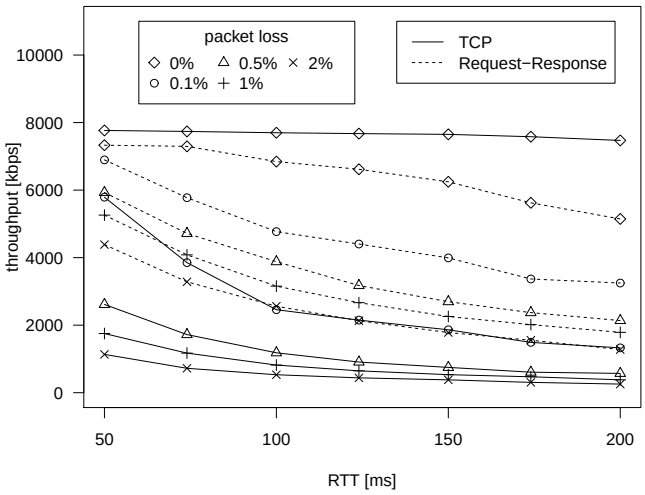

Figure 12: *Measured* throughput performance of a single TCP connection (TCP) r_{tcp} and for the request-response streams r_{rr} using the same conditions as used in Figure 3.

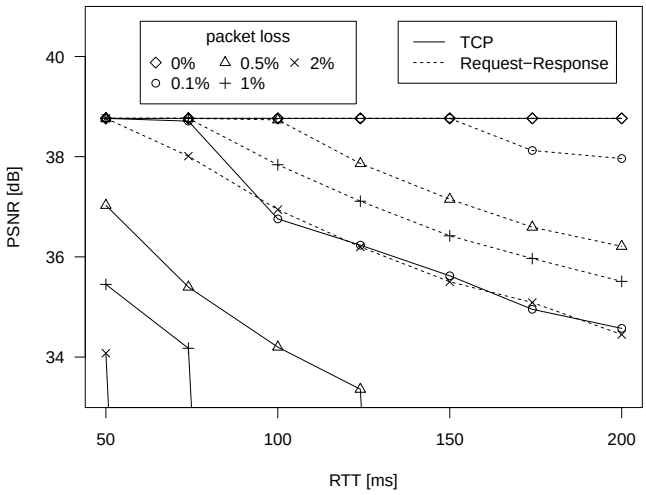

Figure 13: Measured video quality of the streamed test sequence *soccer* in terms of PSNR of a single TCP connection (TCP) and for the request-response streams ($MSS = 1460\ bytes$, $l_{ch} = 160\ kB$, $n_c = 5$, $t_{gap} = 210\ ms$) at a fixed bottleneck bandwidth $BW = 8192\ kbps$ and packet loss rate p.

When using request-response streaming systems in the Internet, the streaming system should provide TCP fairness ($RR/TCP = 1$). Because there is more than a single parameter set which provides TCP fairness, a good trade-off between throughput performance and TCP fairness has to be found. The number of request-response streams also directly impacts the performance in case of packet loss or transmission stalls. Therefore, we propose to use a parameter set with a large number of streams which offers good throughput.

In Figure 12, the performance of such a parameter set for a fixed bottleneck bandwidth of $BW = 8192\ kbps$ is shown. The chunk size was set to 160 kB and 5 request-response streams with an inter-request gap of 210 ms were used for the transmission of the video data. In direct comparison with the performance of a single TCP connection, the benefit of the multiple streams becomes evident. While the throughput of the single TCP connection rapidly decreases with increasing packet loss, the request-response streams are able to mitigate the effect. Also, the performance in an uncongested network with no packet loss ($p = 0\ \%$) is quite good, but of course not as good as TCP, because of the fixed inter-request gap value. When comparing the values of the measurement with the model (see Figures 12 and 3), it can be noticed that the correlation between the model and the real measured values is good. We know that our model is only a simplification of the real world problems, but we think it can be used for an estimation of the throughput performance of the request-response streams (with and without packet loss).

The impact of the streaming performance on the video quality shows a behavior similar to the throughput performance. Figures 13 and 14 present the video quality in terms of PSNR for the test sequences *soccer* and *in to tree*, respectively. In case of no packet loss and stable network conditions, TCP and the request-response streams perform well, with an advantage for TCP in terms of network utilization. If packet loss is present, then this fact changes dramatically. For high packet loss rates, the single TCP connection cannot even transmit the base layer of the video, because of insufficient TCP throughput. High RTTs and the HD video *in to tree* present difficulties for the request-response streams as well. It is quite evident, though, that the request-response streams can make better use of the available bandwidth, while the performance of the single TCP stream is stuck (see Equation 1). Unfortunately, the difference in PSNR values can only be calculated for a packet loss rate of $p = 0.1\ \%$ and the test sequence *soccer*, which is at average 2.04 dB. In the other network scenarios with packet loss and for the sequence *in to tree*, TCP cannot deliver the video at every given RTT. So the main advantage of the request-response streams compared to a single TCP connection is that it can deliver the video, while TCP may not be capable of delivering the video at all.

8. CONCLUSION

Internet video streaming has gained popularity mainly due to social video portals and video-on-demand applications. Because of the dynamic nature of the Internet, constant bit rate video streaming based on TCP is only possible with high over-provisioning [19]. Adaptive video streaming based on TCP/HTTP is certainly the key to Internet video streaming and a lot of systems were proposed [23, 10, 3, 1, 11].

HTTP streaming using video fragments is usually more robust against network fluctuations and is basically stateless on the server-side, because it is in general fully client-driven. Compared to a single TCP connection, request-response streams can mitigate the effects of packet loss and TCP connection timeouts or stalls. In our work, we analyzed the basic properties of HTTP streaming. By introducing HTTP-based request-response streams, we were able to identify the system parameters of HTTP-based streaming. Using knowledge about the bottleneck router led us to a model for the estimated throughput of the request-response streams.

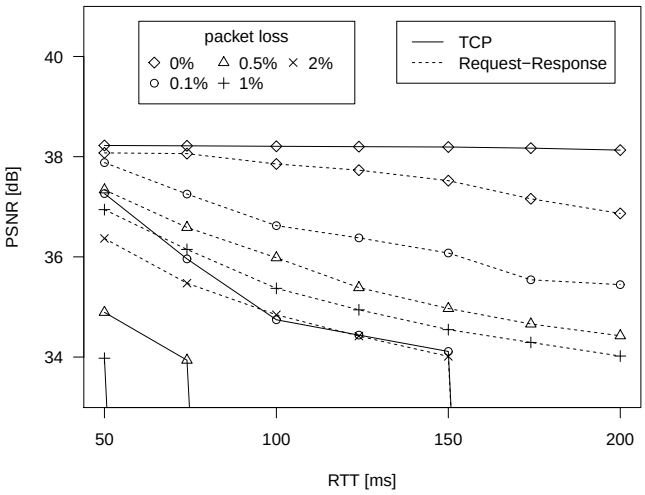

Figure 14: Measured video quality of the streamed test sequence *in to tree* in terms of PSNR of a single TCP connection (TCP) and for the request-response streams ($MSS = 1460\ bytes$, $l_{ch} = 160\ kB$, $n_c = 5$, $t_{gap} = 210\ ms$) at a fixed bottleneck bandwidth $BW = 8192\ kbps$ and packet loss rate p.

An in-depth evaluation of the performance of request-response streams was presented in this paper. It revealed that request-response streams are able to scale with the available bandwidth by increasing the chunk size or the number of concurrent streams. The system parameters have to be carefully chosen, however, based on the bandwidth of the bottleneck router in order to avoid self-congestion. Our proposed model for the throughput shows a good correlation with the experimental results. Regarding TCP-friendliness, we were able to show that there are several combinations of system parameters which exhibit TCP-friendliness. By means of a temporal inter-request gap, we were able to adjust the fairness of a certain chunk size / number of streams combination to a fair level. For such a fair system parameter set, we evaluated the video streaming performance in terms of video quality in the presence of packet loss. While the single TCP connection performs better than the request-response streams if no packet loss is present, the performance of the single TCP connection deteriorates rapidly with increasing packet loss.

In this paper, we provided experimental evidence that multiple HTTP-based request-response streams are a good alternative to classical TCP streaming. Additionally, we presented several system parameter sets featuring TCP-friendliness, which can be used in the design of TCP-friendly streaming systems. Currently the performance of the request-response streams is limited by their fixed parameter settings. Future work will cover an investigation of dynamic parameter sets and their TCP-friendliness.

9. ACKNOWLEDGMENT

This work was supported in part by the European Commission in the context of the Integrated Project ALICANTE (FP7-ICT-248652; http://www.ict-alicante.eu).

10. REFERENCES

[1] Transparent End-to-end Packet-switched Streaming Service; Protocols and codecs. ETSI TS 126 234, V9.3.0, June 2010.

[2] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.*, 34:281–292, August 2004.

[3] L. De Cicco and S. Mascolo. An Experimental Investigation of the Akamai Adaptive Video Streaming. In *Proceedings of the 6th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering (USAB)*, pages 447–464, Nov. 2010.

[4] 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.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999.

[6] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. Joint Scalable Video Model. *Doc. JVT-X202*, 2007.

[7] 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.

[8] M. Hassan and R. Jain, editor. *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*. Pearson Prentice Hall, 2004.

[9] 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.

[10] Move Networks. Move Adaptive Stream - Product Sheet. http://www.movenetworks.com. Last accessed on 2010-12-07.

[11] R. Pantos. HTTP Live Streaming. Internet Draft draft-pantos-http-live-streaming-04, 2009.

[12] R. Kuschnig, I. Kofler, and H. Hellwagner. An Evaluation of TCP-based Rate-Control Algorithms for Adaptive Internet Streaming of H.264/SVC. In *Proceedings of ACM Multimedia Systems (ACM MMSYS 2010)*, February 2010.

[13] R. Kuschnig, I. Kofler, and H. Hellwagner. Improving Internet Video Streaming Performance by Parallel TCP-based Request-Response Streams. In *Proceedings of the 7th Annual IEEE Consumer Communications and Networking Conference (IEEE CCNC 2010)*, January 2010.

[14] H. Riiser, P. Halvorsen, C. Griwodz, and D. Johansen. Low Overhead Container Format for Adaptive Streaming. In *Proceedings of ACM Multimedia Systems (ACM MMSYS 2010)*, pages 193–198, Feb. 2010.

[15] 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.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[17] T. Stockhammer, G. Liebl, and M. Walter. Optimized H.264-AVC-based bit stream switching for mobile video streaming. *EURASIP J. Appl. Signal Process.*, 2006:1–19, 2006.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] A. Zambelli. IIS smooth streaming technical overview. Technical report, Microsoft Corporation, March 2009.