Dipl.-Ing. Robert Kuschnig

# Congestion-Aware Quality-Adaptive Streaming of Scalable Video

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der technischen Wissenschaften

Studium: Informatik

Alpe-Adria Universität Klagenfurt
Fakultät für Technische Wissenschaften

| | |
|---|---|
| 1. Begutachter: | Univ.-Prof. Dipl.-Ing. Dr. Hermann Hellwagner |
| Institut: | Institut für Informationstechnologie |
| | Alpe-Adria Universität Klagenfurt |
| 2. Begutachter: | Prof. Dr.-Ing. Wolfgang Effelsberg |
| Institut: | Lehrstuhl für Praktische Informatik IV |
| | Universität Mannheim |

Juli/2012

# Declaration of honour

I hereby confirm on my honour that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisor has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

I am aware that a false declaration will have legal consequences.

Klagenfurt, Juli/2012

# Acknowledgements

I would like to thank my supervisor professor Hermann Hellwagner for giving me the opportunity and financial support to conduct this research. He encouraged me to focus on TCP-based adaptive video streaming, which was - till recently - seen controversial even for Internet deployments.

In addition, I want to acknowledge professor Wolfgang Effelsberg for giving me important feedback on my thesis. The discussions helped to pinpoint the research focus and the contributions made.

Finally, I want to thank all colleagues at ITEC - especially Ingo Kofler - for their fruitful discussions and our joint work.

# Kurzfassung

Während das Internet anfangs hauptsächlich für die Verbreitung von Informationen genutzt wurde, konnten mit der flächendeckenden Einführung von Breitband-Internet-Anschlüssen auch neue Geschäftsfelder erschlossen werden. Eine dieser neuen Anwendungen ist Internet-Video-Streaming. Doch die Übertragung großer Datenmengen über das Internet ist nicht unproblematisch, da es im Internet keine Möglichkeiten zur Sicherstellung von Dienstgüte gibt. Die daraus resultierenden Probleme sind große Schwankungen in der verfügbaren Bandbreite und der Paketlatenz, Paketverlust und Überlastung des Netzwerkes. Diese Rahmenbedingungen erschweren einen reibungslosen Videokonsum und machen Internet-Video-Streaming zu einem sehr interessanten Forschungsgebiet. Das Transmission Control Protocol (TCP) ist das Standardprotokoll im Internet für zuverlässige Datenübertragung mit Überlaststeuerung. Im Besonderen wurde das Design von TCP auf die Bedürfnisse der zuverlässigen Datenübertragung im Internet abgestimmt. Dieser Umstand verlieh TCP-basiertem Internet-Video-Streaming Auftrieb.

Aus diesem Grund konzentriert sich diese Arbeit auf TCP-basiertes Video-Streaming für das Internet. Das Ziel ist es, Video-Streaming in überlasteten und fehlerbehafteten Netzwerken zu verbessern. Die Arbeit umfasst sechs wissenschaftliche Beiträge, welche im Folgenden näher erläutert werden. (1) Drei adaptive Streaming-Verfahren basierend auf TCP wurden evaluiert, um die Leistungsfähigkeit von TCP-basiertem Streaming unter diversen Netzwerkbedingungen zu bestimmen. Die Ergebnisse dieser Evaluierung dienen als Referenz für weitere Untersuchungen. (2) Untersuchung über den Einfluss von Paketverlust auf die Leistungsfähigkeit von TCP-basiertem Video-Streaming. (3) Ein neues Video-Streaming-Verfahren basierend auf dem Anfrage-Antwort-Prinzip wurde entworfen, um die Einschränkungen von TCP-Streaming zu umgehen (die Leistungsfähigkeit von TCP-Streaming verschlechtert sich rapide mit steigendem Paketverlust). Die Neuerung dieses HTTP-basierten Verfahrens ist, dass die Fairness gegenüber TCP akzeptabel bleibt, obwohl gleichzeitig mehrere Datenströme für die Übertragung verwendet werden. (4) Ein Modell des HTTP-basierten Anfrage-Antwort Streaming-Verfahrens wurde entwickelt, welches die Leistungsfähigkeit des Verfahrens für unterschiedliche Systemparameter und Netzwerkbedingungen beschreibt. (5) Um die Gültigkeit des Modells zu bestätigen, wurde die Leistungsfähigkeit des Anfrage-Antwort-Streaming-Verfahrens unter diversen Netzwerkbedingungen evaluiert. Zusätzlich wurde die Fairness gegenüber TCP gemessen. Die Systemparameter des Streaming-Verfahrens können so konfiguriert werden, dass Fairness gegenüber TCP gewährleistet werden kann. (6) Ein mobiles Video-Streaming Szenario mit großen Bandbreitenschwankungen und RTTs wurde genutzt, um die Leistungsfähigkeit des HTTP-basierten Anfrage-Antwort-Streaming-Verfahrens unter schwierigen Netzwerkbedingungen zu ermitteln. Das Verfahren nutzt die verfügbare Bandbreite effizient und kann die Anzahl der Qualitätsänderungen gering halten.

Das HTTP-basierte Anfrage-Antwort-Streaming-Verfahren vereint die gesamte Streaming-Logik im Streaming-Client und schafft es dadurch, die Komplexität des Streaming-Verfahrens zu minimieren. Zusätzlich kann auf Änderungen der Netzwerkbedingungen schneller reagiert werden, da der Regelkreis nicht das Netzwerk involviert. Ein weiterer Vorteil von HTTP-basierten Ansätzen ist die Möglichkeit, auf vorhandene Infrastruktur (wie z.B. HTTP-Server oder -Proxies) zurückzugreifen, um damit die Einsatzkosten zu verringern oder die Skalierbarkeit zu verbessern.

# Abstract

Internet video streaming is a hot topic in multimedia systems. A large variety of devices (computers, mobile phones, TVs, etc.) are connected to the Internet via wired or wireless networks and are capable of receiving and decoding HD video content. To enable new services like HD video streaming (e.g., online video rental), the Internet's infrastructure was enhanced. But the Internet is still a best-effort network, which does not implement quality-of-service or admission control, resulting in time-varying bandwidth and packet delay, packet loss and network congestion. Because video streaming accounts for a considerable amount of the Internet's traffic, video streaming needs additionally to be congestion-aware, to avoid a congestion collapse of the Internet. The Transmission Control Protocol (TCP) can adapt to changing network conditions and is currently the de facto standard protocol for congestion-aware and reliable data transmission in the Internet. This fact gave TCP-based video streaming a huge momentum.

Consequently, this thesis investigates TCP-based adaptive video streaming for the Internet. The main goal is to provide a solution for congestion-aware video streaming, while still being able to achieve a reasonable performance in error-prone networks. To complement existing work on congestion-aware adaptive streaming, this thesis makes six contributions. (1) The baseline performance of TCP-based adaptive streaming is identified by means of an evaluation of different adaptive streaming approaches. The results represent a reference for further investigations. (2) An investigation on the influence of TCP's behavior in presence of packet loss on the video streaming performance. (3) To overcome the shortcomings of TCP-based video streaming (single TCP connections fail to deliver a good performance in case of packet loss), a new approach to video streaming based on multiple request-response streams was introduced. The novelty of this system is that it is able to make use of multiple HTTP-based request-response streams while still providing TCP-friendliness. (4) A performance model of the HTTP-based request-response streams was developed, to estimate the influence of the system parameters and the network characteristics on the throughput performance. (5) A comprehensive evaluation of the HTTP-based request-response streams under diverse network conditions was conducted, to validate the model's estimations. Additionally, the TCP-friendliness was evaluated, showing that request-response streaming systems can be configured to achieve TCP-friendliness. (6) A cellular network with high bandwidth fluctuations and RTTs was used to investigate the performance of the request-response streaming system in a mobile video streaming scenario. The results indicate that the streaming system can make good use of the available bandwidth, while the number of quality switches is kept low.

While aggregating multiple TCP connections to improve the TCP streaming performance is quite common, usually the improvement comes at the cost of high deployment effort. By placing the streaming logic at the client, request-response streams can avoid this complexity. Additionally, this client-driven approach responds faster to changing network conditions and enables easy recovery from connection stalls or aborts, because the control loop is at the client. To improve the network efficiency and the scalability in terms of number of clients served, HTTP-based request-response streams can utilize HTTP proxies and caches.

# Contents

CHAPTER

# 1 Introduction

## 1.1 Motivation and Use Case

The Internet and its applications are accelerating technical innovation in consumer electronics. Currently, a growing number and variety of devices are connected to the Internet. To support new applications, the network infrastructure was enhanced, resulting in considerably increased last-mile bandwidths [19]. One of these new applications is Internet video streaming, which heavily relies on network bandwidth. While video conferencing was popular for a long period of time, video-on-demand (e.g., online video rental) and Live Video (TV broadcasts) are new emerging video streaming applications. In Figure 1.1, the transmission path in an Internet video streaming scenario is shown. As already mentioned, Internet video streaming generates a considerable amount of network traffic and its still growing [55, 94]. For that reason, Internet video streaming needs to be congestion-aware, to avoid a congestion collapse of the Internet.

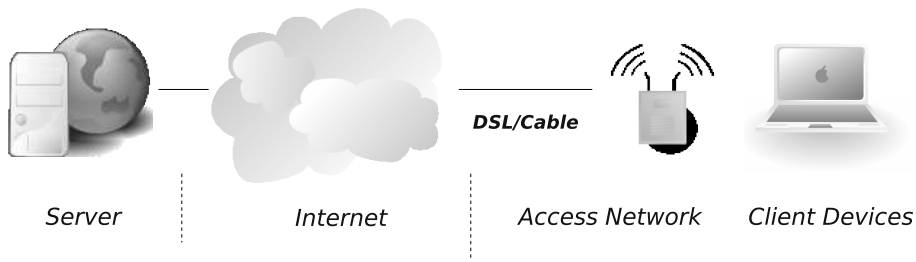



Figure 1.1: Internet video streaming scenario.

The Transmission Control Protocol (TCP) is currently the de facto standard protocol for congestion-aware and reliable data transmission in the Internet. This fact gave TCP video streaming a huge momentum. As a result, video delivery over the Internet based on TCP became very popular in the recent past. TCP was often seen as an improper choice for video streaming [49], because its reliable data transfer and congestion control introduces significant delay. While this is certainly a problem in VoIP, which requires low end-to-end delays, other applications like Video-on-Demand or video in social networks [42] are less demanding. Start-up delays of several seconds are tolerated, if jerky playback can be avoided.

In the beginning of TCP-based video streaming, the download-and-play and progressive-download paradigms were dominant [7]. These methods usually deploy video content with constant bit rate (CBR) and use excessive buffering to overcome possible bandwidth shortages. As a result, very high start-up delays are introduced. While these approaches work reasonably well for short video clips, serving videos of one or more hours play-out time is difficult. But play-out buffers can only compensate packet jitter and/or bandwidth variations. As a result, even large buffers may drain if the play-out duration is very long and a constant network bandwidth/video bit rate mismatch is present. Eventually, the buffer is empty and re-buffering will happen, which leads to a stall in the video play-out (jerky playback).

There are several reasons why such a bandwidth/video bit rate mismatch may occur. One possibility is that the video bit rate may have been selected improperly and exceeds the maximum available bandwidth. While this problem can be solved easily, the networks' characteristics impose more challenging problems. The Internet and the last-mile networks are shared resources, so the available bandwidth changes constantly with users joining or leaving the network. In addition, the amount of traffic generated by the users may change over time. This reflects the best-effort characteristic of the Internet, which features no quality of service (QoS) or admission control. As a result, a bandwidth/video bit rate mismatch may occur at any time, because of the ever changing network conditions, namely the available bandwidth, the packet delay, the packet loss and the congestion level of the network.

Nowadays, a massive shift to *adaptive video streaming* can be observed

[114, 64, 15, 4, 5, 37, 70]. This is because of the increasing requirements of high-definition (HD) video and changed user expectations. Users are simply not willing to wait minutes before the video starts to play. On the other hand, it is not possible to download a whole HD video within seconds. The key to short start-up delays is adaptive video streaming. Because it adapts the video bit rate to the available network bandwidth, bandwidth/video bit rate mismatches are less likely to occur. For that reason, adaptive streaming is able to limit the buffer usage and therefore the needed buffer size. Another aim of adaptive video streaming is the efficient usage of the available bandwidth and to provide the best possible video quality. Changes in the video quality should be kept at a minimum, while offering a good average quality. Because it is not possible to fulfill both, infrequent quality changes and good average quality, a trade-off has to be found between these requirements.

In the following, the state-of-the-art of adaptive video streaming will be reviewed. Section 1.3 will outline the research goals of this thesis, followed by a brief overview of the contributions made to the state of the art in the field.

## 1.2 Related Work

Video streaming spans a broad field of research, involving different scientific disciplines. While the most prominent fields are related to video coding and networking aspects, also other technologies need to be integrated before a streaming system can be deployed.

Additionally, the intended use case has to be considered. The main use cases for adaptive Internet video streaming are *video-on-demand* and *live video streaming*. In video-on-demand, a pre-encoded video (e.g., a movie clip) is streamed from a video server to a client. Because the whole movie is available at the server, the video server can transmit the video slower or faster, to keep the buffer level on the client constant. As a result, the average transmission rate of the video (stream-out rate) can be lower/higher than the average video bit rate. In addition, remote-control-like operations, like fast-forward, rewind, pause, etc., can be easily implemented. In contrast to video-on-demand, live video streaming is more challenging, because the video is not available beforehand. Consequently, the average stream-out rate is usually limited to the average video bit rate.

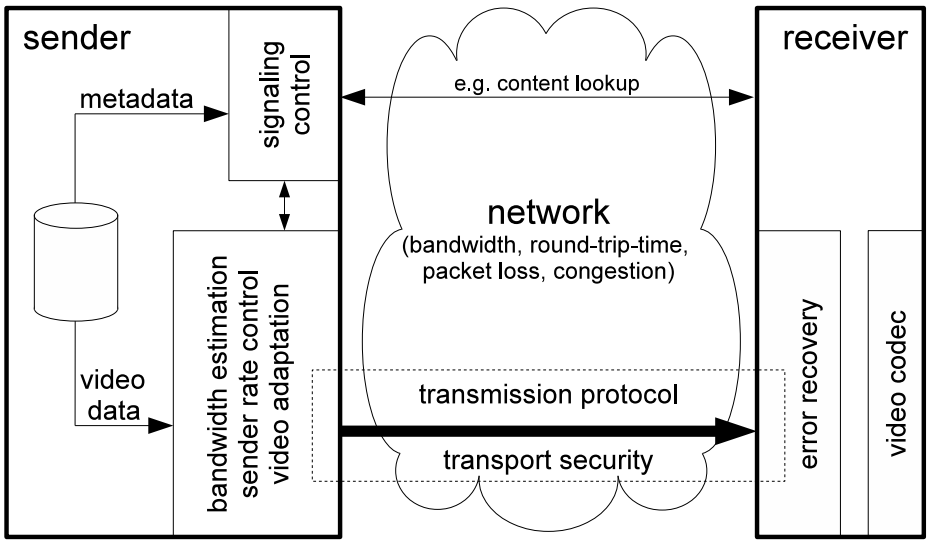



Figure 1.2: Example of an Adaptive Streaming Architecture.

*Adaptive Internet video streaming* can be classified into several research topics [7]. In Figure 1.2, an example of an adaptive streaming architecture is shown. The main topics are described in the following.

**Video Codecs and Adaptation:** Efficient video codecs are an important aspect of video streaming, because they enable the transmission of video content in good quality, even in networks with bandwidth restrictions. Video adaptation is usually deployed to tailor the video content to the user's requirements or current network conditions. Scalable video codecs support adaptation by design, enabling new approaches to adaptive video streaming.

**Transmission of Video Content:** The delivery of the video data to the client is coordinated by *transmission protocols*. Usually, these protocols are either located in the transport- or application-layer and are optimized for the transport of video data. In case of Internet video streaming, also congestion-control and error-resilience have to be provided, to be able to cope with the best-effort nature of the Internet. Because transmission errors are very likely to occur, *error recovery* is needed to handle connection resets or stalls in the transmission, while *error correction/concealment* is used to correct or mitigate transmission errors.

**Adaptive Streaming Systems:** Streaming systems are a challenging research topic, because they aim to optimize the interaction between the single components under various constraints. In addition to video codecs, adaptation and transmission protocols, streaming systems integrate components for *signaling the content-location and content-related metadata*. Adaptive streaming systems also have to cope with bandwidth fluctuations, which call for *bandwidth estimation* or *sender rate control*. Depending on the application, *transport security* may be needed to prevent unauthorized access to the video data. All components needed for building a streaming system are incorporated into a *streaming system architecture*, which additionally defines the interactions between the components.

**Evaluation of Adaptive Streaming Systems:** For streaming systems, the *performance under different network conditions* is usually the benchmark criterion. The performance is measured for different channel bandwidths, round-trip-times (RTTs), packet loss rates and congestion levels and compared in terms of effective throughput and/or video quality (peak signal-to-noise ratio - PSNR). In Internet-like scenarios, also *TCP-friendliness* has to be provided, because TCP is currently the de facto standard for congestion-aware data transmission in the Internet. For real-world deployments, *scalability in terms of number of clients served* and *network efficiency* are of most importance, because they help to keep the deployment costs low.

In the early days of the Internet, video streaming was only possible at low resolutions and bit rates. For that reason, it was necessary to use the available bandwidth in a very efficient manner. The Real-Time Transport Protocol (RTP) [85] - developed for packet switched networks - was used to keep the protocol overhead low and to enable error recovery in case of packet loss. Because RTP needs dedicated server and network infrastructure and has several issues with network address translation (NAT) and firewalls, the usage of RTP for Internet video streaming was rather low. With increasing video traffic in the Internet [55], the deployment of RTP - in most cases on top of the User Datagram Protocol (UDP) - is not justifiable, because UDP lacks congestion control. The Datagram Congestion Control Protocol (DCCP) [48] was created to provide application layer protocols like RTP with congestion control, but - in contrast to TCP - DCCP does not provide reliable data transmission, because it would break the real-time requirements of DCCP [95]. For Internet

applications using low data rates like Voice-over-IP (VoIP), UDP's congestion-unawareness is not seen as a problem. In contrast to this, Internet video streaming has to be congestion-aware, because it accounts for a significant part of the Internet traffic [55, 94]. But even in this case, the usage of DCCP is problematic, because DCCP would need to use forward error correction (FEC) [7] to cope with potentially lost packets. FEC usually recovers lost packets up to a specified packet loss rate, which is dictated by the transmission channel. While channel characteristics are known in private networks (like for IPTV networks [10]), this is not the case for the Internet.

As a result, the usage of DCCP for Internet video streaming is seen very controversial, because nowadays the users are not willing to accept visual artifacts due to errors in the video transmission [10]. Up to now, there was little acceptance of DCCP in research or industry. This is mainly because it is not clear in which cases DCCP will be beneficial compared to UDP or TCP. In contrast to DCCP, TCP is well established and the de facto standard protocol of the Internet. While in classical streaming TCP was often seen as a bad choice [49], TCP's ability to adapt to changing network conditions makes it a good candidate for Internet video streaming.

Consequently, this thesis focuses on *TCP-based adaptive video streaming for the Internet* and investigates its applicability for video-on-demand services. The related research topics are *application-layer transport protocols*, *adaptive streaming systems* and the *evaluation of streaming systems*. In Section 4.5, the most prominent work on TCP-based adaptive streaming will be discussed. Because Internet video streaming is of most importance to the industry, also standards and applications related to TCP-based adaptive video streaming are presented. To keep the review of the related work concise, current developments in video codecs, video adaptation and transmission protocols will be discussed separately in Chapters 2 and 3. Chapter 2 introduces the most prominent video codecs currently used in video streaming systems. In addition, the basic principles of video adaptation will be reviewed to gain insights which adaptation options are available to adaptive video streaming. Chapter 3 will give an overview on currently standardized network protocols with a special focus on the applicability to the envisioned use case (Internet video streaming).

## 1.3 Research Goals and Contributions

Traditional non-adaptive streaming systems are server-centered and therefore allow for a centralized solution. Scalability in terms of clients served and network utilization can be realized by utilizing multicast networks [7, 78]. While multicasting works well in private networks (like IPTV networks), things get complicated in the Internet.

An Internet video streaming system should at least be:

- *Adaptive.* The Internet is a best-effort service, hence, changes in the available bandwidth may occur any time. The system should avoid jerky playback by adapting to the current network conditions.

- *TCP-friendly.* Competition over the available bandwidth should be fair. Currently, TCP-friendliness is the metric to assess the fairness of a network protocol, because TCP is the de facto standard protocol of the Internet.

- *Scalable in terms of clients served.* The overhead to provide adaptiveness and TCP-friendliness should not lower the number of clients served.

- *Network-efficient.* The available bandwidth should be used in an efficient manner and additional redundancies in the video transmission should be avoided. E.g., $n$ clients requesting the same video content should not utilize bandwidth equal to $n$ times the video bit rate, but ideally consume only the bandwidth of a single video stream.

When evaluating existing approaches to adaptive video streaming (see related work in Section 1.2 and 4.5), it became evident that TCP-based adaptive video streaming is most suited to be deployed in the Internet. While streaming systems based on UDP estimate the available bandwidth based on client feedback, adaptive TCP streaming uses TCP's inherent adaptability to steer the stream-out rate. TCP-based adaptive streaming can produce good results even in congested networks, which makes it a good candidate for Internet video streaming.

But TCP also exhibits unfavorable characteristics, which limit streaming performance. Packet loss has a severe effect on TCP's performance, making video streaming almost impossible. Additionally, TCP streaming has several scalability issues. Usually, all streaming and adaptation logic is placed on the server, to avoid adding the network delay to the control loop. This may lead to poor scalability in terms of clients served, because compared to non-adaptive streaming it requires additional effort to manage the client state and adapt the video content. TCP's end-to-end transmission paradigm and the adaptation to the specific client requirements lead to a unique streaming session for each client. For that reason, no caching or proxying of the video data can be deployed in the network, leading to a bad utilization of the network resources.

HTTP-based video streaming can utilize content distribution networks (CDNs) and HTTP caches to enhance the scalability in terms of of clients served and the network efficiency, because - unlike TCP - it uses resource identifiers to locate the video data. In addition, existing infrastructure can be reused, resulting in low deployment and maintenance costs. Apart from the improved scalability, HTTP's client-driven paradigm also enables new approaches to adaptive streaming.

The main research goal of this work is *congestion-aware video streaming in error-prone networks* with a focus on video-on-demand. To complement existing work on this topic, this thesis makes the following contributions:

- *Performance Evaluation of TCP-based Adaptive Video Streaming.* Different adaptation approaches to TCP-based adaptive video streaming are evaluated. As a result, the baseline performance of a TCP-based streaming system is identified, which should represent a reference.

- *Analysis of TCP-based Video Streaming Regarding Packet Loss.* The influence of TCP's behavior in presence of packet loss on the video streaming performance is investigated.

- *A New Approach to Video Streaming based on Multiple Request-Response Streams.* Because single TCP connections fail to deliver a good performance in case of packet loss, a new streaming system based on the request-response paradigm is introduced.

The novelty of this system is that it is able to make use of multiple HTTP-based request-response streams while still providing TCP-friendliness.

- *Performance Model of HTTP-based Request-Response Streams.* A detailed model estimates the influence of the system parameters and the network characteristics on the throughput performance.

- *Comprehensive Evaluation of HTTP-based Request-Response Streams.* The streaming system is evaluated under diverse network conditions to validate the model's estimations. In addition to the throughput performance, also the TCP-friendliness is evaluated, showing that request-response streaming systems can be configured to achieve TCP-friendliness.

- *HTTP-based Request-Response Streams in a Mobile Video Streaming Scenario.* The performance of the request-response streaming system is investigated in a cellular network scenario with high bandwidth fluctuations and RTTs. The streaming system can make good use of the available bandwidth, while keeping the number of quality switches low.

While using multiple TCP streams to enhance the TCP streaming performance is quite common (see related work in Section 4.5), usually the enhancement comes at the cost of high deployment effort. Request-response streams avoid this complexity, by placing the streaming logic at the client. In short, no new TCP implementation is needed and no additional feedback loop between the server and the client. The approaches described in the related work need a rather complex feedback loop between the client and the server, which has to coordinate the transmission in case of connection aborts. This client-driven approach responds faster to changing network conditions and enables easy recovery from connection stalls or aborts, because the control loop is at the client. HTTP-based request-response streams can utilize HTTP proxies or caches to improve the network efficiency and the scalability in terms of number of clients served.

## 1.4 Structure

The remainder of the thesis is organised as follows. The most prominent video codecs used for video streaming, the mechanism of video adaptation, the adaptation constraints and the computational cost of video adaptation will be reviewed in Chapter 2. Chapter 3 will present the most important network protocols used in IP-based video transmission. The main algorithms and architectures behind adaptive Internet video streaming will be shown in Chapter 4. In Chapter 5, the performance of TCP-based adaptive video streaming using H.264/SVC will be evaluated. The behavior of TCP- and HTTP-based request-response streams in error-prone networks will be investigated in Chapter 6. In a comprehensive evaluation, the request-response streaming system will be profiled, comparing the modeled throughput performance to the measured results (see Chapter 7). In Chapter 8, the request-response streams will be evaluated in a mobile video streaming scenario, which shows the impact of a highly time-varying bandwidth on the streaming performance. Chapter 9 concludes this thesis.

CHAPTER

# 2 Video Codecs and Adaptation

Usually, videos are produced at a high resolution and quality, which results - even if compression is used - in very high bit rates. For that reason, videos cannot be used directly in the recorded format, but have to be tailored for specific use cases like video-on-demand or end devices (e.g., smart phones or tablet computers). Video adaptation modifies the characteristics of the video (like spatial resolution, frame rate, visual quality or video bit rate) such that they meet the requirements of the given use case. Also the content of the video can be modified in the adaptation process if the use case requires it (e.g., removing violent scenes). Because video is usually stored in a compressed/encoded format, video adaptation has also to cope with different video codecs.

In this chapter, video adaptation will be reviewed in the context of video streaming. In the following, the main properties of video will be discussed, followed by a review of the currently most prominent video codecs used for video streaming. Different types of requirements which steer the adaptation process will be presented in Section 2.2. The basic methods of video adaptation will be discussed in Section 2.3.

## 2.1 Video Codecs

The amount of information stored within a video is very high. For example, a 1080p HD video at 50 fps using 4:4:4 subsampling represents approximately 2.5 Gbit of data per second. Because of that, it is not practicable to store videos without using compression. Video codecs are used to reduce the amount of data needed for storage by removing redundancies

between consecutive video frames. Further reductions can be achieved by removing information (irrelevancies), which cannot be recognized by the human visual system. The most prominent example is 4:2:0 chroma subsampling, which reduces the resolution of the color information by half in each dimension. Other tools used for encoding images are transformations to the frequency domain, like the discrete cosine transform or the wavelet transform. A video codec bundles different video coding tools to enhance the video compression [79].

Depending on the application domain, different video codecs are used. For example, low-latency applications like video conferencing restrict video codecs by keeping the structural delay of the prediction structures at a minimum. Because video conferencing is a real-time application, also encoding and decoding of the content in real-time is mandatory, which obviously restricts the complexity of the video codec.

On the other hand, the requirements for video-on-demand services are quite different. Because the video content is usually pre-encoded, very efficient video codecs can be used to optimize the rate-distortion ratio. But when streaming video over the Internet, a single quality/bit rate of the video content may not be sufficient. The available bandwidth may change during the video play-out and cause jerky playback. There are several ways to cope with these requirements. One can pre-encode multiple representations of the content with different qualities. This is very straight forward, but creates a lot of overhead in storage and encoding requirements. To overcome these problems, scalable video codecs were proposed, which embed multiple representations of the same video content in a single bit stream. While the design of scalable video codecs is very elegant, the increased complexity in the encoder and decoder and the inferior rate-distortion performance (compared to non-scalable video codecs) are the major drawbacks.

In the following, the video codec H.264 and its scalable extension H.264/SVC will be briefly introduced.

### 2.1.1 MPEG-4 AVC - H.264

ISO/IEC 14496-10 (MPEG-4 AVC) - also called H.264 (ITU-T) - is a joint standard for video coding established by the MPEG and ITU-T consortium [110]. It is a high efficiency video codec supporting a broad range of applications. H.264 works similarly to existing video codecs like MPEG-2 video (ISO/IEC 13818-2) [2], following a block-based hybrid

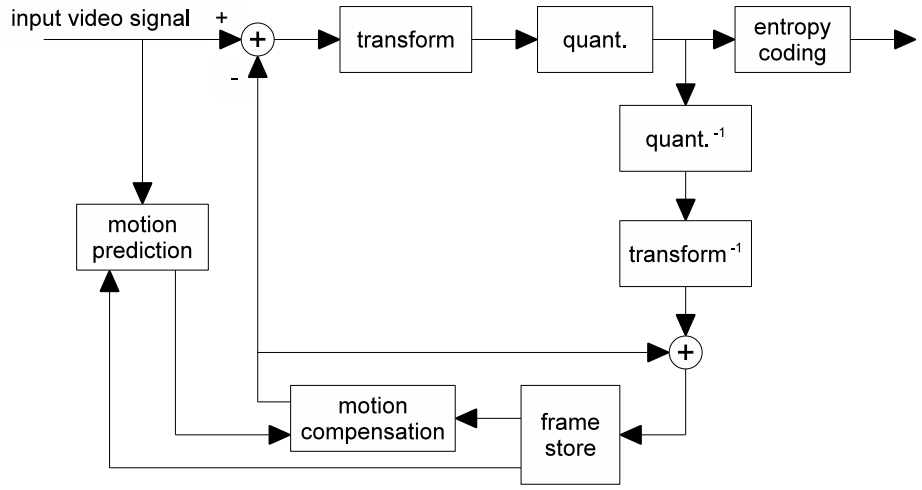



Figure 2.1: Example of a hybrid video coding approach of inter-frame prediction and transform coding.

video coding approach of inter-frame prediction and transform coding (see Figure 2.1). Each video frame is split into blocks (macroblocks), which are the basic elements for further operations. Intra (I) encoded macroblocks are created without using motion prediction. Inter-frame motion prediction is used to remove redundancies between consecutive frames. In H.264, two different inter-frame prediction modes for macroblocks are defined. Predicted (P) macroblocks can use macroblocks of preceding frames as a reference and bi-predicted (B) macroblocks can utilize macroblocks of preceding and future frames for prediction. After the prediction, the residual is computed out of difference of the reference frame (motion compensation) and current frame. The residual is then transformed and the transformed signal is quantized. Typically a transformation to the frequency domain is used, which compacts the information for natural images around the low frequencies (images with structured texture). As a result, the amount of information needed to reproduce perceptual similar results is reduced drastically. To further reduce the bit rate, the transform coefficients are quantized, leading to more or less perceptible artifacts in the resulting image. After quantization, entropy encoding is applied, to remove redundancies within the remaining video data. H.264 provides different tools for motion prediction, transformation and entropy encoding. Profiles and levels for H.264 decoders define the complexity of the decoder by

choosing which video coding tools are to be used and how many macroblocks are to be processed per second. In the following, the most prominent improvements of H.264 compared to former video codecs are presented.

The *multi-frame inter prediction* can use two or more images to predict the content of a frame. In the past, it was usually only possible to predict either from a preceding or a subsequent frame. One application of this tool is the prediction of objects, which were hidden in one frame, but are visible in another frame, basically taking the best fitting parts of each frame. *Intra frame prediction* can be used to significantly reduce the size of I-frames, because it enables the prediction of a macroblock from neighbouring macroblocks. Another new feature is *macroblock partitioning*. Most of the existing video codecs use only macroblocks of a certain size. H.264 can partition its basic 16x16 macroblocks into 8x8 or 4x4 blocks in order to enable fine grained motion prediction. In addition, *quarter-pixel motion prediction* increases the accuracy of the motion prediction. The low complexity *integer transform* avoids the usage of floating point operations and therefore simplifies hardware implementations. An *in-loop deblocking filter* reduces block artifacts and enhances motion prediction. Two new types of *context adaptive entropy encoding* lead to further improvements in compression efficiency.

Another interesting feature is the new *network abstraction layer (NAL)*, which encapsulates each H.264 syntax structure into a logical data packet called *NAL unit*. In general, the NAL units can be grouped into NAL units belonging to the video coding layer (VCL) and NAL units needed to steer the decoding process (non-VCL NAL units). The 1-byte header of the NAL unit consists of the NAL unit type and the NRI (NAL Reference Identification) field, which indicates if the NAL unit is used to reconstruct reference pictures or not. The NAL simplifies the handling of H.264 video data in the transport layer significantly, because there is usually no need to parse the content of the NAL unit again.

H.264 offers high compression efficiency at reasonable computational costs. The simple NAL layer enables easy integration into existing file formats or transport protocols. Also the definition of profiles and levels to customize the codec's properties led to a broad acceptance of H.264. At the time of writing, H.264 is a very successful standard used in many applications and transmission systems.

### 2.1.2 H.264/SVC - Scalable Video Coding

Based on the success of H.264, a new scalable video codec featuring an H.264-compatible base layer was proposed [88]. The main benefit of scalable video coding (SVC) is that it provides scalability at the bit stream level. Properties like frame rate, frame resolution or the quality of the video can be changed without having to re-encode the whole video. Because H.264/SVC also uses the NAL layer, only NAL units have to be removed to change the properties of the video. While former SVC codecs suffered from low coding efficiency, H.264/SVC is the first scalable video codec to deliver a compression performance comparable to its non-scalable counterpart.

The concept of a 'video' can be seen as a sequence of images at a certain resolution, which were captured at a certain point in time with a specific quality. The time between capturing two consecutive images is usually constant. Hence, the number of images per second (also known as the frame rate) is also constant. By dropping frames it is possible to change the frame rate, which is also known as *temporal scalability*. It may be needed if, for example, the client cannot decode as many frames as the initial frame rate specifies.

Another restriction of the client may be the maximum spatial video resolution it can decode. In general, this is defined in profiles and levels. A client may signal that it can only decode videos up to certain level (resolution and frame rate). To adjust the video to the client's requirements, the video may be scaled down to a certain resolution (*spatial scalability*).

The video quality in terms of signal-to-noise-ratio (SNR) has the highest impact on the resulting video bit rate. While with lossless compression the achieved bit rate reduction compared to uncompressed video is very small, lossy compression in H.264 can adjust the resulting video bit rate in that quantization parameters are adjusted. This introduces of course more or less perceptible quality reductions in the video, depending on the video content and the quantization parameters. The ability to change the quality of the video (and video bit rate) without having to re-encode the video is called *quality scalability*.

H.264/SVC implements three scalability options. *Temporal scalability* is enabled by restricting the inter-frame prediction in such a manner that frames can be dropped without

introducing errors. This is also possible in H.264/AVC by means of the hierarchical B-frames [87]. For *spatial scalability*, new prediction modes were introduced. Inter-layer prediction enables the prediction of macroblocks from a lower resolution layer. The prediction for spatial scalability is adaptive. It will choose either inter-frame, intra-frame or inter-layer prediction, depending on which performs best. H.264/SVC provides two modes of *quality scalability. Coarse grained scalability (CGS)* is a special case of spatial scalability. In this case, the resolution does not change and only the quantization parameters differ. As a result, with each CGS layer the quality of the video increases. The second option of quality scalability is *medium grained scalability (MGS).* While in CGS, like in spatial scalability, layers can only be changed on IDR- (instantaneous decoder refresh) or switching frames, MGS is more flexible. In MGS, the quality may change for each frame. In general, changing quality at frame level may introduce errors, because the decoder relies on correctly decoded reference frames. In this context, correct means that the reference frames should be reconstructed to the same reference frames, which the encoder used for motion compensation. If the reference frames differ from the ones used in the encoder, all frames referencing macro blocks from these frames are not decoded correctly either. This chain of events is also called *decoder drift*, which leads to more or less perceptible artifacts in the decoded frames. In MGS, special measures have been taken to avoid continuous decoder drift. *Key pictures* are periodically inserted into the bit stream, for which the prediction is restricted to the base layer [88]. With MGS, H.264/SVC is able to provide both, quality scalability at a fine grained level and good compression efficiency, which is novel in scalable video coding.

Listing 2.1: NALU header with SVC extension

```
+---------------+---------------+---------------+---------------+
|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|0|1|2|3|4|5|6|7|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|F|NRI|  Type   |R|I|   PRID    |N| DID |   QID   | TID |U|D|O| RR|
+---------------+---------------+---------------+---------------+
```

In order to signal the scalability options in H.264/SVC, the NAL unit concept of H.264/AVC (NAL unit header and types) was extended [105]. The NALU header with the SVC extension is shown in Listing 2.1.

The H.264 NALU header (see Section 2.1.1) is extended by three bytes, including parameters like the temporal ID (TID), dependency ID (DID), quality ID (QID), and the priority ID (PRID). The *temporal ID* signals the dependencies within the inter-frame prediction. In general, frames described by NALUs with high TID depend on frames/NALUs with low TID. The parameter can be used to extract certain frame rates out of the H.264/SVC bit stream. Spatial or CGS enhancement layers use the *dependency ID* to signal different layers. Layers with higher DID values always depend on layers with lower DID. The *quality ID* is used to signal MGS quality enhancement layers. The H.264/AVC backward-compatible base layer has always a DID and QID of zero. Because H.264/AVC NALUs have no SVC extension header to signal this information, a prefix NALU was introduced in H.264/SVC, which precedes each AVC NALU. The usage of the *priority ID* is not defined in the H.264/SVC standard and can be allocated based on the needs of certain applications or use cases. The PRID may be used to define a suggested adaptation path, which specifies in which order the NAL units should be discarded in case of adaptation. For example, the Quality Level Assigner tool that is included in the JSVM [40] reference software uses the PRID to create an adaptation path which is optimal w.r.t. rate-distortion.

## 2.2 Adaptation Constraints

In recent years, the way people consume video changed a lot. While usually videos were watched in front of TVs or at the cinema, advances in computer technology enabled the consumption of video on PCs or portable computers. Nowadays, videos can be watched in any living condition, because almost all electronic devices support video playback. But all devices usually differ in display size and computational capabilities, which calls for video adaptation. As a result, the use case (e.g., mobile video consumption) imposes constraints onto the video [13, 71]. The two main classes of constraints are the *usage environment* and the *user preferences*, which will be explained in the next sections.

### 2.2.1 Usage Environment

Every video distribution system builds upon a specific infrastructure, consisting of video servers hosting the video content, networks for the transmission of the video and finally

clients or consumer end-devices for displaying the content. This infrastructure is also called usage environment.

As mentioned before, the video content has to be tailored to consumer end-devices. For example, small mobile phones may not be able to decode HD video and also do not supply a display with the according resolution. Even if the device may be capable of decoding the content in real-time, for smaller display resolutions, the decoded video has to be resized on the client accordingly. This would be a waste of network resources, because more information than necessary would be transported to the client. Therefore, the video should be adapted to the client characteristics. This can be done by either choosing a suitable representation of the video from a set of pre-encoded versions or by adjusting the video to the client characteristics.

While the restrictions of the client devices are somehow static and can be negotiated before the video transmission, the characteristics of a network show a more dynamic behavior. Networks are in general a shared resource used by many applications and users in parallel. This problem can be solved by admission control for wired networks, where the networks are administrated to prevent network overload and thereby congestion. Currently, deploying admission control is only possible in private networks, where a single administrator entity manages the available resources. Best effort networks like the Internet do not feature quality of service or admission control, either because they are too complex to implement or it is not feasible due to the network's architecture. Wireless networks exhibit additional characteristics, inflicted by the wireless transmission channel. Hence, the achievable throughput depends not only on the number of parallel data streams, but also on the quality of the transmission channel and the distance between the sender and the receiver. As a result, the available bandwidth may change over time and certain packets may be lost due to transmission errors. Additionally, congestion may occur in best effort networks when different data flows compete over their network share.

Apart from the constraints of the networks and the user end-devices, other constraints of the usage environment like the brightness or the noise level of the environment may have an influence on the video adaptation [71].

### 2.2.2 User Preferences

While the constraints of the usage environment globally apply to all video transmissions, this class of constraints is directly related to a specific user. Because each user is different in, e.g., age, gender or nationality, also different legal issues or expectations may be taken into account when adapting the video content.

For example, violent scenes may be removed from videos before they are shown to children. But also national laws may restrict the broadcast of videos with certain content. Another example is video adaptation for handicapped people, which have specific restrictions on their perception. E.g., deaf people may benefit from subtitles, which can be embedded into the video stream.

Preferences of the user may also set restrictions on the video adaptation process [71]. One example is that the quality of the video is more important to the user than the time he/she has to wait for buffering the video at the client. Someone else may be annoyed by jerky playback of the video, but the video quality in terms of SNR may not be that much of importance.

Looking at the use case *Internet video streaming*, one can identify several restrictions. The Internet is a best-effort network, so a streaming system has to cope with fluctuations in the available bandwidth and with network congestion. Because bandwidth variations usually result in jerky playback, buffers are used at the client to mitigate this problem. In addition, a transport protocol providing TCP-friendliness should be used, in order to avoid congestion in the Internet.

This short roundup shows that the use case influences not only the adaptation of the video, but also effects the behavior of the streaming system.

## 2.3 Video Adaptation

The overall aim of video adaptation is to maximize the utility of the video. It is a complex process, which has to consider diverse constraints (see Section 2.2). The utility is in general a combined metric, taking the preferences of the user, the service provider and the content provider into account [13]. Video adaptation is a broad term, comprising low-level format

conversions, video property adjustments and structural or semantic changes. This work focuses on the adjustment of the video properties, without changing the actual video content. There are two main approaches for changing video properties, like resolution, frame rate or video bit rate, which are *video transcoding* [101] and *scalable video adaptation* [13, 68]. These techniques will be briefly introduced in the following.

### 2.3.1 Video Transcoding

The most straight forward method to convert a video with certain properties to an output video with different properties is called *full transcoding*. This simply means that first the input video is fully decoded. After this, the resulting uncompressed video is adapted to meet the requirements of the output video. And finally, the adapted video is re-encoded again [101]. The goal of transcoding is to reduce either the *bit rate*, the *spatial resolution*, the *temporal resolution*, or a combination of them. This process is very generic and can be applied to all kinds of video formats. Full transcoding is often used in offline processing of video content, but seldom in adaptive transmission systems, because it comes along with high computational costs and introduces additional transmission delay. To overcome the obvious flaw of full transcoding, a large number of optimizations were proposed to speed up the transcoding process [101]. While a change in all dimensions (temporal, spatial and SNR dimension) usually calls for full transcoding, for reductions in a single dimension, optimizations can be made by utilizing the knowledge about the codec's architecture.

#### Video Bit Rate Reduction

Reducing the bit rate of the video may be required, for example, to adjust the stream-out rate to the available bandwidth. Figure 2.2 depicts a pixel-domain transcoding architecture for bit-rate reduction, which is based on a hybrid video coding approach of inter-frame prediction and transform coding. Due to motion prediction, video encoding is usually much more expensive than video decoding. So one idea is that the effort for motion prediction may be reduced if the motion vectors of the input video can be used in the encoding process of the output video. Because in general the spatial resolution is not changed if only a change in the bit rate is required, it is usually assumed that the motion vectors can be reused [101]. This simplifies the encoding process drastically, because the motion prediction accounts for

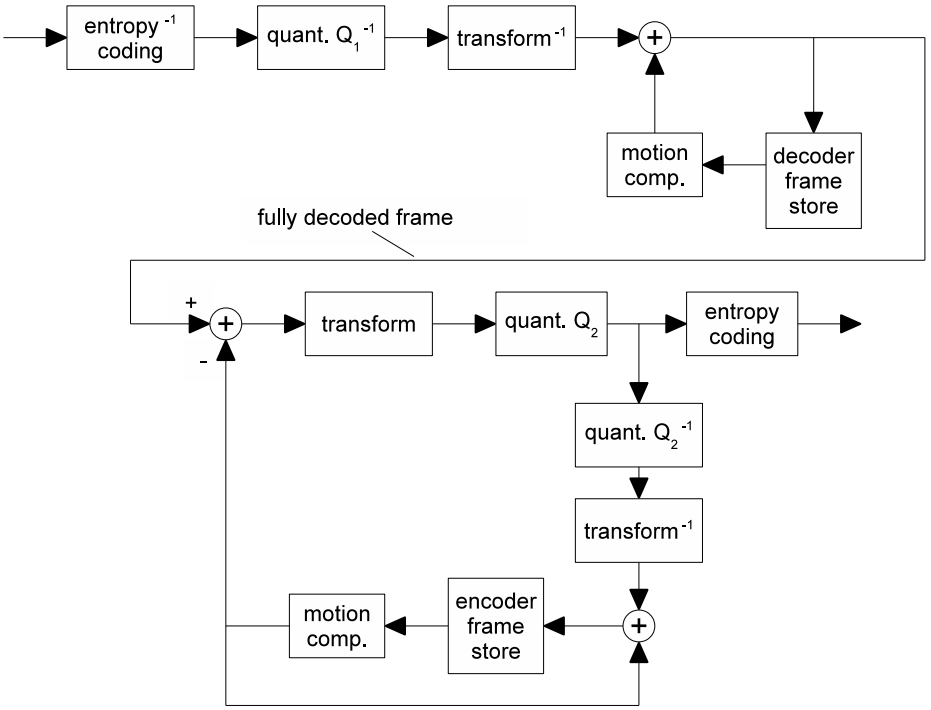



Figure 2.2: Example of a pixel-domain transcoding architecture for bit-rate reduction [101].

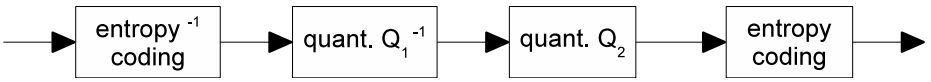



Figure 2.3: Simplified transcoding for bit-rate reduction suffering from decoder drift [101].

a large portion of the encoder's complexity. As a result, in the encoding step, the motion prediction can be omitted (compare Figures 2.1 and 2.2). Only the quantization is changed in the encoding step to reduce the video bit rate.

If perceptual artifacts can be tolerated, even more efficient optimizations can be made. Figure 2.3 illustrates an efficient transcoding architecture for bit rate reduction. Unlike the architecture shown in Figure 2.2, this architecture omits the motion compensation and the transform to the pixel domain. Only the transform coefficients of the residual signal are re-quantized. While this approach speeds up the transcoding process significantly, it does not consider that the motion compensation relies on correctly reconstructed reference frames. Because each decoded frame is a potential reference frame, the errors may propagate, leading to decoder drift (see Section 2.1.2). As a results, more or less perceptible artifacts are

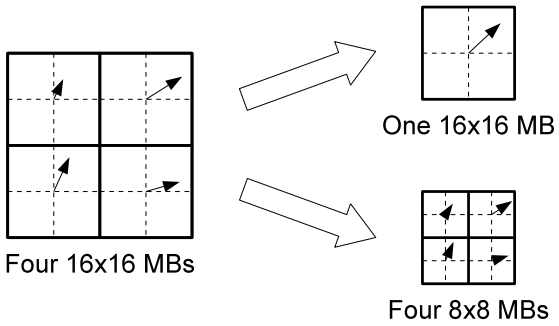



Figure 2.4: Examples of motion vector mapping for four 16x16 MBs [101].

introduced into the decoded frames. The amount of artifacts primarily depends on the video's content and the motion prediction structures used in the encoder.

### Spatial Resolution Reduction

Transcoding architectures for spatial resolution reduction also reuse motion vectors to decrease the transcoder complexity. For example, a technique called *motion vector mapping* combines the motion vectors of adjoined macro blocks to a single motion vector for the new macro block [101]. Another type of mapping utilizes differing macro block sizes. As depicted in Figure 2.4, the motions vectors may be reused directly in macro blocks of smaller size.

The transform to the frequency domain results in low and high frequency components of the image data. Retaining only the low frequency parts of the transform coefficients can be used to change the amount of detail, which can also be interpreted as a reduction of the spatial resolution. The low frequency parts can be packed into smaller macro blocks, resulting in a smaller spatial resolution of the video [101]. This transcoding architecture works similar to the re-quantization approach for bit rate reduction. For that reason, also decoder drift may occur if motion prediction is used in the encoding process.

### Temporal Resolution Reduction

A temporal resolution reduction can be achieved by removing frames from the video. But removing reference frames from the video stream usually leads to errors in the decoding process. In contrast to this, frames not used for motion compensation can be removed

without creating visual artifacts. Because this usually restricts the temporal resolution reduction in too many ways, special methods to speed up the transcoding process were proposed. References frames may be removed if the reference frames can be replaced by other frames. To identify possible candidates, the motion vectors are traced throughout the video stream [101].

A temporal resolution reduction can also be achieved by restricting the prediction structure in the encoding process of the video [87]. Because this topic is more related to the temporal scalability of scalable video, it is discussed in Section 2.1.2.

The transcoding optimizations are usually very codec-specific. While it is possible to reuse motion vectors of different codecs, this may not be possible for transformed coefficients. The reason for that is the different nature of the transforms used in the different codecs (e.g., integer vs. discrete cosine transform). As a result, video transcoding usually comes at high computational costs. Nevertheless, transcoding is a very important technique to handle legacy video data. In recent years, a lot of effort was put into research on transcoding, because of the upcoming transition from MPEG-2 video to H.264/AVC [41, 14, 93].

### 2.3.2 Scalable Video Adaptation

The idea of scalable video is to provide means for video adaptation at the bit stream level. SVC enables changing basic video properties (like frame rate, spatial resolution and video bit rate) without having to transcode the video content. By simply removing parts of the video bit stream, different representations of the video content can be extracted. Figure 2.5 illustrates possible operation points of a scalable video configuration consisting of three temporal (T), three spatial (D) and three quality/bit rate (Q) layers. The scalable video configuration shown enables the extraction of 27 different representations out of the scalable video bit stream, where each representation has a different temporal resolution, spatial resolution and/or quality (bit rate). In the following, a codec-specific adaptation approach based on H.264/SVC will be presented. A more generic way to handle scalable video will be presented later.

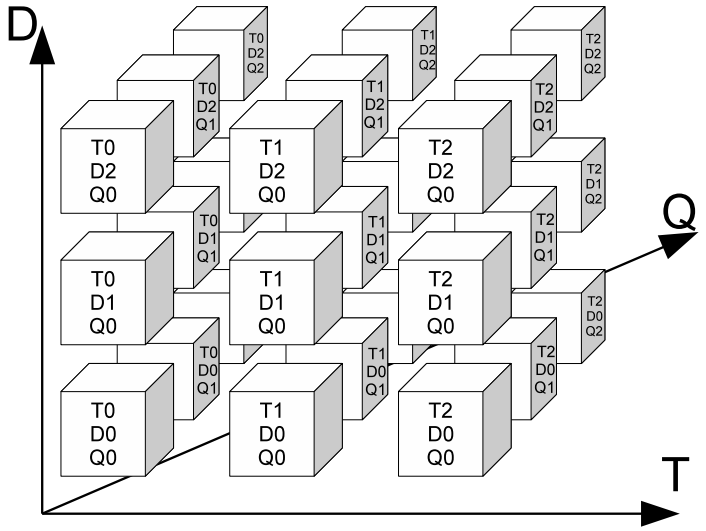



Figure 2.5: Possible operation points of a scalable video with three temporal (T), three spatial (D) and three quality/bit rate (Q) layers.

### H.264/SVC Adaptation

H.264/SVC does not only extend the H.264 video coding layer but also extends the NAL (see Section 2.1.2). The scalability options are signaled in the SVC extension header, which was designed to be easily parsable. As a result, the adaptation process for H.264/SVC is reduced to the parsing of the NALUs' SVC extension header and the decision whether the NALU should be kept in the bit stream or not [105].

For example, a simple adaptation decision compares the temporal ID, dependency ID, quality ID, and the priority ID to a given set of fixed values. Another method may use only the priority ID value, which may in this case be initialized in a rate-distortion optimal manner (see Section 2.1.2). Using this knowledge, an adaptive algorithm can filter out NALUs based on the PRID to achieve a specific video bit rate.

The basic principle of video adaptation is the same for transcoding and scalable video. Both approaches use the chosen frame rate, spatial resolution and/or video bit rate as input for the adaptation process. While SVC offers only a discrete number of adaptation possibilities due to the limited number of enhancement layers, the adaptation process of SVC itself is very simple and computationally inexpensive.

### Format-independent Adaptation based on MPEG-21 DIA

In this subsection, a brief introduction to MPEG-21 based, description driven adaptation will be given. For more information on this topic, the reader is referred to [12, 72, 102, 103].

MPEG-21 Digital Item Adaptation [98] provides different normative description formats, among others, the so called *generic Bitstream Syntax Description (gBSD)*. A gBSD is an XML-based description of a scalable media bitstream. In general, only high-level bitstream structures are described, like packets, headers, or layers. The level of detail of this description depends on the characteristics of the bitstream (syntax elements, scalability, etc.) and the application requirements.

Listing 2.2: generic Bitstream Syntax Description example

```
<gBSDUnit
    xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
    xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
    xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
    xmlns:msi="urn:mpeg:mpeg21:2003:01-DIA-MSI-NS"
    addressUnit="byte" addressMode="Absolute" bs1:bitstreamURI="v.264"
    msi:timeScale="90000" msi:dtsDelta="3000" si:timeScale="90000">
    <gBSDUnit start="2435" length="14590" marker="Frame" si:pts="24000">
        <gBSDUnit length="9" marker="T0D0Q0" />
        <gBSDUnit length="846" marker="T0D0Q0" />
        <gBSDUnit length="669" marker="T0D1Q0" />
        <gBSDUnit length="442" marker="T0D2Q0" />
        <gBSDUnit length="1839" marker="T0D3Q0" />
        <gBSDUnit length="1265" marker="T0D4Q0" />
        <gBSDUnit length="2997" marker="T0D5Q0" />
        <gBSDUnit length="1442" marker="T0D6Q0" />
        <gBSDUnit length="5081" marker="T0D7Q0" />
    </gBSDUnit>
</gBSDUnit>
```

An example of an gBSD for an H.264/SVC frame is shown in Listing 2.2. Each NAL unit of the SVC bitstream is described by a *gBSDUnit*. The NALU's *length* in bytes and the *marker* attribute are mandatory fields to enable adaptation. The marker attribute, for example, can be used to indicate the layer membership of the NAL unit (TID, DID, QID and/or PRID). Usually, the addressing of the NAL units is done consecutively, starting

from the start position of the frame. This gBSD contains all data needed in the adaptation process, which are the parameters of the SVC extension header. But the description is not codec-specific anymore and enables therefore adaptation in a codec agnostic way.

In the MPEG-21 DIA adaptation process, the gBSD of a media bitstream is transformed first, followed by the generation of the adapted bitstream from the original one (guided by the transformed gBSD). The adaptation process mainly comprises simple remove operations of gBSDUnits (bitstream syntax elements) as well as update operations of addresses to keep the adapted bitstream standard compliant. An XSLT style sheet [111] can be used to perform the gBSD transformation, which is usually provided beforehand. Thus, the adaptation process is reduced to transformations of the gBSD XML document, which enables independence from the coding format.

To transmit video data over networks like the Internet, special transmission protocols are required. In the following chapter, the main transmission protocols for video delivery in the Internet will be briefly reviewed.

CHAPTER

# 3 Video Transmission in IP Networks

The introduction of IP networks led to a new paradigm in video transmission systems. IP networks enable the transmission of specifically tailored videos for single users (unicast), which was not possible in broadcasting [7]. As a result, new services like video-on-demand (VoD) or interactive applications emerged. With the help of multicasting [18] also broadcast-like structures can be implemented, which usually introduce less network traffic compared to unicast video transmissions.

On the other hand, IP networks usually do not implement admission control, which can guarantee a specific quality of service [7]. While it is possible to implement admission control in private networks on the application layer (all network traffic is known and administrated), in public networks like the Internet this cannot be put into practice. The lack of admission control leads to new problems like highly varying network bandwidth and network congestion. In the following, several existing transmission protocols are reviewed. A discussion of their application scope and performance in Internet video streaming will show the merit of each protocol.

## 3.1 Internet Protocol Suite

Network protocols define how blocks of data should be transmitted over a network. Unlike in classical broadcasting, packet switched networks enable an end-to-end transmission of data packets by routing them through the network. For that reason network protocols use unique addresses to identify the network participants. Before the transmission, a protocol specific header is added to the data block, resulting in a new network packet.

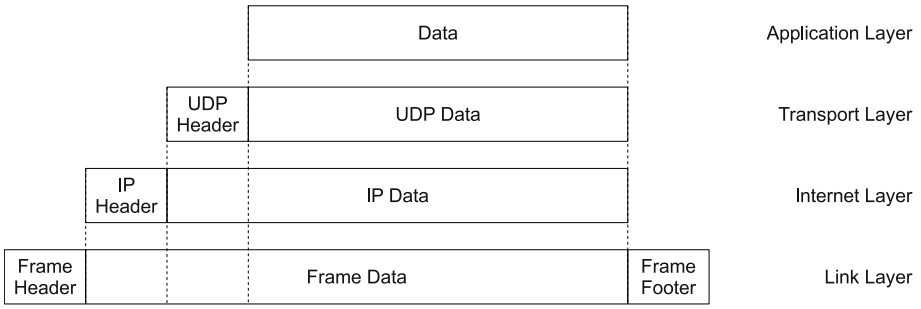



Figure 3.1: Layers defined in the Internet Protocol Suite and the encapsulation of the data (example for the transport protocol UDP).

The *Internet Protocol Suite* defines four protocol layers [11], as shown in Figure 3.1. The *Application Layer* is the top layer of the Internet Protocol Suite. Each application is responsible for the organization and management of the transmitted data. For that reason each application defines a specifically tailored protocol for the transmission of its data. Examples are the well known HTTP protocol [23] or the Simple Mail Transfer Protocol (SMTP) [44]. End-to-end communication is provided by the *Transport Layer*, e.g., by the connection-less User Datagram Protocol (UDP) or the connection-oriented Transmission Control Protocol (TCP). The *Internet Layer* is responsible for the delivery of the data packets from the source host to the destination host. In general, the IP layer does not provide any quality-of-service (QoS), so packet corruption or loss may occur along the network path. If reliable transmission of the data is required, the layers above IP have to implement it (like TCP does). The direct communication with the network is done via the *Link Layer*. Because it represents the interface to the networking hardware it is also called media-access layer protocol. In the following, the main protocols used in video streaming are briefly introduced.

### Internet Protocol (IP)

The communication within the Internet Layer is specified by the Internet Protocol (IP). A *unique IP address* is assigned to each network interface of a host in the network. Currently there are two addressing schemes for IP. IP Version 4 [75] defines 32-bit IP addresses to identify the hosts. Because of the growing number of Internet hosts, the available IP

addresses in the Version 4 schema are running low. The newer IP Version 6 [17] allows for 128-bit IP addresses to resolve this issue. IP is used for routing packets from the source host to the destination host via the Internet, but it cannot distinguish between different data flows. For that reason, raw IP data streams are not suited for video streaming.

### User Datagram Protocol (UDP)

To enable end-to-end communication, transport layer protocols like UDP [74] were put into place. In addition to the destination IP address, a *destination port* is used to address a specific application on the destination host. With the destination IP address and port, the source IP address and the optional *source port*, an end-to-end system can be defined. The idea of UDP is to provide an end-to-end communication mechanism while keeping the protocol overhead at a minimum (UDP is connection-less and state-less). Therefore it inherits all problems of raw IP, which are potential packet loss, out-of-order transmission and the lack of congestion control. The deployment of UDP in the Internet is seen problematic, because excessive UDP traffic may lead to a congestion collapse. But UDP is deployed in private networks, where congestion can be avoided by using admission control [10]. UDP is best suited for low latency applications due to its low protocol overhead, low packet delay and jitter, because UDP simply transmits the packets via the networks as they are sent by the application.

### Transmission Control Protocol (TCP)

While the goal of UDP is to provide a simple protocol for the efficient and fast transmission of data, TCP [76] focuses on reliable, in-order transmission of data. TCP is much more complex than UDP, because it is connection-oriented and features congestion and flow control. The flow control of TCP limits the transmission rate of the sender such that no buffer overflow will happen at the receiver. In the worst case, the transmission is stalled till the receiver empties its buffers. To detect packet loss - introduced by either the transmission channel or network congestion - each endpoint of the TCP connection sends an acknowledgement (ACK) when receiving a TCP data packet. If the sender does not receive the ACK within a given time frame, it is assumed that the packet was lost and the packet is queued for retransmission. TCP's congestion control is based on the additive-increase

multiplicative-decrease (AIMD) algorithm, which assumes that packet loss is triggered by overflowing queues in the routers. While this is in general true for wired networks, in wireless networks packet loss is a common characteristic of the transmission channel. There are several flavors of TCP with different congestion control algorithms [57]. The main issue is the prevention of congestion, while maximizing the network link utilization. The usage of TCP for video streaming is seen controversial [49, 104], because TCP's congestion control introduces a lot of variance in the throughput. Nevertheless, TCP gained a lot of attention in the recent years, because in Internet video streaming easy deployment and the ability to adapt to the available bandwidth are important [104, 50, 114, 4]. In Section 6.1, the behavior of TCP and its performance in the context of video streaming will be reviewed.

### Datagram Congestion Control Protocol (DCCP)

A rather new transport layer protocol is the congestion-aware DCCP [48], which tries to combine the low latency of UDP with the congestion-awareness of TCP. DCCP provides end-to-end connections of congestion-controlled unreliable datagrams by introducing congestion control at the sender. While it is possible to implement congestion control on the application layer with TCP Friendly Rate Control (TFRC) [26, 29], DCCP is a transport layer protocol which is usually integrated into the operating system. Due to this fact, DCCP is able to react faster to changing network conditions, features increased performance and reduced latency. Within DCCP, it is not defined how applications should cope with packet loss. In general, there are two generic methods for handling packet loss [7]. First, one can use retransmissions to transmit the lost packets once again (like in TCP). The second option is Forward Error Correction (FEC) [67, 7, 10], which inserts redundant information into the packet stream to make up for the lost packets. While retransmissions are clearly unwanted in DCCP (because in this case DCCP would act like TCP), FEC needs a maximum packet loss rate as input to determine how much redundancy to insert. Because the Internet is a best effort network, the available bandwidth, the packet delay, the packet jitter and the packet loss are in general unknown [7]. For that reason, the use of FEC and therefore the use of DCCP in the Internet is difficult, because FEC needs to know the packet loss rate. In addition, a new transport layer protocol has to be supported by a broad range of operating systems and the network infrastructure (proxies, NAT, etc.), which is currently not the case

for DCCP.

In the next section, application layer protocols used for video transmission are introduced, which are - in contrast to transport layer protocols - usually media-aware.

## 3.2 Application Layer Protocols for Video Streaming

On top of a transport layer protocol, application layer protocols specify the communication between applications. In the context of streaming digital media, different protocols based on UDP or TCP were defined. Some protocols are responsible for the session initialisation and signaling of the available video content, while the encapsulation and transport of the video data over the network is handled by others. In this section, the main protocols used for video streaming in IP networks and the Internet will be briefly introduced.

### Session Description Protocol (SDP)

The description of the media details, transport addresses, and other metadata of a multimedia session is defined within the Session Description Protocol [34]. SDP does not cover the transport of the media data nor the negotiation of session content. A session may contain multiple media descriptions (e.g., audio and video) and for each media description the used transport protocol and port is given. Additional attributes may further specify which media codecs were used in the encoding process of the media content. To signal the media decoding dependencies, an extension to SDP was defined [83]. which is required when transmitting a single media content over multiple media streams. In case of H.264/SVC video, the base layer and the enhancement layers can be transmitted over multiple streams. In general, the base layer is needed to decode the enhancement layers. This dependency can be signaled in the media decoding dependencies.

### Real Time Streaming Protocol (RTSP)

The negotiation of the session content, the media encodings and the delivery channels are defined in RTSP [86]. While the description of the available media streams can be provided by means of SDP, RTSP can control multiple time-synchronized media streams such as audio and video. For each media session, RTSP manages a state (created via the SETUP method),

which includes information on the media content to stream, the media encoding used and the transport protocol chosen for the delivery of the media content. Once the session is started via the PLAY method, the media is streamed to the client. The PAUSE method causes the streaming of the media to be interrupted temporarily, while the TEARDOWN method is used for the termination of the multimedia session. RTSP can also be seen as a remote control interface to the streaming system.

### Real-Time Transport Protocol (RTP)

The transport of real-time media data (like audio and video) is specified in the Real-Time Transport Protocol (RTP) [85]. The media data is packetized into RTP packets before it is transmitted over multicast or unicast networks. The RTP packet header includes a *payload type* which is used to identify the payload content and a *sequence number* to detect out-of-order delivery of packets. The *timestamp* in the RTP packet header provides timing information of video data contained in the packet and is also used for the synchronization of multiple media streams, like audio and video. The semantics of the *marker bit* is defined by the used RTP profile. An RTP profile for the transmission of audio and video content is specified in the RTP/AVP profile [84]. In this profile, the marker bit denotes the end of a media frame, signaling that the decoding of the frame can be started on the client. The *synchronization source* (SSRC) field of the RTP packet header identifies the media stream the RTP packet belongs to. For that reason, each RTP stream should have a unique SSRC.

In general, RTP is tightly coupled with SDP or other protocols describing the media session, because the definition of the used RTP profile (and therefore payload types), the media codec and the RTP clock rate is not signaled in the RTP header. The RTP profile is signaled in SDP (e.g., the RTP/AVP profile [84]) and can be used to resolve the payload type. In general, two different payload types are defined, static and dynamic payload types. A static payload type can be used to directly identify the media codec, RTP clock rate and other parameters, while the dynamic payload types have to be described in more detail in the SDP. SDP's *rtpmap* attribute is responsible for mapping a dynamic RTP payload type to the used media codec, the RTP clock rate and additional parameters needed for decoding the media stream. RTP payload formats for specific video codecs (apart from the ones defined in the RTP profile) have to be specified in separate documents (e.g., the RTP

payload format for H.264 video [106]). The idea behind using specific payload formats is that the packeting of the media data can be steered in such a manner that for example the latency of the transmission or the influence of packet loss can be minimized.

The RTP specification also defines the RTP Control Protocol (RTCP) [85]. RTCP provides information about the media transport and simple control and identification functionality by periodically transmitting control packets from the server to the client and vice versa. The RTCP *sender reports* (SR) are used to map the timestamps of the RTP packets to the Network Time Protocol (NTP) time [62]. SRs also allow for inter-media synchronization, by normalizing RTP timestamps of the different media streams to the NTP time. The RTCP *receiver reports* (RR) feed information on the quality of the data transmission of a specific SSRC back to the sender. The RRs include the number of packets lost during transmission, the highest sequence number received, the inter-arrival jitter of the RTP packets and the last SR received from the SSRC. This information can be used to detect problems in the media transport, like network congestion.

To modify media streams in the network, RTP also defines intermediate network nodes (called RTP Translator and RTP Mixer), which operate at RTP level. An *RTP Translator* is able to do simple modification of parts of a media stream without having to change the synchronization source identifier (SSRC). For that reason, an RTP translator is in general transparent to the RTP client. Such an RTP translator can for example be used to adjust the media bit rate to the available bandwidth or to change the encoding of the media. The concept of an *RTP Mixer* is more powerful than the RTP translator. An RTP Mixer generates a new RTP stream out of a single or multiple media streams. Therefore, it has to generate a new SSRC and also provides new RTCP sender reports for the new SSRC. In multi-party voice over IP (VoIP) for example, an RTP mixer can be used to mix each voice stream into a single response voice stream, which will be sent back to all participants. Another usage scenario is an RTP mixer acting as a multicast to unicast gateway for layered video. Each video layer is transported in a separate RTP stream. If the client cannot handle multicast streams, such a gateway can be used to generate a single unicast stream out of the layered streams [108].

RTP is very popular for streaming audio and video in private networks, because it features low latency. But RTP was not designed to be congestion-aware. Nevertheless,

several methods were proposed to handle network congestion when streaming with RTP. To enable congestion-aware RTP-based video streaming for the Internet, RTP packets can be embedded into a TCP-based RTSP connection [86]. Another way to handle network congestion is the use of RTP/UDP in combination with TCP friendly rate control (TFRC) [26], which adapts the stream-out bit rate in case of congestion to the fair share. In [29] the usage of RTP/UDP with TFRC was proposed, where the congestion control is implemented on the application layer. In contrast to this, DCCP implements the congestion control on the network transport layer. Therefore DCCP features better performance and less overhead. The usage of DCCP with RTP is defined in [73].

If robustness to packet loss is mandatory, it usually has to be provided in the application layer by means of forward error correction (FEC). While FEC in general assumes that packet loss is a channel characteristic, packet loss is also introduced by network congestion. More details on this topic can be found in Section 3.1 in the description of the DCCP protocol, where the usage of FEC in best-effort networks (like the Internet) was discussed.

### HTTP-based Video Streaming

The Hypertext Transfer Protocol (HTTP) [23] is an application layer protocol working on top of TCP. Basically one can request a resource from an HTTP server, which creates a response accordingly. HTTP gained a lot of attention in the field of Internet video streaming, because of the existing infrastructure such as HTTP servers, proxies and network support.

There are several ways to transmit video data via HTTP. One can fully download and play the video file (also know as the *download-and-play* paradigm) [59], but this has a major disadvantage when playing large video files. The start-up delay increases accordingly with the video file size. *Progressive download* tries to overcome this problem by downloading only the parts of the video file needed in the near future [114, 59]. As a result, the client can decode and display the first part of the video, while downloading the remaining part. This significantly reduces the start-up delay. Seeking within the video can be implemented via HTTP byte range requests. Progressive download is a very popular form of video distribution in the Internet (e.g., YouTube [30]). Because incremental consumption has to be provided by the video file format, not all video file formats are suited for progressive download.

Usually, the video data is obtained from a single file. A new type of HTTP streaming operates on video fragments instead of a single video representation. The idea is to use multiple small progressive downloads to get rid of the stateful transmission scheme, which is usually used in streaming systems. Therefore, no special streaming servers or infrastructure is needed, only an HTTP compliant infrastructure. HTTP streaming also enables another key technology for Internet video streaming: *adaptive streaming* can be used to adjust the streaming bit rate to changed network conditions. Thus, it directly improves the viewing experience of the user, by reducing the start-up delay and jerky playback.

Because this thesis focuses on adaptive streaming with video fragments, the most prominent related work, commercial products and standards using HTTP streaming for video delivery are reviewed in the related work in Section 4.5.

While adaptive video streaming is based on video adaptation and transmission protocols, it additionally requires mechanisms to cope with changing network conditions. In the next chapter, methods for adaptive video streaming in networks with time-varying bandwidth and adaptive streaming architectures will be presented.

CHAPTER

# 4 Adaptive Video Streaming in Networks with Time-Varying Bandwidth

The lack of admission control and quality of service (QoS) mechanisms in the Internet imposes new challenges to video streaming. The Internet offers only a best-effort service, which results in quite drastic throughput fluctuations. Although the Internet does not provide admission control, some mechanisms were invented to avoid network congestion. Congestion-aware transport protocols avoid congestion by continuously adjusting the transmission rate at the sender based on feedback from the client. The review of protocols for video transmission in the previous chapter revealed that TCP, RTP/DCCP and RTP/UDP-TFRC are the main protocols suited for Internet video transmission, because they provide congestion control.

In classical video streaming, TCP is often seen as unsuited for real-time applications. TCP is no real-time protocol and hence cannot give any guarantees on the timeliness of the data delivery. The main reason for this is TCP's congestion control algorithm and its reliability, which may lead to transmission stalls or reduced throughput. Therefore, the use of TCP for multimedia applications is considered controversial in the literature [49, 25]. But there are also good arguments for TCP. TCP's reliable end-to-end transport mechanism makes additional provisions like forward error correction unnecessary, which are mandatory for RTP/DCCP and RTP/UDP-TFRC. Additionally, TCP is connection-oriented and therefore allows for easy traversal of firewalls and NAT devices that may exist along the network path [7]. Because streaming applications can rely on TCP's inherent capabilities,

they turn out to be much less complex than approaches based on the connectionless and unreliable protocol UDP. Finally and maybe most importantly, TCP also offers congestion control. TCP's congestion control assures that only the fair share of the network's bandwidth is consumed by a single TCP connection, which is considered crucial for the stability of the Internet [24].

This thesis focuses on congestion-aware video streaming based on TCP, because it is well supported and can deliver good results for the envisioned use case (see Section 1.1). Consequently, only video streaming based on TCP is discussed in detail in the remainder of the thesis.

Because the achievable throughput in best-effort networks like the Internet can vary significantly over time, disruptions of the video playback or jerky playback may occur if the throughput falls below the required bit rate. *Adaptive video streaming* [7] tackles this problem by adjusting the stream-out rate to the available network bandwidth. The three main approaches to adaptive video streaming are [7, 100, 22]:

- *Rate-Control for Adaptive Video Streaming.* Based on an estimate of the available bandwidth, the stream-out bit rate is continuously adjusted in a fine-grained manner (e.g., per video frame). The goal is to prevent the buffers at the receiver to drain and therefore to avoid jerky playback.

- *Stream Switching.* The video stream can be switched between a certain number of representations with different bit rates to match the available bandwidth. In this sense, this technique works similar to rate-control, but the time between switching points is far greater than in rate-control (multiples of seconds) and the number of different representations is usually small.

- *Priority Streaming.* Unlike the other approaches, priority streaming is not based on bandwidth estimation. Priority streaming utilizes scalable video to provide a graceful degradation of the video content in case of bandwidth shortages. A scalable video bit stream is reordered and transmitted in priority order. Truncating a reordered bit stream only reduces the quality of the video, resulting in a very simple adaptive

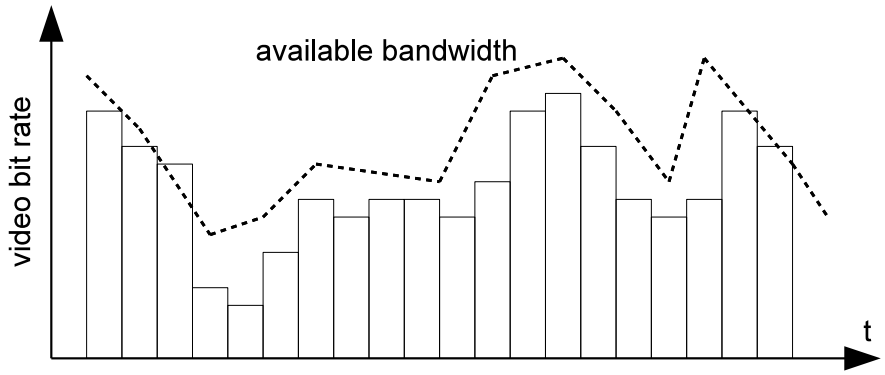



Figure 4.1: Behavior of a rate-control algorithm in presence of bandwidth variations.

transmission scheme.

In the following, the principles and methods of the three existing approaches are introduced. A special section on adaptive streaming architectures will review the most commonly used architectures for adaptive Internet video streaming. After introducing the possible options for adaptive Internet video streaming, the most prominent work, commercial products and standards related to HTTP/TCP streaming are reviewed.

## 4.1 Rate-Control for Adaptive Video Streaming

The goal of adaptive video streaming is to enable continuous video playback and to provide a good video quality. Rate-control [7] continuously adapts the bit rate of the streamed video (e.g., per frame) in such a manner that it matches the available network bandwidth. If the rate can be aligned closely to the available network bandwidth, the buffer drainage at the receiver should be limited to a minimum. Figure 4.1 shows the behavior of a rate-control algorithm. The video bit rate is continuously adjusted to the available bandwidth in a fine-grained manner.

In general, the available bandwidth or the possible throughput in best-effort networks are not known. For that reason, the network's characteristic has to be estimated. In RTP for example, receiver reports give feedback on certain network characteristics, like the packet loss and inter-arrival jitter [85]. RTP/RTCP with UDP in combination with TCP-friendly rate control (TFRC) [26, 29, 115] can be used to determine a stream-out rate which would

not be unfair to concurrent TCP connections. Also TCP can be used to estimate the available bandwidth. The throughput history [77] or information obtained from the TCP stack [9] can be used to estimate the future TCP throughput [51].

The main input parameter of a rate-control algorithm is the estimated available bandwidth. Additionally, the receiver buffer level has to be taken into account, because the buffer is used to compensate transmission problems. The buffer level should be kept above a certain value. A draining buffer can be filled again if the video is streamed out faster than real-time, which is usually only possible with pre-encoded video. To fill the buffer, the rate-control chooses a video bit rate which is smaller than the available bandwidth, but streams the video at the rate of the available bandwidth [51]. On the other hand, if the buffer level is getting too high, the buffer can be drained by using a video bit rate which is larger than the available bandwidth. The part of rate-control maintaining the buffer level is also called *buffer control*.

This example of a rate-control algorithm is very simple, but illustrates the main issues all rate-control algorithms are coping with. The result of the rate-control is a video bit rate and a proposed stream-out rate. The video bit rate is used in the adaptation process, whereas the stream-out rate is used for streaming the adapted video. The video bit rate can be either changed directly in the encoding process or the encoded video can be adapted to a different video bit rate (see Section 2.3). If scalable video is used, the video bit rate can be changed by dropping parts of the video bit stream [7]. After the adaptation, the video is streamed out to the client.

Live transmission of video content is a special case of adaptive video streaming, because it allows only small buffer sizes. One of the characteristics of live video is that the stream-out rate cannot be faster than real-time. For that reason the receiver buffer can only be filled at the beginning. A prominent example of live video streaming is video conferencing, for which it is recommend to use end-to-end delays smaller than 300 ms and audio-video synchronization delays not larger than 200 ms [39, 38].

While rate-control-based video streaming systems can react fast to changing network conditions, the main problem of this approach is that it heavily relies on the accuracy of the estimated available bandwidth. Because a continuous mismatch would drain the receiver buffer, information about the buffer level has to be fed back to the rate-control algorithm

[51], thus enabling the algorithm to correct this problem.

Because of the high complexity (bandwidth estimation, receiver buffer feedback, video adaptation) and scalability issues (on the server one rate-control instance per stream is needed), rate-control-based video streaming systems are not widely deployed. The only exception is video conferencing, which uses sophisticated rate-control algorithms to keep the buffer requirements low. In the next section, an approach similar to rate-control is presented, which trades flexibility for simplicity.

## 4.2 Stream Switching

Currently, a popular way of watching videos in the Internet is *representation selection*. For each video several representations with different resolution or quality properties are presented. The user chooses a representation before the streaming begins, based on his/her personal knowledge or preferences. Usually, the video representations have a constant bit rate for the whole play-out duration. The main problem is that the user has to decide if the available bandwidth will be sufficient for a continuous play-back of the selected video representation. Even more problematic are changing network conditions, which are very common in best-effort networks. In case of stalls or jerky playback, always user interaction is needed. The user decides how to cope with the situation and can either wait for the re-buffering of the video content or the user can switch to another quality level, which has to be buffered once again. To avoid these problems, users can choose the lowest quality or wait till the whole video is buffered.

While complete buffering may be possible for short video clips, this is not feasible for high definition movies lasting on the order of hours. *Stream switching* makes things easier for the user, by simply choosing the appropriate video quality based on the current network conditions. Like rate-control, it estimates the available bandwidth and monitors the receiver buffer level, but reduces the complexity by allowing to switch only between a set of pre-encoded representations of the video content [7, 92]. Because the switching is usually only possible when certain conditions are met (e.g., end of GOP), the time between the switches is usually high (several seconds). In stream switching, a part of the video which is transmitted as a whole is usually called *video fragment*. Figure 4.2 illustrates the behavior of stream switching. While rate-control can continuously adjust the video bit rate

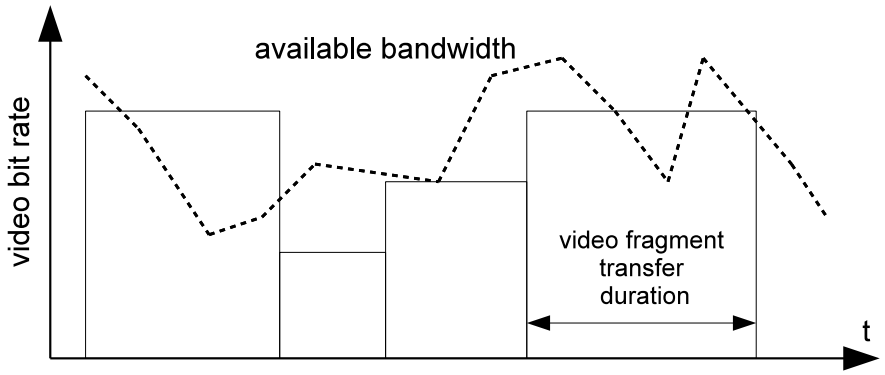



Figure 4.2: Stream-switching in presence of bandwidth variations.

to the available bandwidth (see Figure 4.1), stream switching chooses from a discrete set of pre-encoded video representations. In addition, the time between changes in the video bit rate is much larger. It can also be noticed that the time to transmit a video fragment is usually not constant, because it depends on the video bit rate and the available bandwidth. If they do not match, the transmission of a video fragment may be delayed or may be too fast. Like in rate-control, this problem is handled by buffer control (see previous section), which manages the client's buffer level by adjusting the video bit rate.

To enable stream switching, the video content has to be encoded in different qualities. While this additional effort can be neglected for VoD (the content can be encoded offline), it is rather challenging for streaming live video content. In this case, multiple encoders have to be deployed to encode the video content in parallel. The encoding of the video has to be steered in such a manner that it allows for switching between the qualities at specific points in the video. H.264 defines switching and IDR frames for that purpose [92].

In addition to the computational overhead, also the storage demand is increased because of multiple video representations. Hence, stream-switching solutions offer usually only a handful of different video qualities for economical reasons. In Section 2.1.2, the concept of scalable video was presented, which allows to have a significantly higher number of different bit rate representations of the same content. Another advantage over non-scalable video is that the video can be stored in a single bit stream at the server and only a single encoder is required in the case of live content.

Stream switching is one of first deployed techniques for adaptive streaming. Current

standards recommend to use well established video codecs and network protocols, like the non-scalable video codec H.264 and HTTP [4, 5, 37, 70], to maximize interoperability and ease the deployment in existing network infrastructures. In the related work section (see Section 4.5), an overview on the current standards and applications utilizing stream switching is given.

Stream switching is a widely deployed adaptation technique for VoD and video broadcasting in the Internet, because of its easy deployment and good scalability (in terms of number of clients served). Usually, several seconds of content are buffered in advance to mitigate possible variations in the available bandwidth. Like rate-control-based video streaming, stream switching heavily relies on the bandwidth estimation and the information on the client buffer level. Because stream switching uses a rather low number of different qualities, it cannot adjust smoothly to the available bandwidth. As a result, the client buffer level fluctuates a lot (details on that will be shown in Chapter 5). In the following, a different approach to adaptive video streaming is presented, which does not rely on bandwidth estimation or the information on the client buffer level and is therefore in general more robust.

## 4.3 Priority Streaming

Classical approaches to adaptive video streaming rely on the estimation of the available bandwidth. While measuring the channel bandwidth of an empty network link is quite easy [20], things get complicated with concurrent infinite-source TCP streams. This is because TCP tries to fully utilize the network link by constantly increasing its congestion window. Once TCP reaches the limit of the link, network packets get lost and TCP starts to throttle the stream-out rate by adjusting the congestion window. As a result, in case of congestion, only the fair share of the network bandwidth can be considered as available bandwidth. In addition to the rather difficult estimation of the available bandwidth, changing network conditions may lead to delayed video data packets. Classical approaches cope with these problems by using buffering at the receiver.

*Priority streaming* [22, 50] follows a different paradigm to overcome temporary bandwidth shortages. Instead of the traditional buffering at the receiver, the video quality

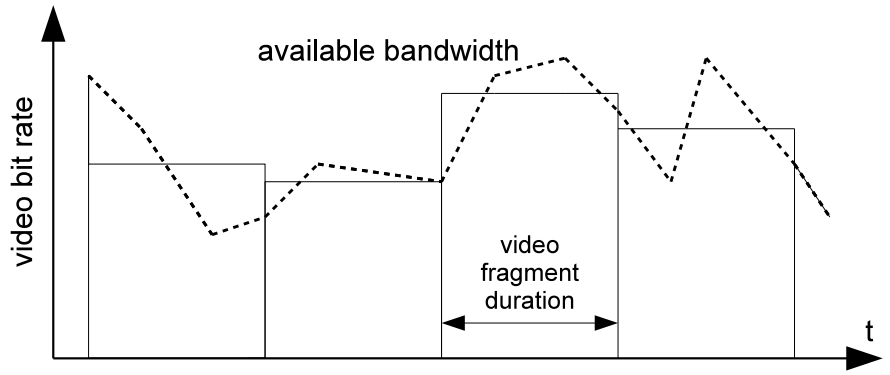



Figure 4.3: Behavior of priority streaming in presence of bandwidth variations.

decreases with the available bandwidth gracefully. The idea is to download the video in such a manner that the quality of the video increases with the amount of data downloaded. Hence, in priority streaming, a truncated video is not truncated in time (this is what usually happens), but truncated in terms of video quality.

With this approach it is possible to efficiently utilize the available bandwidth without having to deploy large buffers at the client. To enable this behavior, the video is split into *video fragments*. In priority streaming, the download duration of a video fragment is limited to the play-out duration of the fragment, to guarantee continuous playback. As a result, the video bit rate of a retrieved video fragment is simply identical to the average transmission rate of the video fragment (see Figure 4.3).

The size (play-out duration) of the video fragments is usually kept constant, because in this case a video fragment can be played out immediately after it is received. If the size of the fragments differs, additional buffering may be needed. This becomes evident when looking at an example with fragment sizes of 2 and 10 seconds. For this example, it is assumed that after retrieving a small fragment, a consecutive large fragment is fetched. While the large fragment is downloaded (takes up to 10 seconds), the small fragment (2 seconds) will be played out. This mismatch usually leads to jerky playback, because the play-out will be likely over before the download is finished. In this case, at least 10 seconds of video (instead of 2) should be buffered.

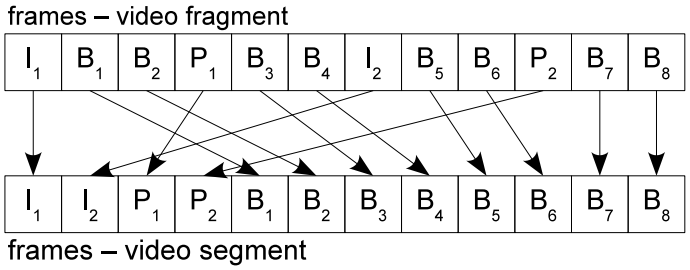



Figure 4.4: Priority re-ordering of a video fragment (resulting in a video segment).

This also exposes the obvious drawback of priority streaming. The decoding (and playback) cannot be started immediately after receiving the first video frame. In general, a video fragment is available for play-out once the download is finished or the maximum download duration is exceeded. This delayed play-out can be interpreted as buffering. Hence, priority streaming buffers at least one video fragment (in case of diverse fragment durations this would be the maximum duration). This limits the use cases in which priority streaming can be deployed to non-real-time applications.

A prerequisite for priority streaming is the usage of a video codec with supports scalability at the bit stream level, because it assumes that the bit stream of a video can be downloaded in such a manner that the video quality increases with the download progress. In general, video codecs do not support graceful degradation of the video quality. Bit streams of scalable video codecs can be rearranged based on the priority of their video syntax elements (e.g., slices, frames, layers). A reordered video fragment is called a *video segment*. If only temporal scalability were to be used, the I-frames would precede the P-frames and the B-frames (see Figure 4.4). Because the I-frames are usually of most importance, they are downloaded first. If no more data could be downloaded in the given time frame, the resulting video would have a reduced frame rate. Adaptive streaming usually tries to minimize the artifacts shown by the adaptation, so quality scalability is commonly deployed to avoid major changes in the perception.

Because of the reordering, it is not necessary to retrieve the whole video segment. It can be truncated at virtually any point. The amount of video data retrieved relates directly to the video quality of the reconstructed video. In general, the changes of the video quality should be kept at a minimum, while offering a good average quality of the video. While this

favors larger video fragments, also the start-up delay increases with the size of the video fragments. The size of the video fragments is usually limited by the maximum start-up delay. Unlike in stream switching, large fragments do not lead to bad network utilization or large buffers, because priority streaming does not rely on single adaptation decisions. It continuously adapts the quality during the download. More details and experimental evidence will be shown in Chapter 5.

Priority streaming is also known as priority-progress streaming (PPS) [50] or deadline-driven streaming [51]. In the literature, different adaptive streaming architectures were proposed. In the following, the main architectures are presented.

## 4.4 Adaptive Streaming Architectures

The standard way of video delivery is streaming the video from a video source (in most cases a server) to a video sink (client) via a network. In a lot of deployments, the server and the client form an end-to-end system, but also intermediate nodes may be present, which cache (e.g., proxy) or modify (adaptation node) the video content. Based on these communicating partners, it is possible to classify video streaming architectures in many different ways.

In adaptive video streaming, the most important question is the location of streaming and adaptation logic. Typically the logic is kept in a single place to keep the complexity of the system low. Therefore, three types of adaptive streaming systems can be differentiated:

- *Server-side Systems.* The streaming and adaptation logic is placed on the server. The client has minimal logic, only receives the video data and gives feedback on the received quality.

- *Client-driven Systems.* The client requests the appropriate video representation from the server, based on an adaptation decision made on the client. The server degenerates to a simple streaming service.

- *Systems using In-Network Adaptation.* An intermediate node between the server and the client intercepts the video stream and adapts the video to the current network conditions.

In the following, the commonly used architectures for adaptive video streaming are briefly reviewed.

### 4.4.1 Server-side Adaptive Streaming

Classical streaming approaches broadcast the video from centralized servers to the clients. While being built for large deployments, broadcasting is not applicable in best-effort networks because of the different bandwidth restrictions of the clients. In unicast streaming, the server sends an individual video stream to each client. In this architecture, the streaming and adaptation logic is on the server, resulting in a simple client implementation. The client requests content from the server once and the server continuously sends the video to the client (push paradigm). Control commands of the user (play, pause, etc.) can be sent to the server, to allow the user to interactively change the video play-out. The most prominent example for unicast video streaming is RTP/RTCP in combination with the control protocol RTSP. RTP's main advantage is the low latency introduced in video transmission. For that reason, RTP can be used for almost any application, ranging from video conferencing to video-on-demand.

Server-side adaptive streaming extends unicast streaming by taking also the changing network conditions into account. By adapting the bit rate of the streamed video to the available bandwidth, the client buffer level can be stabilized. While the adaptation of the video is identical in all architectures, the estimation of the available bandwidth and of the client buffer level is different. In the following, two approaches using server-side adaptation are presented.

#### Server-side Adaptation based on Client Feedback

In this approach, the clients feed information on packet loss, packet jitter and the receiver buffer level back to the server. The server uses this information to model the network link between the server and the client. In addition, the server monitors the round-trip-time of the network connection and estimates the available bandwidth. Like depicted in Figure 4.5, this architecture adds the network to the control loop, which leads to high response times to changing network conditions.

In best-effort networks like the Internet, it is important to deploy a congestion-aware

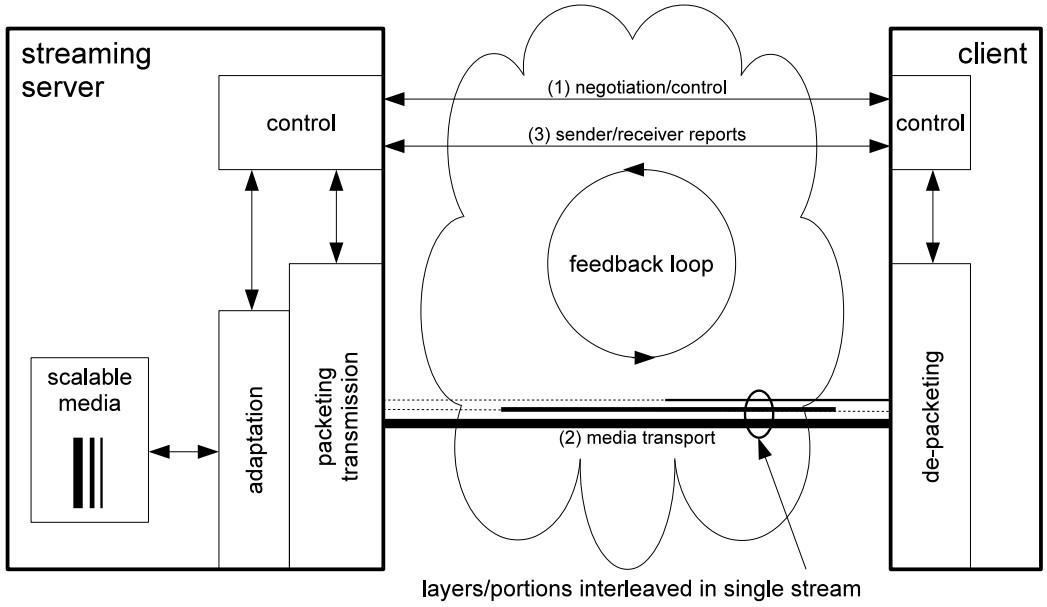



Figure 4.5: RTP-based architecture of a server-side adaptation system based on client feedback.

rate-control algorithm to avoid a congestion collapse of the network. A well known example is RTP/RTCP with UDP in combination with TCP-friendly rate control (TFRC) [115]. The RTCP messages give feedback on the network characteristics, which are used in the calculation of the TFRC transmit rate. This rate is an estimate of the available bandwidth and is used to steer the streaming process.

The main problem of server-side adaptive streaming is the information on the client status. In general, information has to be fed back to update the server about the client state. This information may be delayed or lost on the way to the server, because it has to be sent over a potentially error-prone network. For that reason, the behavior of the adaptation control loop is directly influenced by the network conditions, leading in general to increased adaptation delays.

### Server-side Adaptation based on Estimation of Client Status

The previous approach usually implements congestion control on the application layer (e.g., on top of RTP/UDP). An inherent congestion-aware transport protocol (like DCCP or TCP) can react much faster to changing network conditions. In addition, it allows to determine

the possible throughput by observing the data transmitted [77, 51].

In case of TCP, which also features reliable data transmission, it is possible to estimate the client buffer level on the server [51]. The buffer estimation is based on the end-to-end paradigm of TCP. It assumes that a TCP connection works like a FIFO buffer, which introduces only a comparatively small delay between the buffer level maintained at the server and the client buffer level. By exploiting this knowledge, a very fast control loop can be constructed, which keeps the changes to the client buffer level to a minimum. More details on this approach will be presented in Section 5.

### 4.4.2 Client-driven Adaptive Streaming

By placing the adaptation logic at the client, client-driven adaptive streaming can avoid slow control loops. Rich clients are a common trend in multimedia communication. Hence, the assumption that a streaming client must be kept simple, may not be true anymore. This gave client-driven adaptive streaming a huge momentum, because it simplifies adaptive video streaming significantly. It concentrates the adaptation logic at the client, while the server component is reduced to a service. The server only transmits the data that the client requests. There are two main approaches to client-driven adaptive streaming, which differ in timeliness of delivery and adaptation granularity.

#### Subscription-based Adaptive Streaming

In contrast to server-side streaming, subscription-based adaptive streaming utilizes either multiple representations of the content or scalable (layered) media [56]. The client decides on which representation or how many layers it will subscribe to, based on the network conditions and the buffer level (feedback-based rate allocation [56]). After subscription, the video layers are continuously streamed from the server to the client. If the video quality has to be changed, the client issues a further request to the server.

The most prominent example is receiver-driven layered multicast (RLM) [61, 78] based on RTP/UDP (see Figure 4.6). In this system, the client retrieves information on the bandwidth requirements for each layer from the server and decides based on the currently available bandwidth which layers can be retrieved. The client subscribes to one or more

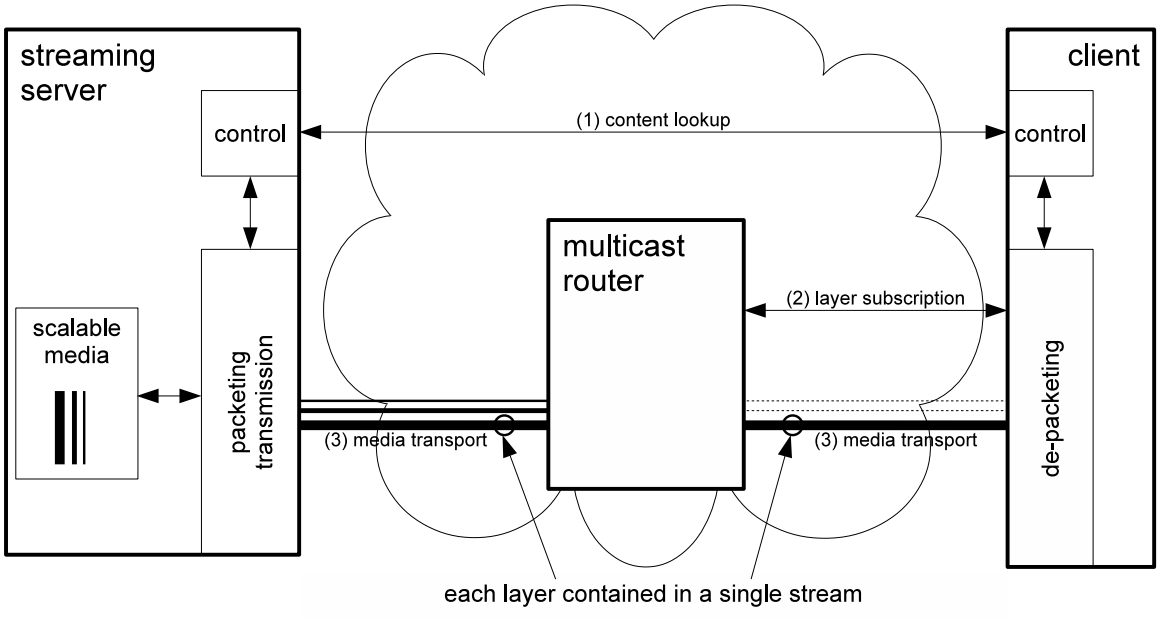



Figure 4.6: Architecture based on receiver-driven layered multicast.

layers by joining the specific multicast groups. The video data is delivered via multicast networks to the client.

Multicast networks are widely used for broadcast-like video delivery in IPTV networks, but they are usually not used for video-on-demand because of the high complexity [58]. RLM enables adaptive streaming with low switching delays, because the adaptation decision has to be propagated only to the next multicast router to initiate the subscription change. For Internet deployments, the main problem of RLM is that it requires a multicast infrastructure. In addition, an Internet video streaming system has to provide congestion control. But the implementation of congestion control in multicast networks turns out to be complicated [56].

While it is possible to use TCP or other transport protocols in subscription-based adaptive streaming, they introduce much more complexity and inefficiencies than RLM with RTP/UDP. Regarding TCP, which is an end-to-end protocol (unicast), the individual layers should be streamed out synchronously, which may be difficult because not all TCP streams may achieve the required throughput. In addition, the adaptation delay (the change of the subscription) is increased, because the server has to be informed of the change instead of

the next router.

#### HTTP-based Request-Response Streams for Adaptive Streaming

Adaptive HTTP video streaming breaks with the traditional definition of streaming. Traditionally, streaming is seen as an active process on the server, sending continuously data from the server to the client. HTTP-based request-response streams are different, because the active process runs on the client. The client continuously creates requests for data and the server answers the requests (by responses), generating a continuous data stream to the client [52]. Compared to the previous approaches, also the streaming logic is now implemented on the client.

The adaptation logic is similar to the one in the subscription-based approach. The client monitors the network and its buffer level and decides which parts of the content to request next. For that reason, the client has to know in advance about the available parts of the content. While the HTTP-based request-response streams introduce notable overhead compared to a single TCP connection, they also exhibit favorable characteristics (e.g., request-response streams can cope with packet loss and connection aborts). An in-depth analysis on request-response streams will be given in Chapter 6.

### 4.4.3 In-Network Adaptation

In general, video streaming can be seen as a form of communication between a streaming server and a client. In-network adaptation is quite different from client-server adaptive streaming, because it deploys video proxies - also called *Media-Aware Network Elements (MANEs)* - in the network, instead of placing adaptation logic on the server or the client. The MANE concept was introduced together with the RTP payload format of H.264/SVC [107, 108]. Different types of network elements are defined in RTP, including the concept of a *mixer* [85], which is the foundation of an adaptation MANE. RTSP/RTP challenges in-network adaptation in many different ways.

An RTP mixer (see Figure 4.7) acts as an endpoint for incoming RTP streams and creates new outbound RTP streams with a different synchronization source (SSRC). The processing/adaptation on the RTP mixer does not lead to inconsistent RTCP sender and

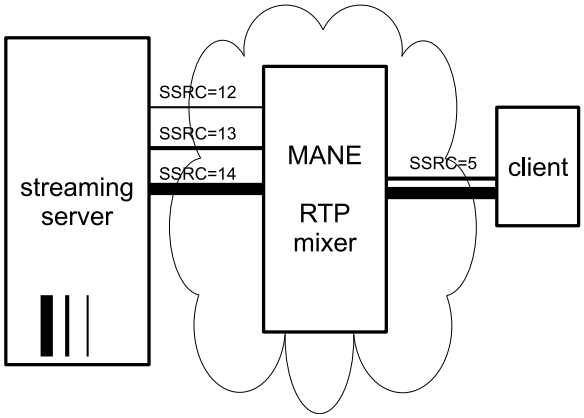



Figure 4.7: Simple RTP mixer [54].

receiver reports for both sides (server–mixer, mixer–client), because the incoming and outgoing RTP streams are decoupled. In addition, there are no gaps in the RTP packet sequence numbers. The requirements on such an RTSP signaling-aware RTP mixer (MANE) can be summarized as follows [108]:

- *Endpoint functionality.* The MANE acts as an endpoint to the server and creates new RTP streams for the client.
- *Signaling awareness.* The MANE needs to listen to the RTSP communication to identify which RTP streams contain adaptable content or metadata.
- *Stateful operation.* A state has to be associated with the RTSP communication/sessions and is needed for the processing of the RTP streams.
- *Security context.* The MANE has to be in the security context, otherwise it will not be able to listen to the RTSP signaling.

The H.264/SVC-based adaptation MANE shown in Figure 4.8 acts as an RTP mixer, which receives and delivers the video data in a single unicast RTP stream [54, 45]. RTSP requests from the client are handled by the MANE and converted into new RTSP requests for the RTSP/RTP server serving the content. The server returns a description of the RTP streams by means of the SDP protocol [34]. A new state is created on the MANE, including the RTSP session and SDP information. The RTP mixing includes the full de-packeting of the incoming RTP streams and the processing/adaptation on bit stream level. After the

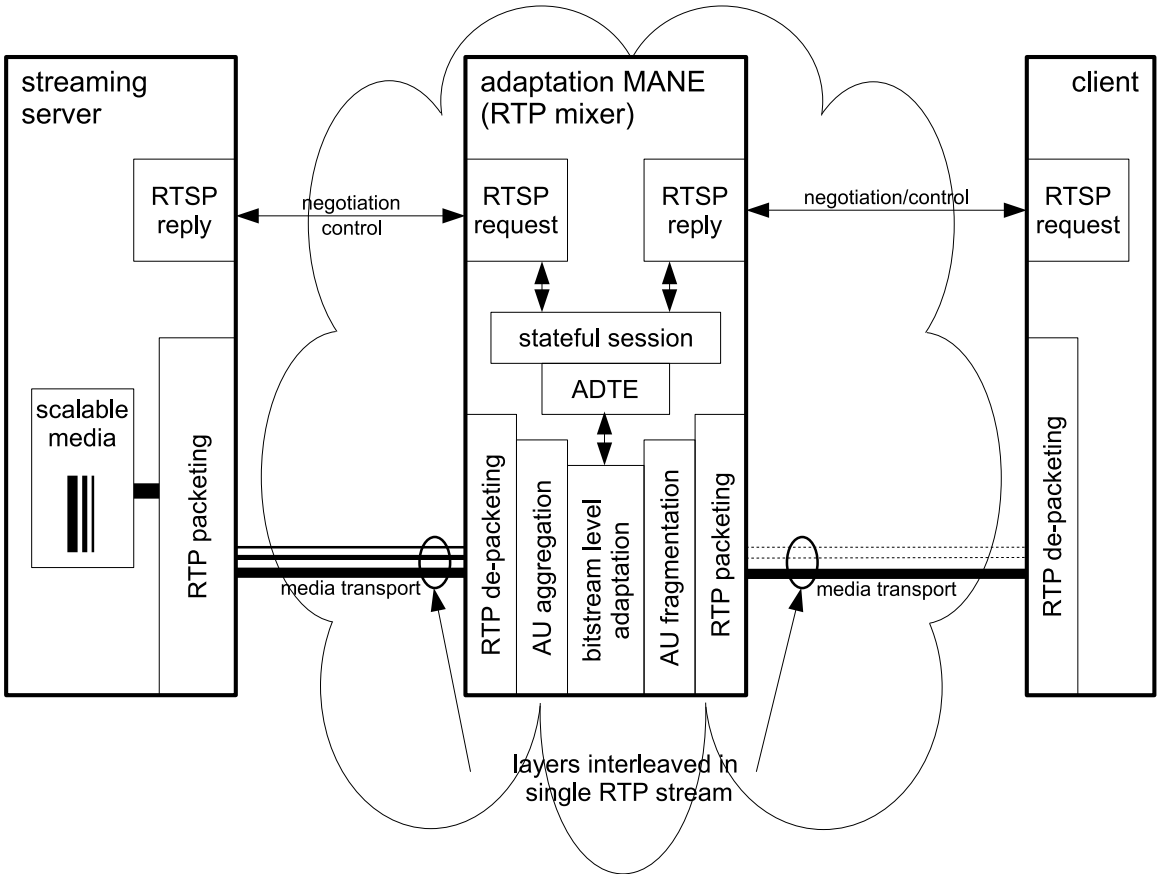



Figure 4.8: Adaptation-enabled MANE based on RTSP/RTP [54].

adaptation, the H.264/SVC bit stream is packed with a new SSRC and sequence number and delivered to the client. The *bit stream level adaptation* component is exchangeable and enables an easy replacement of the adaptation mechanism. It is steered by an *adaptation decision taking engine (ADTE)*, which supplies information on how to actually adapt the media bit stream. Adaptation decision taking chooses the best video representation by taking the user preferences, network conditions, terminal capabilities and natural environment descriptions into account [47].

A MANE can also be used for other purposes than adapting the video data to the available bandwidth or user preferences. It makes sense to deploy a MANE to avoid NAT and firewall traversals [107]. In addition, a MANE can aggregate one or more video streams into a single RTP stream tailored to the client's requirements. This is especially useful when building multicast-to-unicast gateways [108].

### 4.4.4 Hybrid Systems

The different types of architectures are a result of the different requirements of the streaming applications. For that reason, it is only natural that also hybrid systems have been deployed. An example is Akamai's adaptive HD video streaming service. The client reports the buffer level back to the server, where the adaptation decision is taken [15]. While this is exactly the way the server-side adaptation based on client feedback approach works, the system differs in the transport of the video data. The client handles the streaming process by pulling the video data from the server via HTTP. The reason for this somehow sub-optimal strategy may be that the company wants to stay in control of the adaptation logic and therefore places it on the server [16]. On other hand, they may use HTTP because it can easily be deployed and can cope with NATs and firewalls. Hence, it can be assumed that in real-world applications not only the technical superiority is important, but also that the system can be administered in a centralized fashion.

## 4.5 Related Work

Because of the increasing popularity of Internet video streaming, TCP- and HTTP-based adaptive video streaming gained a lot of momentum. In the following, the most prominent

work on TCP-based adaptive streaming will be reviewed. Additionally, emerging standards and applications for HTTP-based adaptive streaming are presented.

### 4.5.1 Related Work in Research

A lot of research was done in the field of TCP-based streaming. The requirements needed to stream constant bit rate (CBR) video via the Internet were investigated in [104]. To avoid jerky playback and high start-up delays, it is suggested that the network bandwidth should be twice the video bit rate. While this is feasible for low resolution video content like user generated content, problems arise when streaming HD videos in good quality.

To counter these problems, significant research on enhancing TCP's performance was done. New TCP flavors like TCP CUBIC [33] or Compound TCP [96] aim to improve the TCP throughput in networks with high bandwidth-delay-products. While these implementations achieve better link utilization, they are usually unfair to TCP Reno [63]. Nevertheless, all TCP variants experience similar performance problems with unanticipated packet loss, because the congestion control usually assumes that the packet loss is due to congestion.

A different way to cope with the shortcomings of TCP is the use of application-layer protocols, which aggregate multiple TCP connections for a single use [97, 43, 99, 35]. If ignored, this may hurt the concept of TCP-friendliness, which expects all concurrent data flows to share the available bandwidth in a fair manner. Aggregating multiple TCP streams for a single use is seen as unfair to concurrent single connections, because more bandwidth than the fair share is utilized. While being able to improve the throughput performance, the solutions in [99] and [97] exhibit this unfairness. The protocol described in [43] provides TCP-friendliness to concurrent single TCP connections, by adjusting the stream-out rate of each TCP connection such that the aggregated throughput corresponds to that of a single TCP connection. The system described in [35] modifies the TCP implementation to delay the ACKs. The resulting submissive TCP connections were aggregated and should exhibit the same TCP-friendliness as a single TCP connection. The presented approaches mainly focus on enhancing the throughput of the streaming systems, enabling a better utilization of the available bandwidth.

TCP-based adaptive streaming is able to cope with congestion, by adjusting the video bit rate to the actual TCP throughput. Different approaches to adaptive TCP streaming can be found in the literature [31, 32, 22, 35, 49, 50, 77]. To enable low-latency applications, modifications to the TCP implementations were proposed in [31, 32]. Low-latency applications have to react fast to changing network conditions in order to avoid delays in the transmission. Delayed data packets are usually discarded if they do not meet the real-time requirements of the application. The idea of [31, 32] is to keep the TCP socket buffer at the sender as small as possible, minimizing the amount of data in the transmission pipeline. Hence, an adaptive streaming application can react fast to changed network conditions by adapting the content to the available bandwidth.

Early work on priority streaming [22] made use of the reliable and ordered data transfer of TCP. By reordering the video bit stream syntax elements in terms of priority and sending the more important parts of the video before the low-priority parts, it was possible to adapt the video quality to the current throughput of the used TCP connection. The video content is split into video fragments, which are reordered in terms of priority. For each video fragment a maximum transmission time is specified. The reordered video fragment is transmitted over a single TCP connection to the client till the accounted transmission time is used up. This priority-based technique is based on temporal scalable video, enabling graceful degradation of the received video.

Usually, scalability in the signal-to-noise (SNR) domain is better suited for Internet video streaming, because it produces less perceptible artifacts. Hence, priority-progress streaming [49, 50] extended the idea of [22], using a scalable video codec in order to deliver a rate-distortion optimal video. Because priority streaming is based on video fragments, also the influence of fragment sizes was investigated. Additionally, dynamic fragment sizes were used to decrease the start-up delay, because the start-up delay is usually dictated by the fragment size.

Adaptive video transcoding was used in [77] to adapt the video content to the current TCP transmission rate. A delivery module feeds information on the current transmission rate back to an adaptation module. The adaptation module uses the transmission rate as a bandwidth estimate, which is used to steer the adaptation of the video. The transcoder module provides adaptation capabilities in the SNR domain. By changing the quantization

of the transform coefficients of the input video, the video bit rate can be adjusted to match the bandwidth estimate.

### 4.5.2 Standards and Applications

As mentioned before, TCP has become the de facto standard protocol in the Internet. Hence, adaptive video streaming using HTTP, which is based on TCP, has recently become quite popular. In [114], Microsoft introduced HTTP-based adaptive streaming with their Smooth Streaming System. Instead of a single video download, multiple small HTTP downloads (of so-called video fragments) are performed. Each video fragment comprises several seconds of video and is usually aligned with the GOP boundaries of the video. To enable adaptive streaming, for each video different representations of different quality exist. By changing representations after each finished download, it is possible to switch the media bit rate (and hence the quality). The video fragments are consecutively transferred to the client, where the video is decoded and displayed. The system can react to changing network conditions by switching to different representations of the video content.

There are a lot of similar HTTP-based adaptive streaming systems, the most prominent are Move Networks [64], Apple's HTTP Live Streaming [70], the Akamai HD Network [15] and Adobe's HTTP Dynamic Streaming [36]. The adaptation system of the approaches works in a similar manner, because they utilize stream switching [92] to adjust the stream-out rate to the available bandwidth. While all approaches rely on a set of video representations with different qualities, the main difference between the systems lies in the metadata for describing the media content and in the media container format [80]. Additionally, the algorithms for adaptation decision taking are different, which decide when to switch from one media quality to another [6, 112, 66]. The majority of the streaming systems use stream switching to adapt to the changing network conditions, without taking advantage of a scalable video codec (like H.264/SVC [109]) which enables fine-grained adaptation. The video fragments are mainly transmitted via HTTP's request-response paradigm, because HTTP can easily be deployed in existing network infrastructures.

Adaptive HTTP streaming gained a huge momentum in the last years, because more

and more consumer devices are connected to the Internet. To enable interoperability, it was also necessary to standardize the interfaces between the streaming server and the client. In HTTP-based streaming systems, the interfaces usually comprise of a description of the video content, its different representations and its location. The 3GPP Adaptive HTTP Streaming standard [4] additionally extends this interface by detailed video characteristics, dependencies and application-specific metadata. The interface also includes how the video data can be accessed and how long the resource will be available, which is useful in case of live streaming. But the standard does not describe how to implement the adaptive streaming process itself. 3GP-DASH (Progressive Download and Dynamic Adaptive Streaming over HTTP) [5, 91] extends the adaptive HTTP streaming standard of 3GPP by supporting scalable video. The MPEG Dynamic Adaptive Streaming over HTTP (DASH) standard [37, 65] is partly identical to 3GP-DASH, but implements additional use-cases which may be needed in large scale deployments.

CHAPTER

# 5 TCP-based Adaptive Video Streaming for the Internet

TCP-based adaptive video streaming is mainly deployed for video-on-demand, allowing several seconds of startup delay. Stream switching and priority streaming are the most suited approaches for non-real-time adaptive video streaming, because the changes of the video quality are kept low. Although rate-control can offer this capability too, it performs quite different in terms of scalability and network efficiency with multiple clients (see Section 4.1 for more details). For that reason, rate-control is not considered a good candidate for video-on-demand.

In this chapter, an in-depth evaluation of three existing TCP-based adaptive video streaming approaches will be presented:

- *Stream Switching using Application-layer Bandwidth Estimation*
- *Stream Switching using TCP-Stack-based Bandwidth Estimation*
- *Priority Streaming*

The mentioned approaches can utilize the scalability features of the H.264/SVC video codec [109] to enable fine-grained adaptive streaming. In the evaluation, different metrics will be used to benchmark the systems under diverse network conditions. The average quality of the received video indicates how well an approach can make use of the available bandwidth. The variance of the quality shows how the approaches cope with TCP's fluctuating throughput. In addition, the buffer level on the client and its variance is monitored to assess the quality of the client status estimation. An evaluation of the TCP-friendliness

will show if the adaptive approaches utilize their fair share of the network's bandwidth. The contribution in this chapter is an analysis of the baseline performance of TCP-based adaptive streaming; the results represent a reference for further investigations.

In the following, TCP-based adaptive video streaming is reviewed and the an outline of the different approaches will be presented.

## 5.1 TCP-based Adaptive Video Streaming

Due to the easy deployment and congestion awareness of TCP, video streaming based on TCP has become very popular. Currently, a lot of content providers use TCP for streaming multimedia content over the Internet [81, 49]. TCP performs very well in networks with small round-trip times (RTTs), because it uses acknowledgements to signal a successful transmission. But TCP has performance issues in networks with high bandwidth-delay products, like the Internet. TCP's congestion control is based on the additive-increase/multiplicative-decrease (AIMD) algorithm. The AI step tries to increase the congestion window until a packet is lost. On packet loss, the MD step reduces the TCP window (and therefore the throughput) drastically. This leads to a notable variation of the throughput in networks with large RTTs, because the time to recover from a packet loss is directly dependent on the RTT. In [104], an extensive analysis of TCP streaming was conducted. The analysis shows that TCP streaming achieves good results when the available bandwidth is twice the media bit rate. While this kind of overprovisioning is feasible for streaming media at low bit rates, high-definition media demands for significantly higher throughput. Currently, a large number of last-mile networks offer downlink bandwidths greater than 4 Mbps [19]. But providing twice the bit rate of an HD video stream turns out to be difficult. Another problem of TCP-based adaptive video streaming is error recovery on connection abort or stalls. Usually, this has to be handled on the application layer.

In addition to the systematic problems of TCP, the performance is also very dependent on the implementation of the TCP stack and the applications using TCP. The TCP stack is in general implemented in the kernel space of the operating system to provide high performance and avoid security issues. To access TCP from the user space, the TCP socket API and a socket buffer have to be used. The size of the TCP socket buffer directly influences the TCP performance, because it limits the maximum throughput by restricting

the maximum size of the congestion window [113]. Usually, it is recommended to set the TCP socket buffer size to twice the value of the bandwidth-delay product (BDP) [113].

In adaptive video streaming, it is not the goal to utilize the network link beyond the maximum video bit rate. The main goal is to transmit the video data in such a manner that jerky playback is avoided. Large TCP socket buffers usually increase the adaptation delay, because the data in the socket buffer cannot be changed any more. Thus, it is recommended to keep the TCP socket buffer size as small as possible for adaptive systems. In the following, the TCP socket buffer size will not be set considering the BDP but the maximum video bit rate (see Equation 5.1). To avoid a server overload, the stream-out rate is restricted to be approximately the maximum video bit rate.

$$TcpSockBufferSize = 2 * VideoBitRate * RTT \tag{5.1}$$

In the literature, different approaches to adaptive video streaming exist (see also Section 4). For this evaluation, the two approaches appropriate for TCP streaming were selected, which are *stream switching* and *priority streaming*. Both approaches only observe the TCP channel and utilize the gathered information in the adaptive streaming process. The streaming systems do not change the behavior of the TCP implementation or the network protocol stack of the operating system. In addition, two different types of bandwidth estimation (BE) are evaluated for stream switching. In the following, a total of three different approaches to adaptive TCP streaming are briefly recapitulated. The realisation by using H.264/SVC will be further explained in Section 5.2.

**Stream Switching using Application-layer Bandwidth Estimation (APP-BE):** The idea is to measure the time spent for the transmission of a specific media block and estimate the available bandwidth from it. In [77], a system utilizing application-layer bandwidth estimation is presented. A delivery module writes the media data into a blocking TCP socket and calculates the bit rate by simply counting the bytes for a specific period of time. One of the benefits of application-layer bandwidth estimation is that it does not rely on any operating system specific information. The estimated value of the available bandwidth is used in the adaptation process to configure a media transcoder. As shown in [77], it can adapt to the available bandwidth. The application-layer bandwidth estimation

works also in combination with stream switching (see Section 4.2), where it can be used for selecting the video representations.

**Stream Switching using TCP-Stack-based Bandwidth Estimation (TCP-BE):** The current congestion window and the estimated RTT of a TCP connection can be used to estimate the available bandwidth [60]. In [9], a simple model for the expected latency of the packets sent with TCP Reno is presented. Additionally, this model also estimates the available channel rate and the expected distortion. The TCP parameters are used to steer the encoding process of the video, allowing the streaming system to adapt to the available bandwidth. But this type of bandwidth estimation can also be used in stream switching, where it steers the adaptive selection of the video representations.

**Priority Streaming:** In contrast to traditional buffering at the receiver, which only tries to overcome bandwidth shortages or bandwidth estimation errors, priority streaming [22, 50] tries to average the video quality over a period of time (see also Section 4.3). A video sequence is split into fragments, comprising the same play-out duration. The idea is to rearrange the video syntax elements (e.g., slices, frames, layers) in priority order. A reordered video fragment is called video segment. As a result, a video segment can be truncated at virtually any point. The quality of the video segment depends only on how much of the video segment is retrieved.

TCP-based priority streaming uses a single TCP connection for the transmission of the video data. The video segments are sent sequentially from the server to the client. The server maintains a timeout for each video segment, which can be considered as the maximum download duration of the segment. When reaching the timeout, the server stops transmitting the segment and switches to the next one. At the client, the video is reconstructed from the received (and possibly truncated) video segments and displayed to the user.

The next section will show how these three approaches can be combined with H.264/SVC, followed by the implementations used in the evaluation.

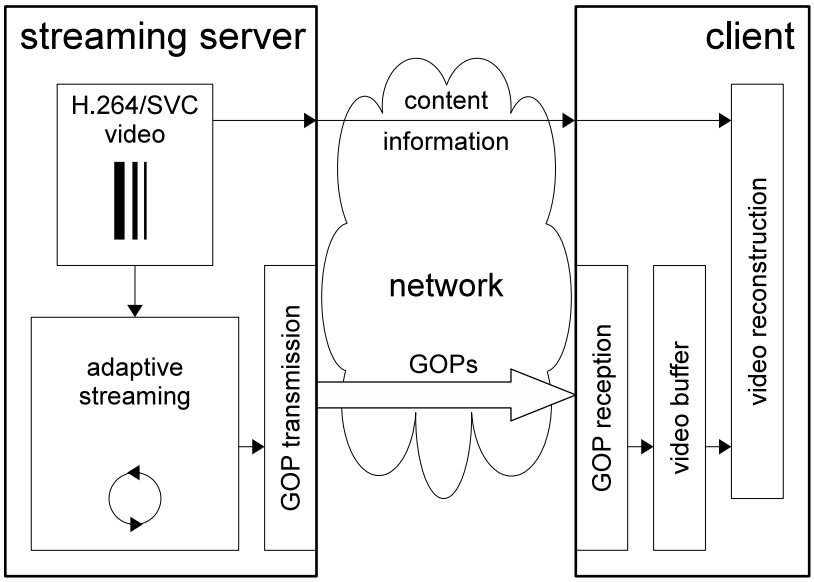



Figure 5.1: Server-side adaptive streaming system [51].

## 5.2 TCP-based Adaptive Streaming of H.264/SVC

For the evaluation, the adaptive streaming approaches mentioned above were adapted for use with H.264/SVC. To asses the capabilities of the different approaches, an architecture using server-side adaptation based on estimation of the client status was chosen [51]. The rationale behind this choice is an easier comparison of the adaptive streaming solutions, because the adaptation logic and the streaming logic are placed on the server. It is also aligned with the classical view of streaming, where the server pushes the media data to the client. Because the number of quality changes should be comparable for all approaches, the video fragments are restricted to comprise exactly one GOP. Hence, the transmission of the video is performed on a per-GOP basis. The common architecture for all server-side streaming solutions is shown in Figure 5.1.

At the server, the video is stored as a scalable H.264/SVC bit stream. The adaptive streaming component adjusts each video fragment/GOP based on the adaptation decision, before it is transmitted to the client. At the client, the GOP is received, buffered and decoded for playback. All three approaches do not use client feedback (like buffer level, etc.) to steer the adaptive streaming process. Only the implicit feedback of the TCP connection is used as an indicator of the transmission progress. For instance, blocking write

operations on the TCP socket indicate the connection status.

Unlike in progressive download, where the client steers the download rate by changing its advertised window (TCP's flow control), in server-side adaptive streaming the timely transmission of the video is controlled at the server. On the one hand, this means that the server streams out the content at the same speed the client consumes it. The advantage of the real-time play-out is that a single client with a fast network connection cannot overload the streaming server by consuming more than its fair share, which is at most the maximum bit rate of the video. On the other hand, this server-side scheduling mechanism is further necessary to prevent video fragments from arriving too late at the client. Since late fragments drain the buffer at the client and cause the video playback to stop at the player, all three approaches employ counter-measures to prevent this buffer drainage. In the following, the implementations of the three approaches are described in detail.

### 5.2.1 Stream Switching using Application-layer Bandwidth Estimation

The first stream switching approach (see Section 4.2) is based on bandwidth estimation that is done at the application layer (APP-BE). To estimate the available bandwidth, the number of bytes sent through the TCP socket for a time interval is measured. The measurement is performed on a per-GOP basis, because the smallest transmission unit is a single GOP. By measuring the time that is needed to send the GOP via the socket to the client, the current throughput of the TCP connection can be calculated. For future GOP transmissions, a bandwidth estimate is computed as the average of the last five throughput measurements. Based on that estimate, the next GOP is adapted by selecting a representation of the H.264/SVC bit stream with a bit rate below the bandwidth estimate. Additionally, the estimate is adjusted by the buffer control algorithm to prevent the client buffers from draining in case of delayed GOP transmissions. Delayed GOP transmissions are partly an effect of TCP's reliable transmission system. In addition, changes in the network (like congestion) may reduce the available bandwidth and thereby increase the time needed for transmitting a GOP. If the delivery of a GOP is delayed, all GOPs sent afterwards are delayed too. To correct this mismatch, the bit rate of the GOPs has to be adjusted in order to enable timely delivery.

More formally, the adaptive streaming algorithm can be described as follows. Let $d^{gop}$ be

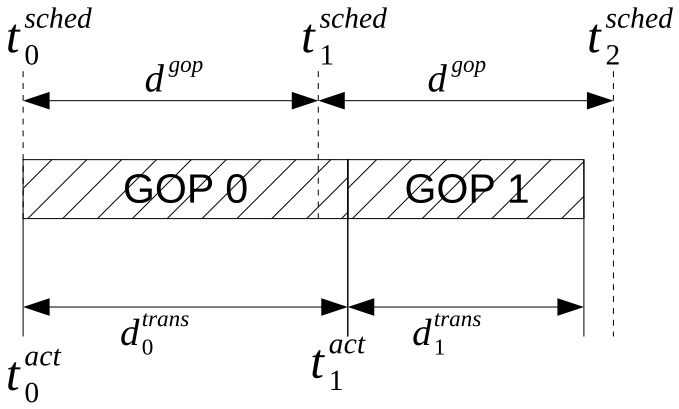



Figure 5.2: GOP timing [51].

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, one can define $t_0^{sch} = 0$ and the scheduled start of any other GOP $i$ as $t_i^{sch} = t_0^{sch} + d^{gop} * i$. To detect a potential drift in the start time, the actual start time of each GOP ($t_i^{act}$) at the moment before the first byte of a GOP is sent via the socket API is recorded. After the successful transmission of 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 5.2. Given the length of the GOP in bytes ($l_i^{gop}$), the measured throughput for the transmission of a GOP $i$ can be calculated as $th_i = l_i^{gop}/d_i^{trans}$. The estimated bandwidth available for the transmission of the next GOP $i+1$ is the average of the last five throughput measurements, corrected by a multiplicative term. This correction prevents that the scheduled start times $t_i^{sched}$ and the actual start times $t_i^{act}$ begin to diverge and is called *buffer control* in this work. The buffer control on the server is used to prevent the client buffer from either overflowing or draining. For that reason, the buffer control function $f_{bctl}(delta, d^{gop})$ depends on the difference between both timestamps and the duration of one GOP ($delta = t_i^{act} + d_i^{trans} - t_{i+1}^{sch}$). The function is illustrated in Figure 5.3 and described in more detail in the section below. Finally, the estimated bandwidth ($be$) for GOP $i+1$ is defined as follows [51].

$$be_{i+1} = \frac{1}{5} \sum_{j=i-4}^{i} th_j \cdot f_{bctl}(t_i^{act} + d_i^{trans} - t_{i+1}^{sch}, d^{gop}) \tag{5.2}$$

The estimated bandwidth is used for selecting the quality of the next GOP. Each GOP

is encoded using the H.264/SVC video codec (see Section 2.1.2 for more details) and uses the priority id to define different representations of the content. Using this semantic, up to 64 representations can be extracted out of an H.264/SVC bit stream. For each representation, the resolution, bit rate and other parameters are stored in a scalability information SEI message [109]. This bit rate information is then used for selecting the representation of a GOP in a way that it matches the target bandwidth.

Because the stream out rate of the video is limited to the maximum bit rate of the video (real-time), a constraint is imposed on the transmission start of each GOP. The transmission should not start earlier than scheduled, i.e., $\forall i : t_i^{act} \geq t_i^{sched}$. In addition, it is required that the size of the GOP is larger than the TCP socket buffer on the server, to be able to measure the throughput by means of the blocking TCP socket API. If the GOP size is too small, the socket implementation may simply copy the video data into the TCP socket buffer and return immediately from the sending operation, which would render the throughput measurement useless. On the server, the TCP socket buffer should be adjusted according to the bit rate of the video and the envisaged RTT range, to guarantee consistent measurements (see Equation 5.1).

**Buffer Control Function**

Adaptive streaming based on bandwidth estimation heavily relies on the accuracy of the estimate. If the bandwidth estimate is too much off, it may happen that the buffer on the client drains. The buffer control function steers the buffer usage at the client by adjusting the estimated bandwidth. Without any adjustment, the scheduled start time and the actual start time of each GOP's transmission usually drift apart. On the other hand, throttling the used bandwidth too much may lead to an under-utilization of the available bandwidth (fair share) of the network link. As a result, a TCP connection which does not participate in the competition with possible concurrent TCP connections may suffer from starvation.

The buffer control function, as shown in Equation 5.3 and Figure 5.3, is designed by considering the following situations [51]. If the difference between scheduled and actual start time (*delta*) is small - less than 5 percent of the GOP play-out duration, it can be assumed that more than sufficient bandwidth is available for the transmission of the video

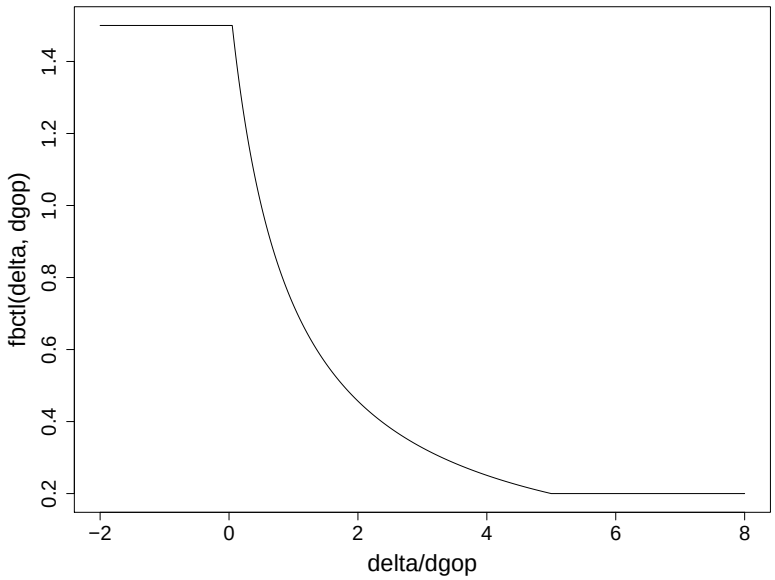



Figure 5.3: Buffer control function [51].

data. For that reason, the buffer control function increases the bandwidth estimate by 50 percent, to enable TCP to acquire the fair share of the network bandwidth. On the other end, if *delta* is very large and exceeds five times the GOP duration, the actual transmission of the GOP lags significantly behind the scheduled transmission. In that case, the rate for video stream-out is reduced to be 1/5 of the estimated bandwidth. No adjustments will be made if *delta* is exactly the half of the GOP duration (function value is 1). Using these three operating points as a basis, 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 5.4.

$$f_{bctl}(delta, d^{gop}) := \begin{cases} 1.5 & \text{if } delta/d^{gop} < 0.05 \\ 1/5 & \text{if } delta/d^{gop} >= 5 \\ \psi(\frac{delta}{d^{gop}}) & \text{otherwise} \end{cases} \tag{5.3}$$

$$\psi(x) := \frac{1.46}{x + 0.893} - 0.0476 \tag{5.4}$$

### 5.2.2 Stream Switching using TCP-Stack-based Bandwidth Estimation

The second approach using bandwidth estimation uses information provided by the TCP stack (TCP-BE). The algorithm works similar to the application layer mechanism introduced before. First, an estimate of the bandwidth is obtained and the representation of the next GOP to be transmitted is selected based on that estimate. The estimate is adjusted again by a multiplicative term in order to limit the drift between the scheduled and actual transmission time of each GOP. The main difference of the two approaches is the way how the bandwidth estimate is calculated. While application-layer bandwidth estimation measures the TCP throughput at the application layer, this approach obtains the information about the TCP connection directly from the network stack. Unix-type operating systems like Linux provide the `getsockopt` API call, which - in combination with the option `TCP_INFO` - can be used to acquire the parameters needed for the bandwidth estimation. The throughput estimate can be calculated from the current congestion window size $cWnd$, the maximum segment size $MSS$, and the estimate of the round trip time $RTT$ [60]. Following this idea, the estimated throughput can be defined for the $i$-th GOP as follows.

$$th_i = \frac{cWnd \cdot MSS}{RTT} \tag{5.5}$$

Like for the application-layer approach, the bandwidth estimate ($be$) for GOP $i+1$ can be determined using Equation 5.2, which also takes the buffer control function into consideration.

### 5.2.3 Priority-based Adaptive Streaming

While the previous approaches heavily rely on a correct estimate of the available bandwidth, the priority-based adaptive streaming (PRIORITY) algorithm does not require any estimate, because it is based on the priority streaming paradigm as introduced in Section 4.3. The video is split up into video fragments, where one fragment comprises exactly one GOP. Unlike the methods introduced before, priority streaming allows to truncate video fragments at virtually any point. The reason for that is the priority reordering process, which precedes the data transmission. As a consequence, the maximum transmission time for the rearranged GOP can be fixed to a certain value - usually the GOP play-out duration. During that time interval, the server pushes as much video data as possible to the client.

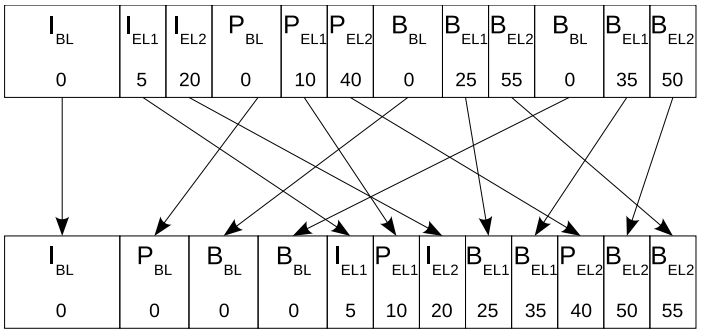



Figure 5.4: PRID-based NAL unit reordering [51].

After the reserved time for the GOP has passed, the server starts transmitting the next GOP.

In contrast to stream switching, which can also make use of non-scalable media without performance penalties, a prerequisite for priority streaming is the usage of scalable media. In case of H.264/SVC, each GOP of the 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 the enhancement layers. Usually, the bit stream is organized in decoding order which means that both kinds of NAL units are interleaved. To enable priority streaming, the NAL units have to be rearranged such that the NAL units are transmitted in order of their priority (usually indicated by the application-specific priority id). NAL units with identical priority ids are arranged in decoding order. In the context of video transmission, a lower value of the priority id signals a higher priority of the NAL unit [108]. The basic principles of the reordering process are illustrated in Figure 5.4. For the sake of simplicity, the figure shows a video bit stream consisting of only four pictures (I, P, B, B). Each picture has three different representations, which are 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 of each NAL unit. Because the bit stream cannot be decoded without the base layer, all NAL units of the base layer have the highest priority. The NAL units are ordered according to their priority to enable priority streaming. As a result of the scalable property of H.264/SVC, the quality of the decoded bit stream monotonically increases with the number of NAL units received by the client. Since the client requires the NAL units to be in decoding order, this transmission-specific reordering has to be reversed

before decoding. To enable fast reordering, proprietary reordering information is embedded into the transmitted bit stream. Although the reordering process can also be implemented using standard-compliant methods [46], for the sake of simplicity a proprietary approach was used.

The transmission schedule for priority streaming is similar to the one already defined for the previous two approaches. The scheduled start time for the transmission of the $i$-th GOP is denoted as $t_i^{sched}$. In general, the end of the transmission of 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 all NAL units via the TCP socket individually. After each send operation, the server checks if the time limit for the current GOP has been reached. However, each send operation itself is never preempted. In case of an overprovisioned network link, the transmission of a GOP will be usually finished before the time limit has been reached. However, the start of the next GOP's transmission will be delayed until the scheduled start time of the next GOP, to prevent excessive stream-out rates (similar to the stream switching approaches). If the available bandwidth 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 priority streaming does not make use of an explicit H.264/SVC adaptation process. The actual adaptation is performed by truncating the priority-ordered GOPs implicitly upon stream-out.

All approaches presented in this section rely on H.264/SVC adaptation based on the priority id mechanism. The Quality Level Assigner tool (QLA) of the JSVM software [40] was used to acquire information about the importance of each MGS NAL unit. The tool determines the priority for each NAL unit based on the rate-distortion of the reconstructed video. After assigning the priorities, the representations of the video can be extracted in a rate-distortion optimal way.

However, the tool defines the semantic of the priority id such that high priority id indicates a high priority of the NAL unit, which differs from the one specified in [108] (a low value of the priority id signals a high priority of the NAL unit). For that reason, in a second step, the priority id values of the output of the Quality Level Assigner have to be translated. Because only the QLA tool is used for assigning the priority ids, the evaluation is limited

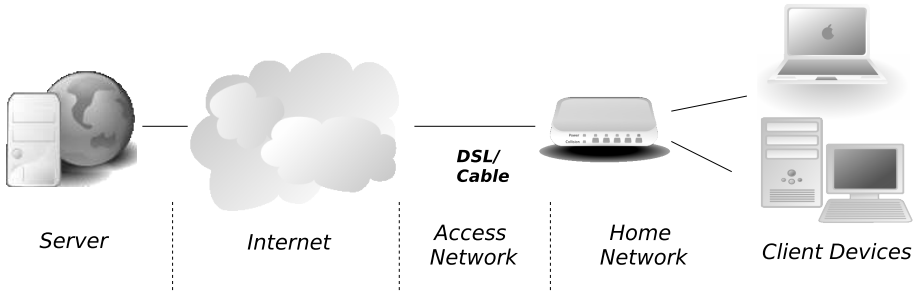



Figure 5.5: Use case for TCP-based Adaptive Streaming [51].

to adaptation in the SNR domain. Nevertheless, the adaptive streaming approaches can handle also spatial and temporal scalability. Sticking to SNR adaptation also enables easy comparison of the results in terms of PSNR, unlike in temporal or spatial adaptation where the direct comparison in terms of PSNR is seen controversial. MGS scalability is considered as most suitable in the context of Internet streaming, because it offers a wide range of bit rates. In contrast to temporal adaptation, MGS scalability enables smooth playback at the client and avoids severe quality changes that would arise when switching between different spatial resolutions. If continuous playback is of utmost importance, temporal and spatial scalability can be used as a fall-back solution to avoid jerky playback. This may be necessary in heavily congested networks, when the possibilities of SNR scalability get exhausted.

## 5.3 Evaluation of TCP-based Adaptive Video Streaming

Adaptive streaming based on TCP has to cope with various problems, like the significantly varying throughput and packet loss. The variance of the throughput of a TCP connection depends on both, the network link characteristics and the amount of congestion. TCP's performance deteriorates with unanticipated packet loss, because the congestion control assumes that the packet loss is due to congestion. To show the performance of the different adaptive approaches and not TCP's behavior in the presence of packet loss, packet loss is not considered in this evaluation. A typical use case for Internet video streaming is illustrated in Figure 5.5. It slightly differs from the main use case described in Section 1.1, which also takes packet loss into account. Several computers or consumer electronic devices in a home network are connected to the Internet via a DSL or cable-based access network.

In this deployment, the bottleneck link is assumed to be the access network (last mile). Concurrent TCP connections may cause congestion because of network link saturation or competition over the available bandwidth. This competition also influences the performance of an adaptive video streaming application.

The behavior and performance of the proposed adaptive streaming approaches will be evaluated under conditions typical for Internet streaming. These include under-provisioned network links and congested links due to competing TCP traffic. The average quality of the received video will show how well an approach can utilize the available bandwidth. The variation of the quality will indicate how well an approach can cope with the fluctuating throughput of TCP. To check the accuracy of the client status estimation, the buffer level on the client will be monitored. A further criterion for the evaluation is the TCP friendliness of the approaches, which is seen as crucial for applications intended to be deployed in the Internet. The stability of the Internet is based on the rationale that an application/protocol does not consume more than its fair bandwidth share. Since all presented approaches only make use of a single TCP connection, TCP-friendliness can be assumed.

### 5.3.1 Test Setup

The evaluation setup consists of six computers representing two servers, two routers and two clients as illustrated in Figure 5.6. Each node in the network runs the Ubuntu Linux operating system (kernel 2.6.27) and all client and server computers are configured to use the TCP Reno implementation. The routers use the Linux kernel module Netem [28] to perform the emulation of the network characteristics like delay, jitter and packet loss. The symmetric end-to-end delay of the Internet and the access 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 access network and limits the up- and downstream bandwidth (BW) between the Internet and the clients, allowing a maximum queuing delay of 200 ms. The asymmetric access network was emulated by setting the upstream bandwidth to 1/8 of the downstream bandwidth, which is common practice for DSL/cable networks. While the test setup is also able to generate packet loss in the access network, no packet loss will be emulated for this evaluation. The three TCP-based adaptive streaming approaches were implemented using the Python programming language. In this test setup, the adaptive

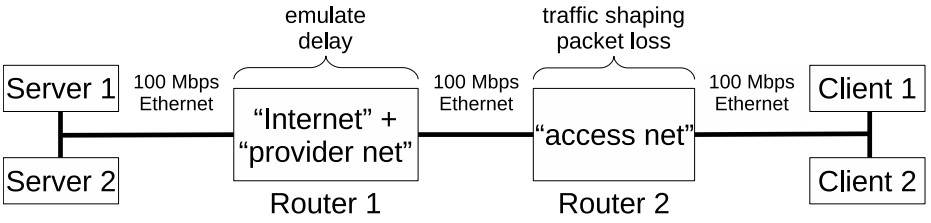



Figure 5.6: Test setup for TCP-based Adaptive Streaming [51].

streaming server is deployed on Server 1 while the client is located at Client 1. For the emulation of network congestion, Client 2 issues parallel infinite-source HTTP downloads from Server 2, which compete over their network share with the adaptive streaming traffic. The Apache HTTP Server [27] is used for serving the HTTP downloads.

### 5.3.2 Network Scenarios

For the envisioned use case, Internet video streaming, three different network scenarios are considered to be important because of the best-effort characteristics of the Internet. In the first scenario, an *overprovisioned network link* is used for the transmission of the video data, in which the available bandwidth substantially exceeds the video bit rate. The scenario is used to evaluate the ability of an approach to fully utilize the bandwidth for video transmission and the ability to cope with throughput variations. These throughput variations may occur in networks with high bandwidth-delay products, because of TCP's AIMD-based congestion control. At the other extreme, the adaptation ability of the approaches is analyzed in a scenario of an *underprovisioned network link*. This should demonstrate how well an approach can adapt to a specific network bottleneck bandwidth. In the third scenario, cross traffic congests the access network (last mile). To emulate the congestion, one, two and three concurrent infinite-source HTTP downloads are started in parallel to the video stream. 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, the approaches should make good use of their fair share of the network bandwidth and provide a TCP-friendly behavior.

Because TCP cannot cope with channel-related packet loss, an evaluation of the performance of TCP-based adaptive streaming in presence of packet loss was omitted. An

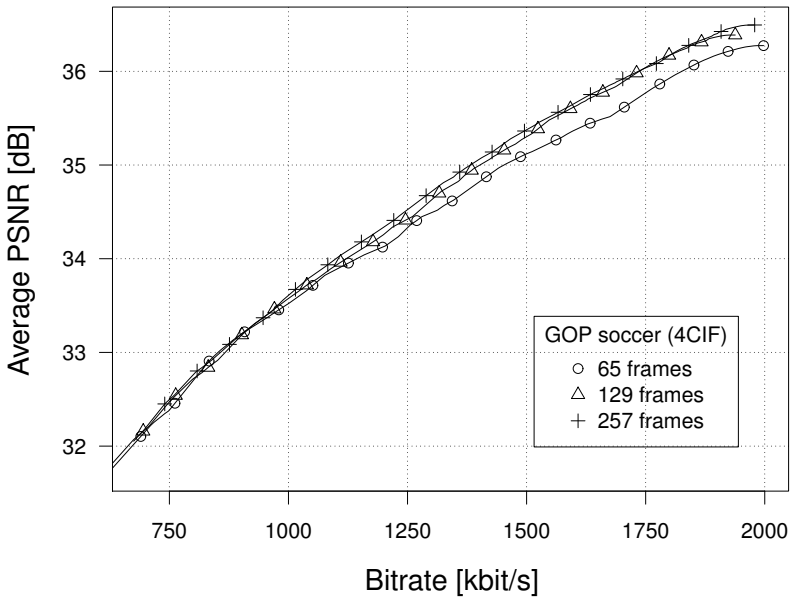



Figure 5.7: Rate-distortion curves of the test sequences [51].

in-depth evaluation on this topic will be given in Section 6.1.

### 5.3.3 Test Content

For the evaluation, the test sequences were created by means of the H.264/SVC codec provided by the Joint Scalable Video Model (JSVM) [40] 9.15 software. The video *soccer* in 4CIF resolution was encoded using three different GOP sizes. The first *65*, *129* and *257* frames of the video content form a test sequence, comprising 2.16, 4.30 and 8.57 seconds of video ($d^{gop}$) at 30 fps, respectively. Because of the dyadic prediction structures (hierarchical B-frames [87]) of the JSVM software, uneven GOP sizes were chosen. Dyadic GOP sizes generate B-frames at the end of each GOP, which results in missing P-frames for inter-frame prediction. Because the quantization parameters for B-frames are usually fixed to a specific value, the quality of the last B-frame would be bad. For that reason, the GOP size was incremented from $2^n$ to $2^n+1$ to enable the encoder to add an additional P-frame at the end of each GOP. By doing this, the prediction structure is closed once again and the quality of the last B-frame is good.

In addition, the test videos *crew* and *foreman* were evaluated. But they are omitted from

the presentation in this work, because they did not reveal any new findings. This is mainly because the adaptive streaming algorithms do not use any content-specific information. In case of stream switching, the bandwidth estimation suggests only a transmission bit rate, which can be used by the adaptation process to deliver a distortion-optimal video. The transmission system in priority streaming only relies on the priority-ordered bit stream of the scalable video. Hence, it is not content-aware at all.

Each test sequence consists of an H.264/AVC backward compatible base layer and one MGS quality enhancement layer (H.264/SVC). The 16 transform coefficients of the MGS quality enhancement layer are uniformly partitioned into four NAL units (4x4), resulting in five different quality representations for each video frame (base layer plus four quality enhancement layers). Because in MGS the quality can be changed on frame level, it is possible to extract more than five quality representations out of the scalable video bit stream (theoretically up to $1 + n_{frames} * n_{enhlayers}$). The Quality Level Assigner tool (JSVM) was used to assign 64 different priority ids to the NAL units based on rate-distortion values. Figure 5.7 shows the rate-distortion values of the test sequences, ranging from the lowest bit rate (highest priority) to the maximum bit rate (lowest priority). It can be observed that the sequences with smaller GOP sizes exhibit a slightly worse rate-distortion performance, because the prediction structures are limited by the GOP size. Although each test sequence comprises a different number of frames, it can be observed that the average PSNR values of the sequences are similar (see Figure 5.7). This simplifies the interpretation of the evaluation results, because it allows for a direct comparison of the results for different GOP sizes.

### 5.3.4 Evaluation Methodology

A typical application of TCP-based adaptive video streaming is video-on-demand. For that reason, the evaluation focuses on rather long stream-out durations and analyzes the received video quality and variation. The *soccer65* sequence is streamed 400 times in a loop, resulting in approx. 900 seconds stream-out duration. Because each test sequence has a different play-out duration and the approaches aim for real-time stream-out, the *soccer129* and *soccer257* sequences are streamed 200 and 100 times in a loop, respectively. As a result, the stream-out durations of all test sequences are similar, which enables a fair comparison of the evaluation results for the different GOP sizes.

The three network scenarios only consider different network conditions in the access network. To show the effects of different server locations also the RTT is varied, ranging from 50 ms to 200 ms. Each experiment is repeated three times and the values are averaged to reduce the influence of the run-time environment on the results. In the evaluation runs, no frames or GOPs were lost, because the adaptive streaming systems are based on the reliable transmission of TCP.

The size of the TCP socket buffer should be set based on the maximum throughput envisioned (see Section 5.1). When using overprovisioning, streaming systems should be able to utilize the additional bandwidth to enhance their streaming performance. For that reason, the maximum allowed TCP throughput is set to be 50 % higher than the video bit rate. Therefore, the TCP socket buffer size $l_{sock}$ is determined using 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* (see Equation 5.1). Although automatic TCP buffer tuning algorithms exist in modern operating systems [89], the effect of such an algorithm is very implementation specific. To be able to provide results independent of this buffer tuning implementation, a fixed buffer size was chosen. Hence, in the evaluation the TCP buffer size was set to $l_{sock}$ for all network scenarios and RTTs.

The metrics used for comparing the adaptive streaming approaches are the *reconstructed video quality in terms of PSNR*, the *deviation between the scheduled and the actual arrival time* of a GOP on the client $deltaC = t_i^{act} - t_i^{sched}$ and the variance of them. The value of *deltaC* corresponds to the buffer level in seconds on the client and will be monitored to check the accuracy of the client status estimation. For priority streaming, the absolute values of *deltaC* have to be interpreted with care. Priority streaming buffers at least one video fragment (see Section 4.3), because it needs to rearrange the received GOP into decoding order. In contrast to this, the APP-BE and TCP-BE approaches are able to start the decoding of a GOP immediately after the first frame of the GOP is received. A further criterion for the evaluation is the TCP friendliness of the approaches. All approaches should only make use of their fair share of the network's bandwidth.

The evaluation focuses on the dependability of the adaptation algorithms in terms of timeliness of delivery. To get a good starting point for the bandwidth estimation, the first GOP is transmitted in full quality.

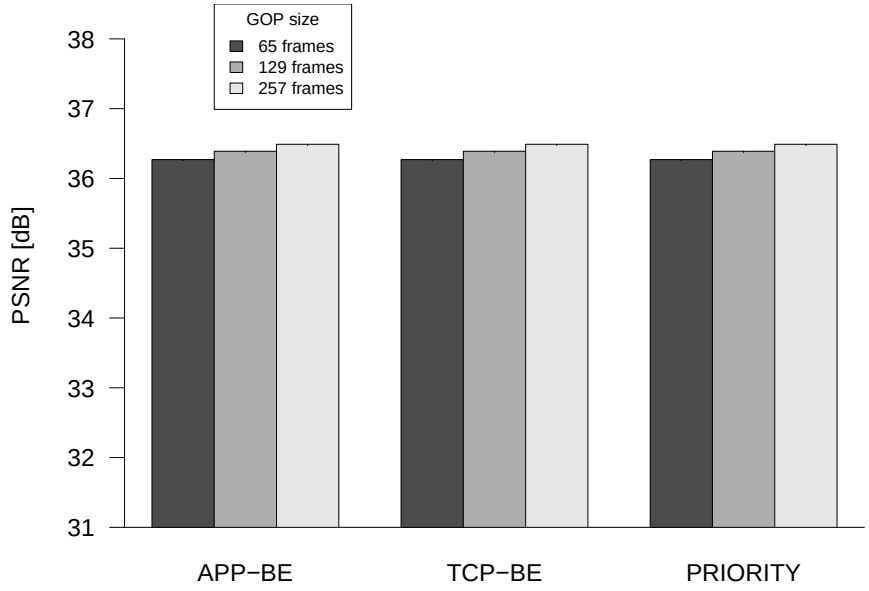
## 5.4 Results

The performance of TCP-based adaptive streaming is directly linked to the performance of TCP, which again is very dependent on the network conditions (see Section 5.1). For the assessment with respect to adaptive Internet video streaming, different network scenarios were introduced and the RTT was varied ranging from 50 to 200 ms. Because TCP performs best at low RTTs, the evaluation results are shown for the minimal RTT (in our use case 50 ms). Additionally, the results are shown for an RTT of 200 ms. In case of TCP-based adaptive streaming, this would represent the worst case scenario. In the following, a discussion of the evaluation results for each network scenario is presented.

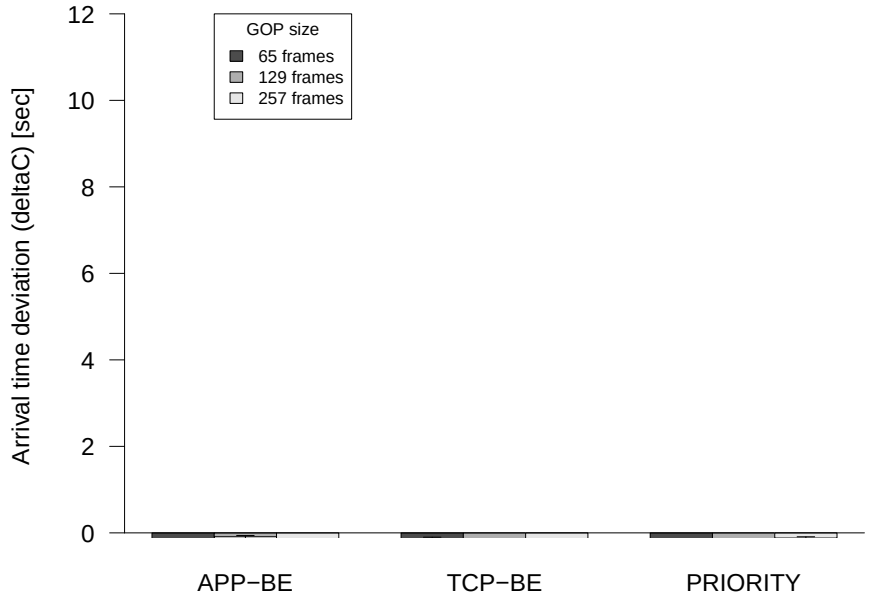
The performance of the adaptive streaming approaches in an *overprovisioned network* is shown in Figures 5.8 and 5.9. Stream switching and priority streaming perform very well and maximize the video quality by utilizing the additional bandwidth. This is supported by an investigation on constant bit rate streaming with TCP [104], which states that good streaming performance can be achieved if the available bandwidth is about twice the media bit rate. The small differences in the absolute PSNR values for the different GOP sizes are not transmission related, but caused by the better rate-distortion performance of larger GOPs (see Figure 5.7). The variation of the emulated packet delay and the dynamic behavior of TCP are leading to throughput variations, which result in the adaptation of some GOPs even in the case of overprovisioning. But of course this is only happening in the worst case with an RTT of 200 ms.

The timeliness of delivery of each GOP is monitored by means of the arrival time deviation of each GOP on the client (*deltaC*) for the different algorithms and GOP sizes. If the available bandwidth exceeds the bit rate of the adapted video, the delivery of a single GOP usually takes less time than the GOP play-out duration, resulting in *deltaC* values of zero. This is also supported by the results, as Figures 5.8(b) and 5.9(b) show arrival time deviations near zero. Because all GOPs arrive in time, only little buffering will be needed at the client. The standard deviation of the PSNR and the arrival time deviation is shown as error bars in the plots.

The ability of the streaming systems to adapt to a static bottleneck link with 1536 kbps

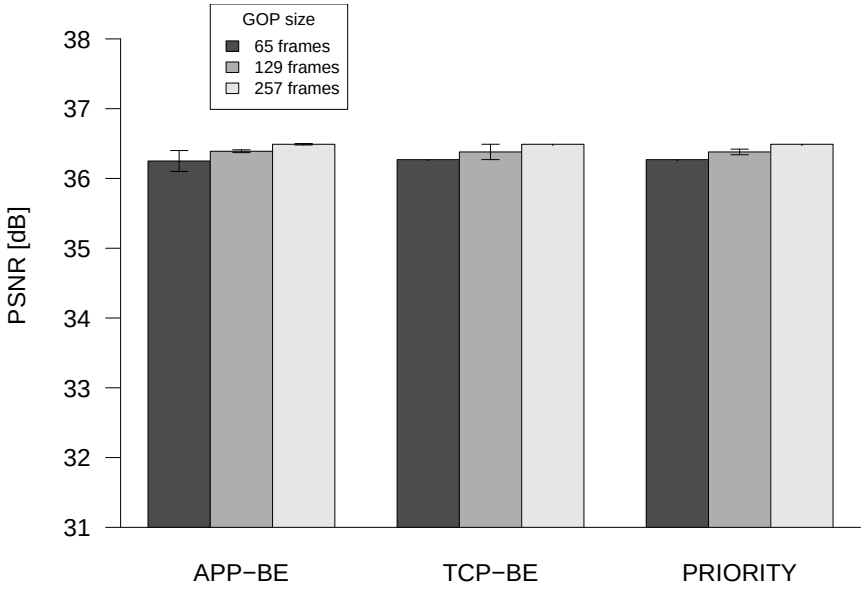



(a) *PSNR*

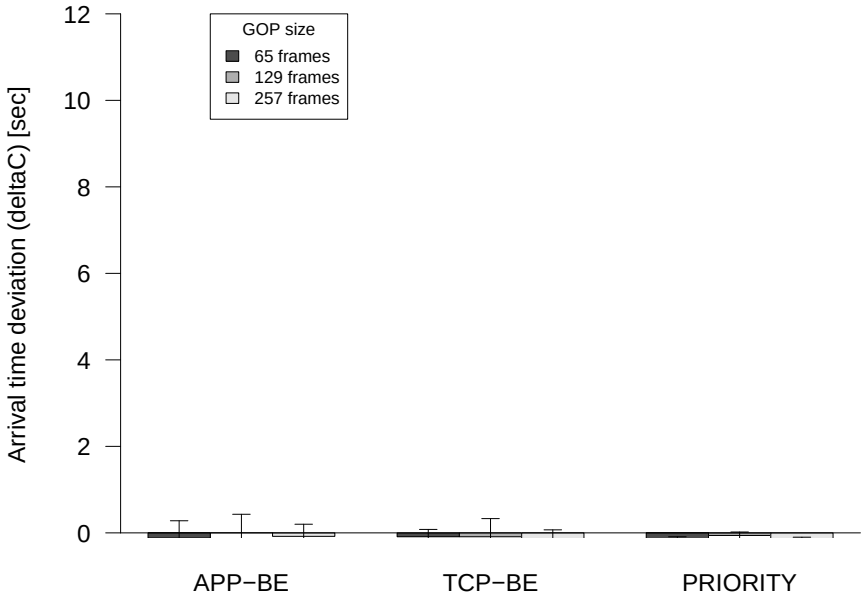



(b) *Arrival time deviation deltaC*

Figure 5.8: Overprovisioned network: $BW = 4096$ *kbps*. PSNR and arrival time deviation on the client (*deltaC*) for the different algorithms and GOP sizes at 50 ms RTT.

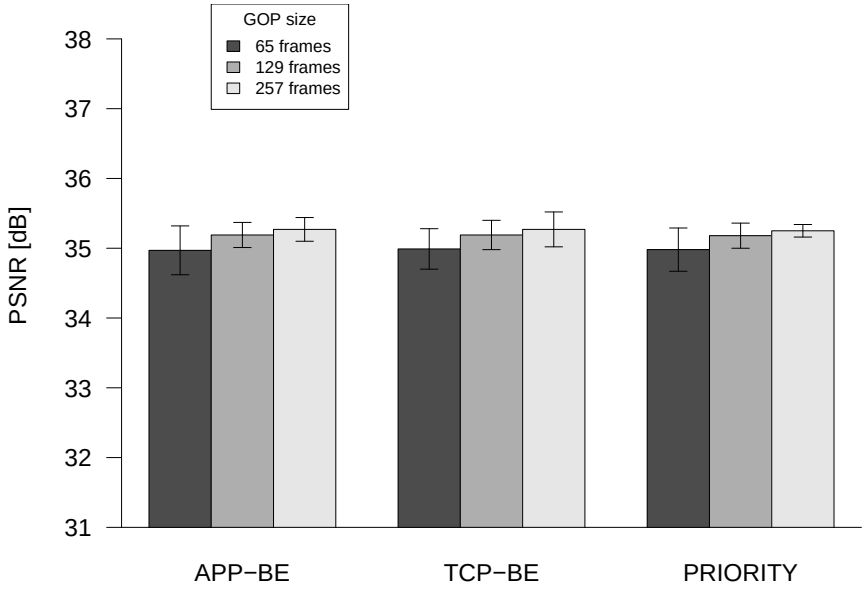



(a) *PSNR*

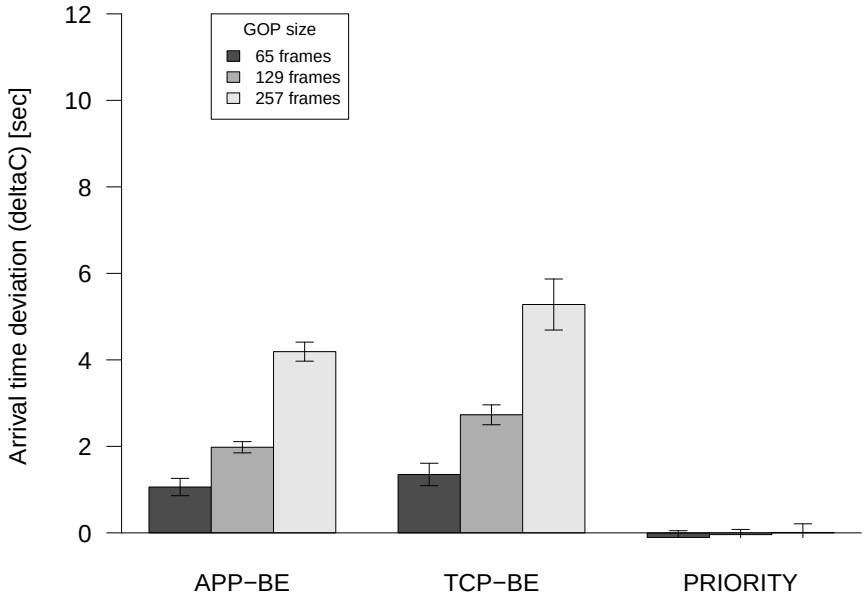



(b) *Arrival time deviation deltaC*

Figure 5.9: 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.

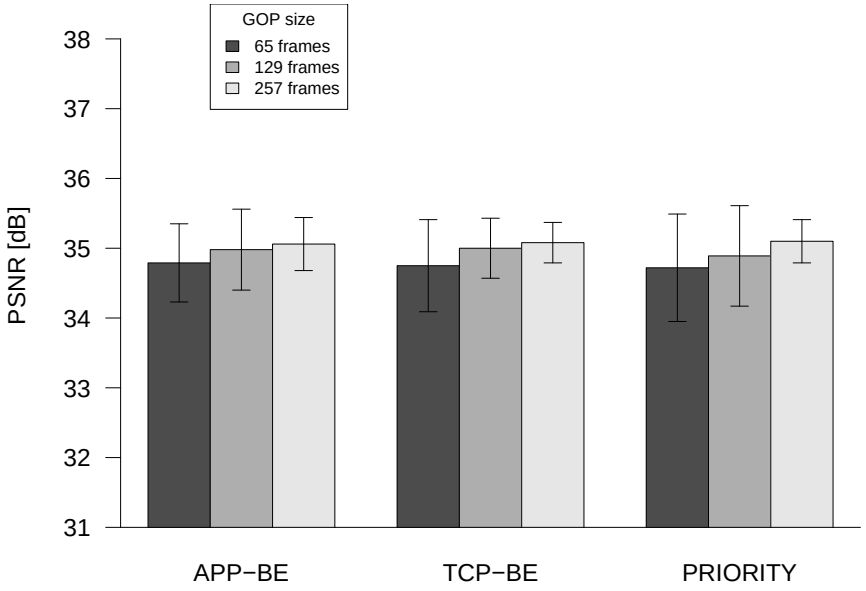



(a) *PSNR*

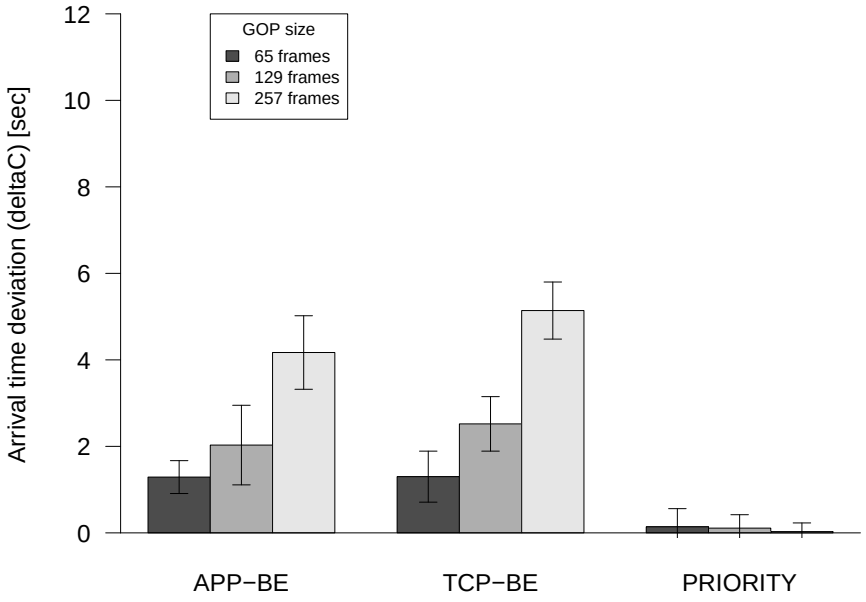



(b) *Arrival time deviation deltaC*

Figure 5.10: Underprovisioned network: $BW = 1536\ kbps$. PSNR and arrival time deviation on the client ($deltaC$) for the different algorithms and GOP sizes at 50 ms RTT.




(a) *PSNR*



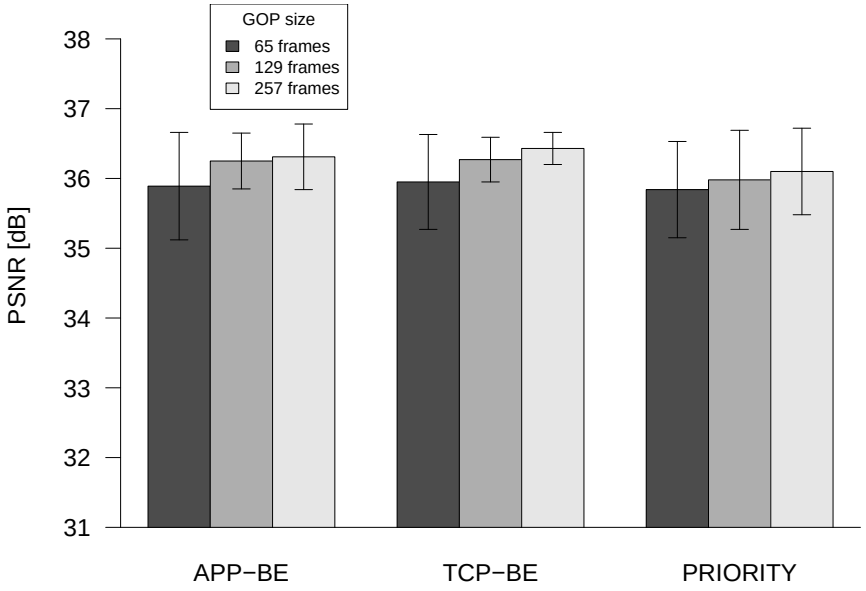

(b) *Arrival time deviation deltaC*

Figure 5.11: 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.

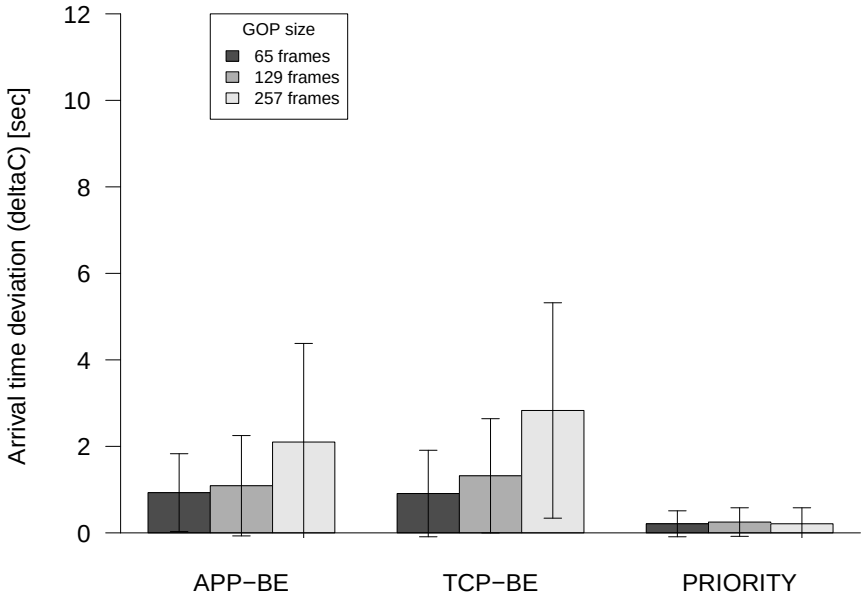
bandwidth (*underprovisioned network*) is shown in Figures 5.11 and 5.10. The average PSNR values at RTTs of 50 ms and 200 ms (see Figures 5.10(a) and 5.11(a)) are approx. 35 dB, which corresponds to a video bit rate of $\approx$ 1400 *kbps* (see Figure 5.7). This indicates that all streaming approaches can adjust their stream-out rate to the available bandwidth. But it can also be noticed that a higher RTT leads to a decreased average quality (PSNR) and increases the fluctuations of the quality. Larger GOP sizes tend to stabilize the PSNR values, because one of TCP's characteristics is that TCP features a stable throughput only in the long term [69]. When transmitting GOPs within small time windows, also the dynamic behavior of TCP will be noticeable, leading to high standard deviation values of the PSNR.

The results of APP-BE and TCP-BE in Figures 5.10(b) and 5.11(b) show that the arrival time deviation for GOP sizes 65, 129, 257 is limited to approx. 1, 2 and 4 seconds, respectively. This corresponds to about the half of the GOP's play-out duration $d^{gop}/2$ (see Section 5.3.3). The reason for that is the buffer control function, which starts reducing the stream-out rate if *delta* on the server exceeds half of the GOP's play-out duration (see Section 5.2.1 and Figure 5.3). The results indicate that the client buffer estimation on the server works well and can limit the buffer usage on the client to a predefined value (currently $d^{gop}/2$). In contrast to that, priority streaming does not rely on bandwidth estimation, so a timely delivery can be achieved independent of the GOP size. Like for the average PSNR, higher RTTs also lead to an increased variation of the arrival time deviation.

The influence of congestion on the adaptive streaming approaches is shown for an RTT of 50 ms in Figures 5.12, 5.14 and 5.16 and for an RTT of 200 ms in Figures 5.13, 5.15 and 5.17. It can be noticed that the achievable throughput and, consequently, the video quality decreases the more congestion on the network link occurs. The PSNR values in Figures 5.12(a), 5.14(a) and 5.16(a) are approx. 36, 34.5 and 33.5 dB, respectively. Like in the underprovisioned network scenario, the values of the PSNR are a little bit lower for higher RTTs (see Figures 5.13(a), 5.15(a) and 5.17(a)). In addition, the quality variation of the received video increases (standard deviation) with the congestion level, because of the increasing competition on the network link. The PSNR values correspond to bit rates of 1800, 1250, and 1000 kbps, respectively (see Figure 5.7). Thus, in all congestion scenarios

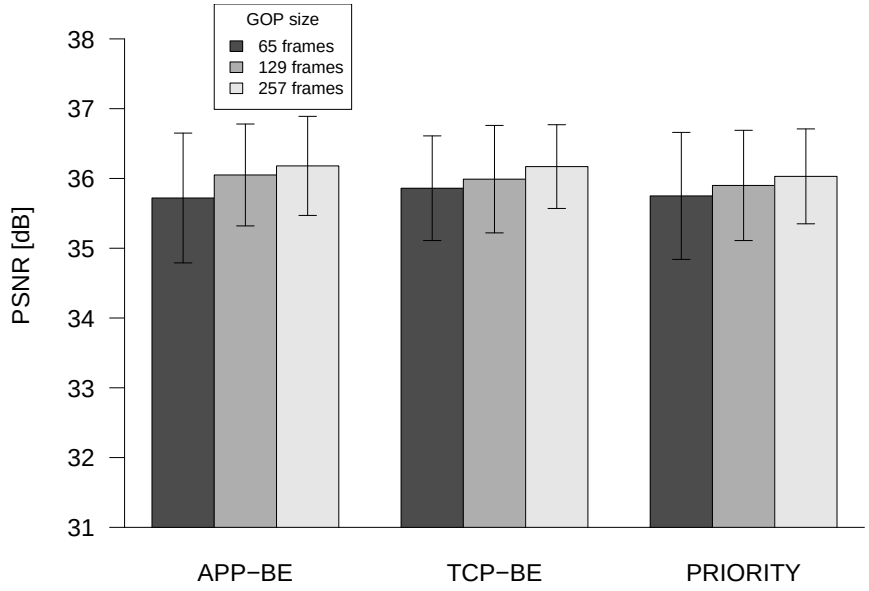



(a) *PSNR*

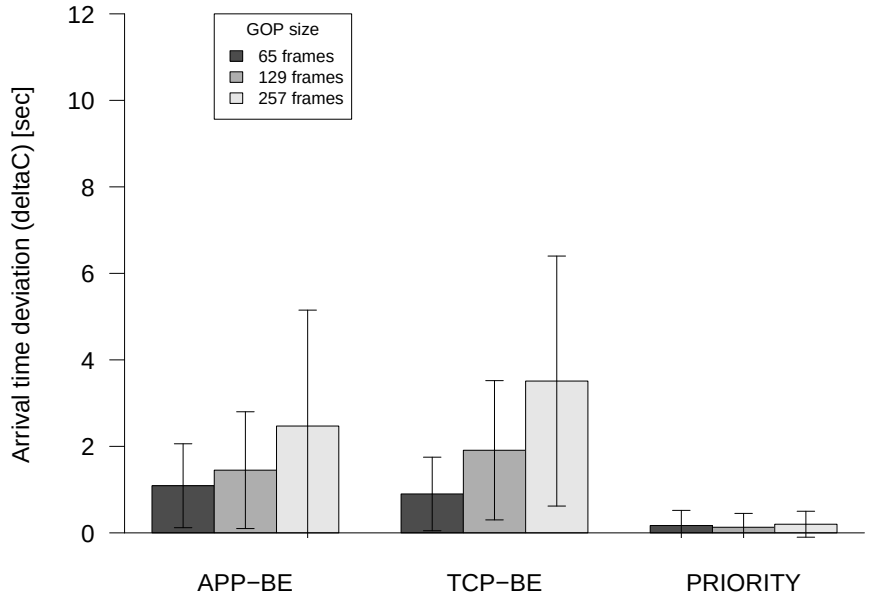



(b) *Arrival time deviation deltaC*

Figure 5.12: 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 50 ms RTT.

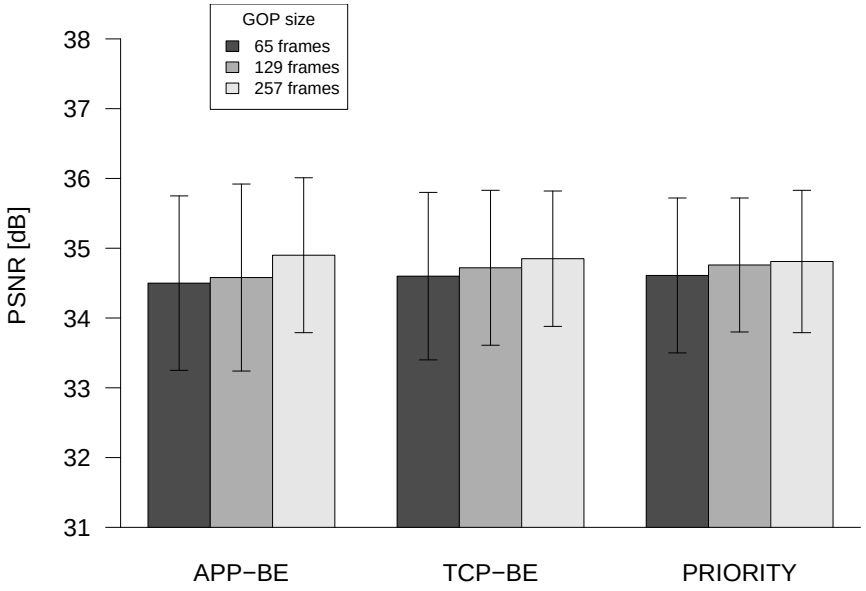



(a) *PSNR*

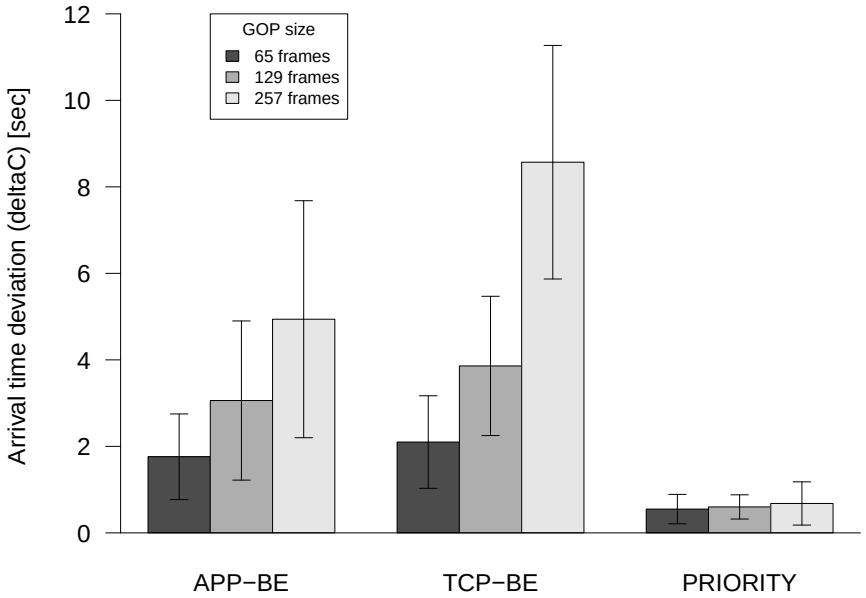



(b) *Arrival time deviation deltaC*

Figure 5.13: 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.

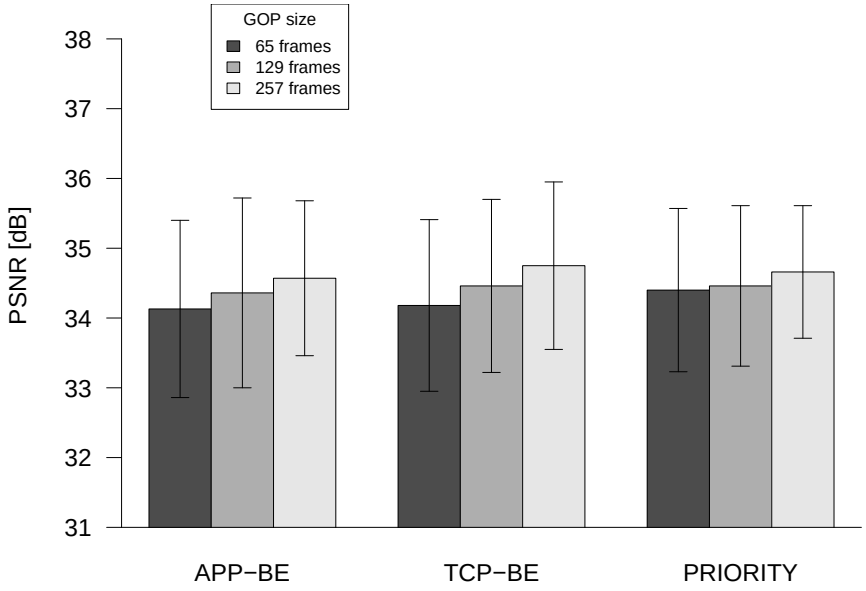



(a) *PSNR*

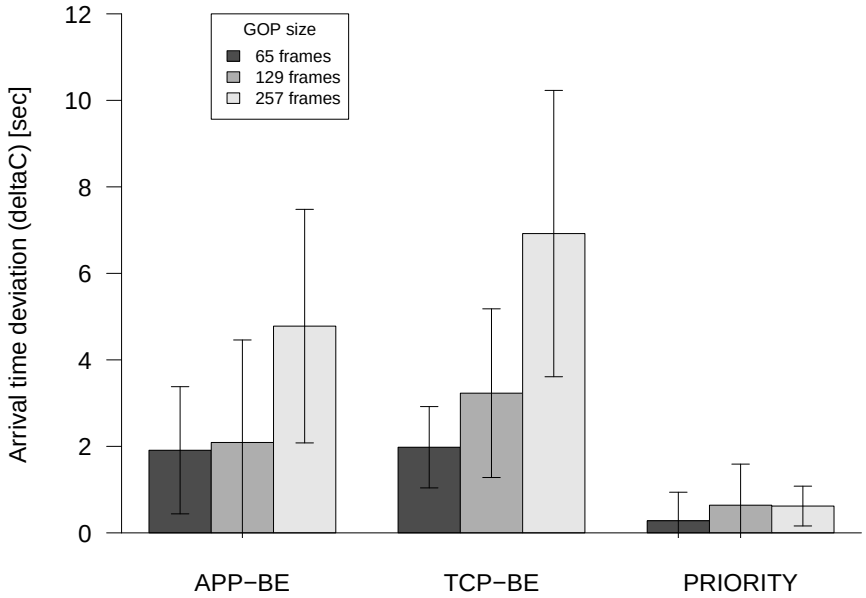



(b) *Arrival time deviation deltaC*

Figure 5.14: 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 50 ms RTT.

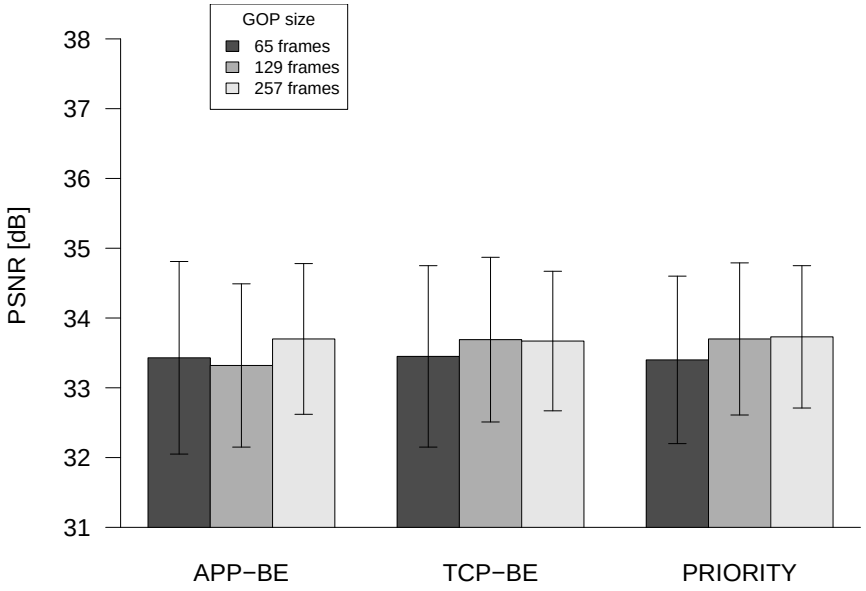



(a) *PSNR*

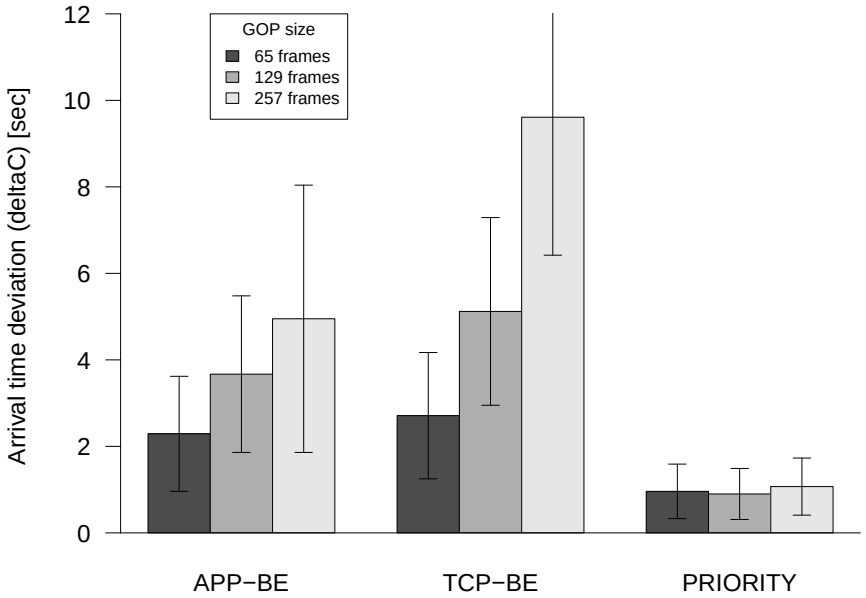



(b) *Arrival time deviation deltaC*

Figure 5.15: 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.

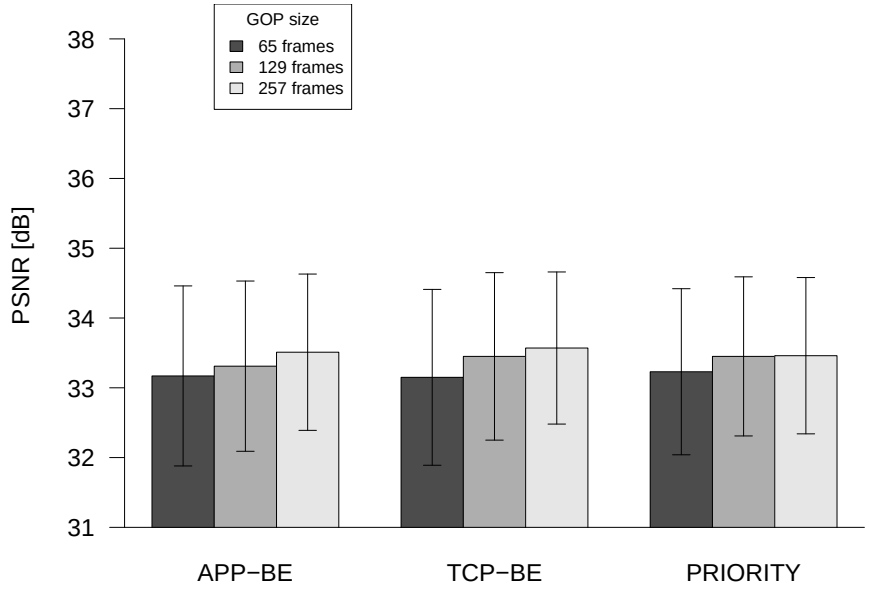



(a) *PSNR*

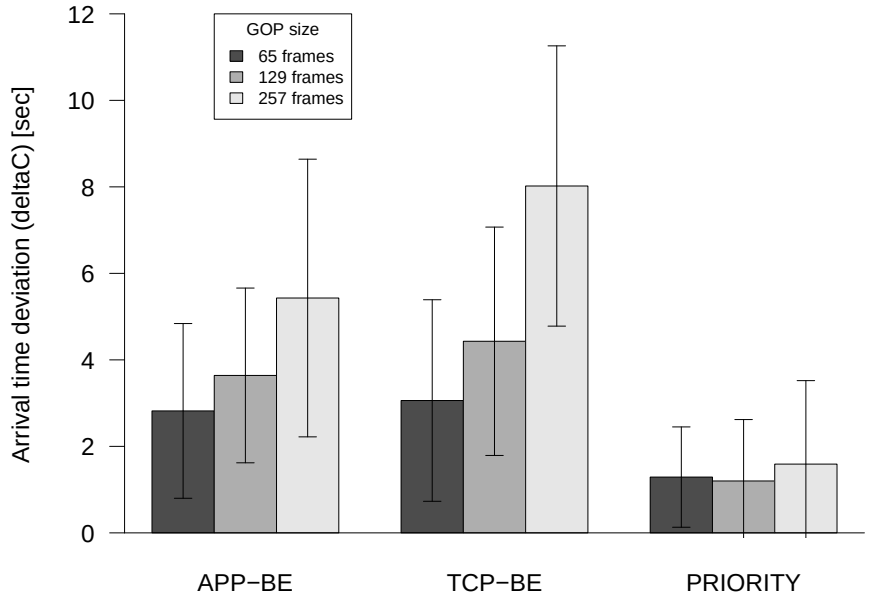



(b) *Arrival time deviation deltaC*

Figure 5.16: 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 50 ms RTT.




(a) *PSNR*




(b) *Arrival time deviation deltaC*

Figure 5.17: 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.

only the fair share of the available link bandwidth is used by the approaches. While all approaches are able to adapt to the available/fair bandwidth, it can be observed that priority streaming can make use of larger GOPs to stabilize the overall quality of the video.

The performance in terms of video quality is similar to the results in the underprovisioned network scenario. In contrast to this, the behavior of the arrival time deviation (*deltaC*) is quite different. Because of the congestion on the network link, the achievable throughput varies significantly, leading to large errors in the bandwidth estimation and therefore to high standard deviations of *deltaC*. For APP-BE and TCP-BE, the variation of *deltaC* increases with the amount of congestion and the GOP size, while priority streaming can achieve a stable performance for all GOP sizes. This is mainly because for larger GOP sizes a bad bandwidth estimate leads to a larger deviation of the delivery duration, which directly influences *deltaC*. Nevertheless, the arrival time deviation for APP-BE and TCP-BE does not drift away because of the buffer control at the server.

The evaluation results indicate that of the three GOP sizes used, GOP size 65 is likely to be the best choice for APP-BE and TCP-BE in this configuration. This is because errors in the bandwidth estimate have a greater impact on the performance when larger GOP sizes are used. On the other hand, small GOP sizes usually lead to a higher variation of the video quality (PSNR), because the quality changes happen more frequently.

In contrast to these approaches, priority streaming is much more robust regarding changes in available bandwidth. In addition, priority streaming is able to enhance the performance with larger GOP sizes, so a GOP size of 257 frames is considered to be the best choice in this setup. Figures 5.18, 5.19, and 5.20 show the average PSNR of the adaptive streaming systems with respect to the RTT. In absence of 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 the tough competition on the network link.

## 5.5 Limitations of TCP-based Adaptive Video Streaming

The evaluation of adaptive TCP-based streaming in an Internet-like setting showed that, despite the simplicity of the adaptive approaches, they can effectively cope with bandwidth limitations and congestion, even at high RTTs. The use of scalable video allows the bit rate

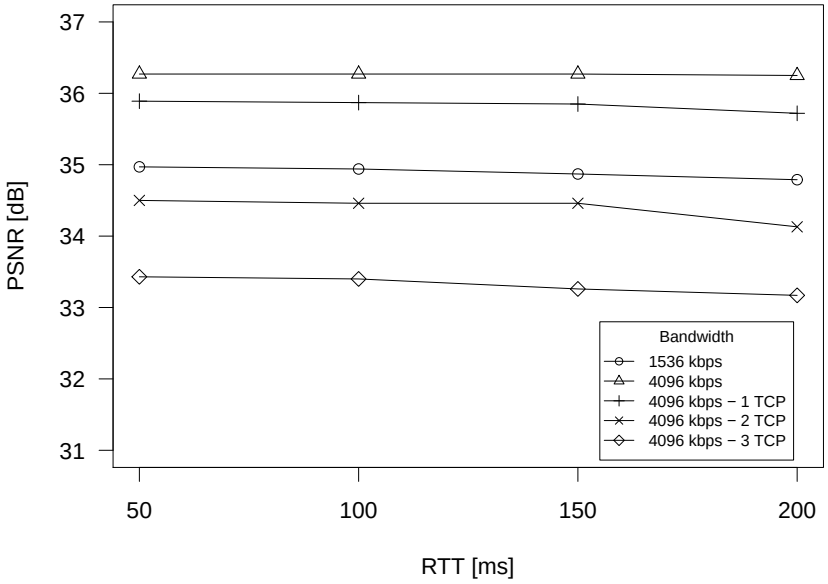



Figure 5.18: Average PSNR for APP-BE with GOP size 65 with respect to the RTT.

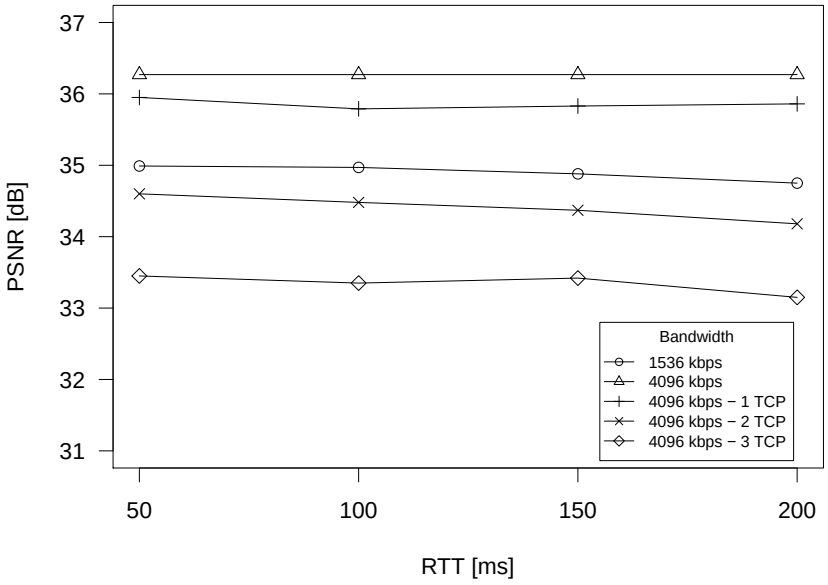



Figure 5.19: Average PSNR for TCP-BE with GOP size 65 with respect to the RTT.

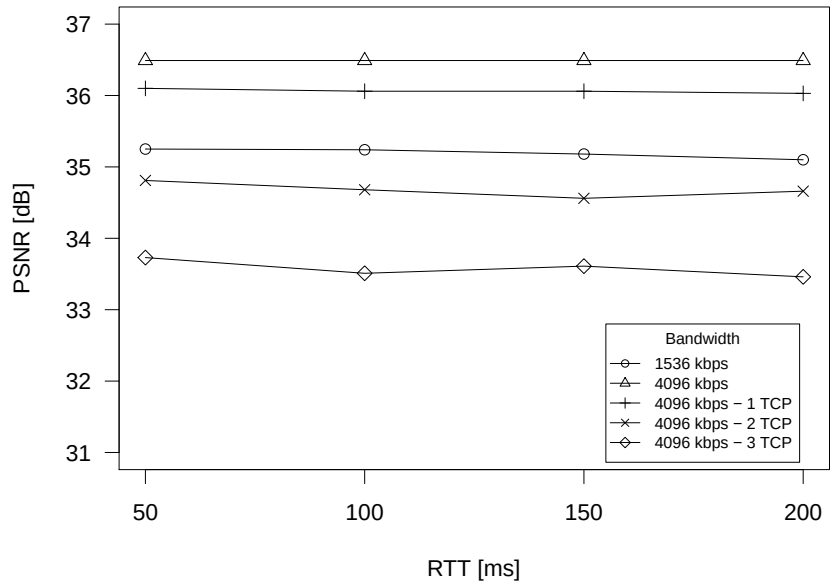



Figure 5.20: Average PSNR for Priority Streaming with GOP size 257 with respect to the RTT.

and quality of the delivered video to be adapted in a fine-grained manner. Because TCP only features a stable throughput in the long term, the adaptive streaming approaches utilize rather large GOPs to minimize fluctuations in the video quality. However, the approaches using bandwidth estimation suffer from large GOP sizes, since inaccurate bandwidth estimates may impact the quality of the video and increase the variation of the arrival time deviation. Thus, determining the optimal size of the client buffers may become difficult. Priority streaming reorders and transfers the video data according to their importance (e.g., rate-distortion). Large GOPs help to improve the performance, by averaging TCP's throughput and, consequently, the video quality over a longer period of time. For priority streaming, the variance of the arrival time deviation is more predictable, because it does not rely on bandwidth estimation. This also simplifies the dimensioning of the client buffers.

TCP-based adaptive streaming performs robustly in terms of video quality and timely delivery, both on under-provisioned links and in case of congestion. It is suited for Internet streaming, because it adapts to the available bandwidth and is congestion-aware. The TCP

connection and its specific characteristics, like the AIMD algorithm, are the main limiting factors of the streaming performance. In the presence of packet loss, TCP's performance deteriorates rapidly, rendering video streaming impossible (see Section 6.1).

Additionally, using TCP in adaptive streaming leads to several scalability issues. While client state estimation on the server helps to increase the responsiveness of the adaptive control loop (all logic is placed on the server), the server has also to cope with the client states, adapt the video according to the network conditions and manage stream-out of the video data. This is considered to be computationally expensive compared to non-adaptive stream-out of the video data and usually reduces the number of clients a server may serve, leading to poor scalability. In the network, TCP's end-to-end transmission paradigm and the adaptation to the specific client requirements prevent caching or proxying from being deployed.

To overcome these restrictions, HTTP-based video streaming was introduced, which uses CDNs and HTTP caching to enhance the scalability in terms of network efficiency. Because existing infrastructure can be reused, low deployment costs can be achieved. The client-driven paradigm of HTTP also allows for new approaches to adaptive streaming. One of these will be presented in the following chapter.

CHAPTER

# 6 Parallel HTTP-based Request-Response Streams

Congestion awareness is considered as most important in the Internet. For that reason, TCP/HTTP-based adaptive video streaming gained a lot of momentum. In the envisioned use case, Internet video streaming (see Section 1.1 and Figure 1.1), it is assumed that the access network forms the bottleneck link. In addition, network packets may be lost due to transmission errors in wireless networks.

In the following, TCP-based video streaming will be analyzed in detail for this use case. To complement the shortcomings of TCP-based video streaming, in this thesis, multiple parallel HTTP-based request-response streams [52, 53] will be introduced and investigated for the use in video streaming. An analytical model for the achievable throughput of a request-response streaming system will give more insights into the behavior of the request-response streams. Additionally, the system parameters of a request-response streaming system will be investigated regarding their influence on the system performance under diverse network conditions.

## 6.1 TCP-based Video Streaming in Error-Prone Networks

TCP and especially 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. While there are dedicated protocols for video streaming in IPTV networks (like RTP/UDP [85]), TCP-based video streaming is able to replace more and more of the established streaming protocols in the Internet. For a long time, TCP was considered a bad match for video transmission [49]. The reason for that is TCP's

dependency on the network conditions, because TCP's reliability and adaptive behavior are based on acknowledgments and retransmissions. In general, dynamically changing RTTs or packet loss rates lead to a degradation of TCP's performance [57].

The throughput rate of TCP depends on the maximum segment size ($MSS$) and the round trip time ($RTT$). Considering 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 [60]:

$$r_{tcp} = \frac{MSS}{\sqrt{p}} \cdot \frac{1}{RTT} \tag{6.1}$$

From Equation 6.1 it becomes evident that for a given RTT, the maximum throughput of TCP is only limited by the packet loss rate $p$. While the equation only estimates the long-term mean value of the TCP throughput, the additive-increase/multiplicative-decrease (AIMD) algorithm of TCP's congestion control introduces also a variation of the throughput. Thus, TCP has been considered unsuitable for video streaming for many years. The performance of TCP for streaming constant-bit-rate content was investigated in [104]. As a result, it was stated that the achievable TCP throughput should be at least twice the video bit rate in order to avoid jerky playback, leading to a very bad utilization of the network link. Even if it is possible to provide this kind of overprovisioning at the start of the play-out, adapting the video content may be required to be able to cope with changing network conditions during the play-out. The results presented in Chapter 5 showed that TCP-based adaptive video streaming can adapt to the available bandwidth quite well and is able to handle bandwidth fluctuations without introducing too much start-up delay (buffering). The disadvantages of TCP-based adaptive streaming are mainly inherited from the single TCP connection used for the transmission of the video data. The high computational costs to serve the video to a client and the high network load (each client receives a unique video stream even if the same content is streamed to multiple clients), makes TCP-based adaptive video streaming an undesirable choice for large scale deployments. In addition, packet loss or transmission stalls of the TCP connection may lead to disruptions in the video playback.

Adaptive HTTP video streaming (see also Section 4.5.2) tries to overcome these problems, by exploiting HTTP's request-response paradigm. The idea is to download a lot of small so-called video fragments instead of the whole video at once, where each fragment comprises several seconds of video. The video fragments are downloaded consecutively by

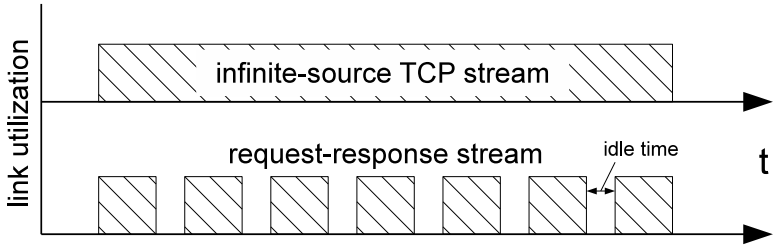



Figure 6.1: Utilization of a network link for an infinite-source TCP stream and a request-response stream.

the client and the video can be decoded and displayed to the user. Like in stream switching (see Section 4.2), the content is available in several qualities on the server. After each fragment download, the quality of the next fragment to fetch is determined based on the current network conditions and the client buffer level. Because HTTP streaming is usually client-driven and based on video fragments, a good scalability in terms of clients served and network usage can be achieved.

In the following, a single HTTP connection transporting video fragments is called an *HTTP-based request-response stream*. While the streaming systems introduced in Section 4.5.2 are mainly based on a single request-response stream, this work focuses on the behavior of *multiple concurrent HTTP request-response streams*. In the next section, the HTTP-based request-response streams will be described in detail.

## 6.2 Modelling HTTP-based Request-Response Streams

TCP-based video streaming usually uses server-side mechanisms to deliver the media data. The server continuously streams audio and video to the client using a long-lived TCP connection. HTTP streaming behaves differently from classical TCP streaming, because it is based on the request-response (rr) paradigm. *HTTP-based request-response streams* have to request the media data to initiate their transmission. In addition, request-response streams use responses that are in general small chunks of media data. When using very large junk sizes, request-response streams become similar to long-lived TCP connections, which is usually unfavourable because also the advantages diminish. In any case, small chunk sizes lead to short responses, which may experience unfairness in congested networks [57]. The reason for this behavior is that between the end of the response and the next

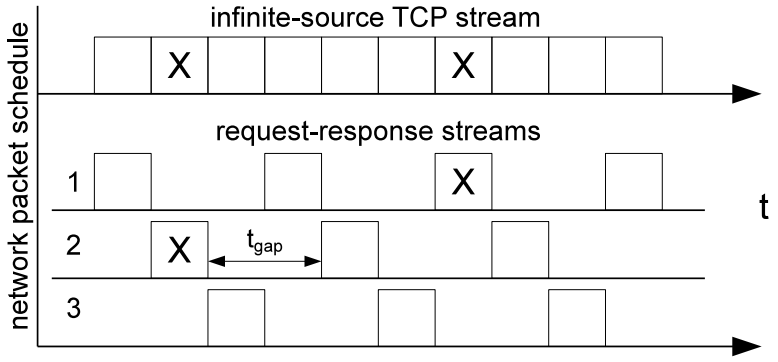



Figure 6.2: Network packet schedule for an infinite-source TCP stream and three request-response streams under packet loss (X).

request there is usually a short idle time where the network link is not used by the request-response stream (see Figure 6.1). This idle time comprises the processing time on the client to create the new response, the time needed to transmit the request to the server (network delay) and the processing time on the server to create the request. In this idle time, other concurrent TCP connections may increase their congestion windows and request-response streams may struggle to reclaim their fair share of the network's bandwidth.

Because TCP connections usually share the available bandwidth in a fair manner [57], aggregating multiple TCP streams for a single use is regarded unfair to concurrent single connections. The fact that idle times in the TCP transmission change the TCP friendliness, led to the idea to control the TCP-friendliness of request-response streams by introducing temporal gaps between requests (inter-request gap $t_{gap}$). These inter-request gaps increase the idle time between the requests of the request-response streams [52, 53]. When considering rather small chunks, which may fit into a single congestion window, the inter-request gap can also be interpreted as an increased RTT. In general, TCP features no throughput fairness between connections with different RTTs [57]. For that reason, it should be 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, because a lost packet only reduces the throughput of a single request-response stream [99, 52, 53]. While the probability to lose a packet is the same for all request-response streams, the number of packets a single stream carries is reduced. As shown in Figure 6.2, the packets are distributed over multiple

streams. For that reason, the time between packet loss events for a single request-response stream is higher compared to a single TCP connection, resulting in a better performance of the aggregated request-request streams.

Because the transmission scheme of the request-response streams is based on video segments and multiple connections are used to fetch the video, the proposed transmission scheme is related to P2P [21] and multi-source streaming [82]. In contrast to these systems, an HTTP-based request-response streaming system is fully client-driven, which enables fast recovery in case of connection aborts and features session roaming. The computational complexity of the request-response streams is lower than in P2P systems, because the number of parallel connections is rather small. For that reason, also the behavior of the request-response streams is easier to estimate (compared to P2P systems), especially if all streams connect to the same server. Finally, request-response streams are able to feature *TCP-friendliness*, which is seen crucial for Internet deployments. Providing TCP-friendliness is still problematic in P2P and multi-source streaming environments.

### 6.2.1 Simple Model

To estimate the achievable throughput of the request-response streams, a simple model describing the upper bound of the throughput was created. Assuming that a chunk of size $l_{ch}$ is transferred within a single RTT and $n_c$ concurrent request-response streams are used for the transmission of the data, the upper bound for the throughput without packet loss $r_{rrsimple}$ can be calculated as follows [52]:

$$r_{rrsimple} = n_c \left( \frac{l_{ch}}{RTT + t_{gap}} \right) \tag{6.2}$$

To model the influence of packet loss, it is important to know how many packets are in flight. For the data transmission, $n_c$ request-response streams are used and each data block sent is fragmented into $n_{ch}$ packets of size MSS ($l_{ch} \approx n_{ch} * MSS$). As a result, a total of $n_p = n_c * n_{ch}$ packets are transmitted per RTT. Assuming the same packet loss pattern as described in Section 6.1, the *maximum number of lost packets* $n_l$ is defined as:

$$n_l = \left\lceil \frac{n_p}{1 + 1/p} \right\rceil \tag{6.3}$$

To keep the complexity of the model low, the packet loss is assumed to be distributed equally over all request-response streams. In addition, the model relies on the assumption

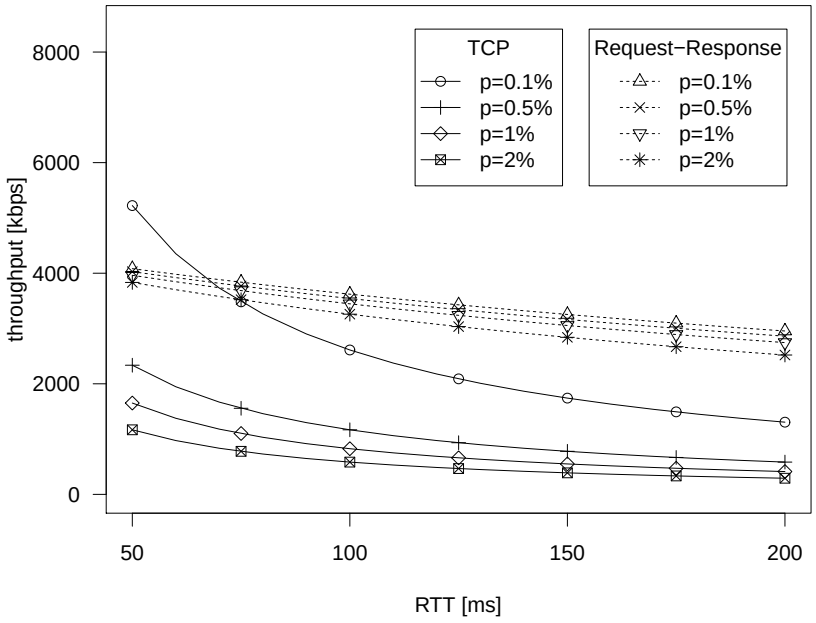



Figure 6.3: Theoretical upper bounds for throughput of a single TCP connection (TCP) $r_{max}$ and request-response streams $r_{rrmax}$ ($MSS = 1460\ bytes$, $l_{ch} = 20480\ bytes$, $n_c = 10$, $t_{gap} = 350\ ms$) under packet loss $p$. [52]

that each TCP connection does not experience more than one lost packet per transported data block (which is transmitted within one RTT). As a consequence, the number of lost packets per RTT should not exceed the number of TCP connections ($n_l \leq n_c$). Thus, the *upper bound for the throughput under packet loss* $r_{rrmax}$ of the request-response streams can be estimated as [52]:

$$r_{rrmax} = (n_c - n_l)\left(\frac{l_{ch}}{RTT + t_{gap}}\right) + n_l\left(\frac{l_{ch}}{2 * RTT + t_{gap}}\right) \tag{6.4}$$

From Equation 6.4 it becomes evident that the throughput can be controlled by means of $n_c$, $l_{ch}$ and $t_{gap}$. While the number of concurrent streams $n_c$ and the chunk size $l_{ch}$ are responsible for enhancing the throughput, the inter-request gap $t_{gap}$ can be used to tune the TCP-friendliness. The goal is to stabilize and enhance the overall throughput and to increase the robustness to changing network conditions compared to a single TCP connection. On the other hand, its also important to be fair to other concurrent TCP connections in case of competition on the network link.

This simple model was evaluated in [52] with a network bandwidth (BW) of 8192 kbps,

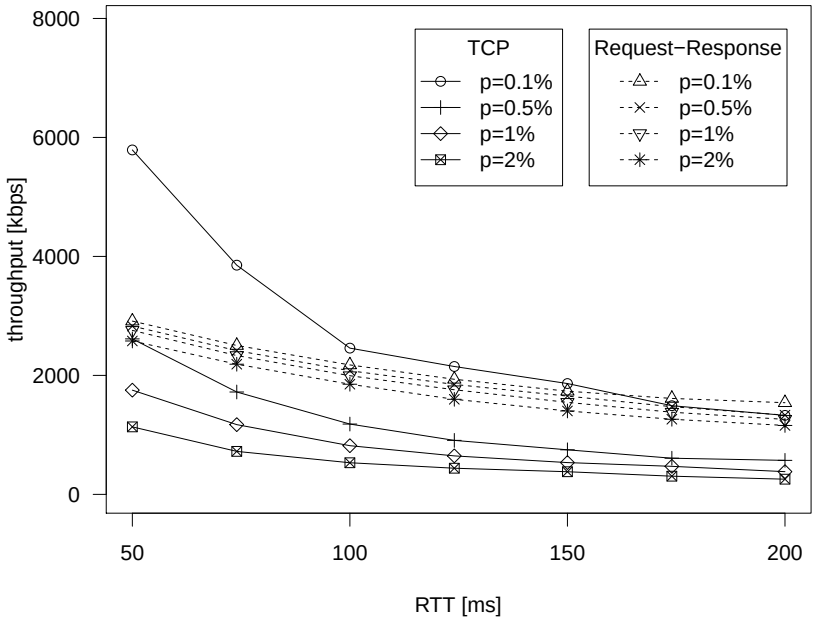



Figure 6.4: Measured throughput of a single TCP connection (TCP) and request-response streams ($BW = 8192$ $kbps$, $MSS = 1460$ $bytes$, $l_{ch} = 20480$ $bytes$, $n_c = 10$, $t_{gap} = 350$ $ms$) under packet loss $p$. [52]

which should not limit the maximum throughput. In Figure 6.4, the average throughput for a single TCP connection and the request-response streams are shown. For small RTTs, the request-response streams and the single TCP connection are able to cope with packet loss via retransmissions. With increasing RTT and packet loss, the throughput of the single TCP approach deteriorates rapidly, while the impact on the request-response streams is limited. The estimated throughput of the simple model (see Figure 6.3) shows a good correlation to the measured results in Figure 6.4. Although lower, the measured throughput is similarly shaped and behaves according to the assumptions.

In this simplified model, the upper bound for the throughput of the request-response streams is defined very optimistically, because the model assumes that each data block is transmitted within a single RTT and no processing on the server is done. When considering real world conditions (e.g., a limited network bandwidth or congestion), multiple RTTs may be needed for the transmission, depending on the used configuration ($l_{ch}$, $n_c$ and $t_{gap}$). Even if each data block can be transmitted in a single RTT, the processing of the request on the server may increase the effective RTT.

In the envisioned use case, network congestion is very likely to occur. For that reason, an extended model will be presented in the following section, which is aware of the network's characteristics.

### 6.2.2 Enhanced Model utilizing Network Characteristics

The assumption of the simple model, i.e., that a chunk can be transferred within a single RTT, may not be valid for large chunk sizes. For example, transferring a large chunk of 10 Mbytes over a network with a bandwidth of 1 Mbps takes at least 80 seconds ($t = l_{ch}/BW$). But RTTs larger than 500 ms are not realistic for the envisioned use case. Hence, such a chunk cannot be transferred within one RTT. To be able to estimate the amount of RTTs needed for the transmission of a chunk, the model was extended to take also the network's characteristics into account.

This extra knowledge can be used to explore the limitation of the request-response streams under certain network conditions [53]. For that reason, it is assumed that the bottleneck bandwidth $BW$ and the maximum queueing delay $t_q$ of the bottleneck router are known. By using $BW$ and $t_q$, the maximum bottleneck queue size $l_q$ can be calculated as follows, $l_q = BW * t_q$. Because the single request-response streams are competing on the network links against each other, it is assumed that the bottleneck router queue is shared between all $n_c$ concurrent TCP connections. Therefore, each request-response stream occupies $l_{rr} = l_q/n_c$ bytes of the bottleneck router queue. Assuming that each request-response stream can transfer at most $l_{rr}$ bytes per RTT, the number of round trips needed to transmit a chunk is defined as $n_{rt} = l_{ch}/l_{rr}$.

Each request-response stream uses its own TCP connection for the transmission of the data. Because the TCP connections try to maximize the throughput, it is assumed that the router queue will be fully utilized. Looking at a single request-response stream, the AIMD algorithm of TCP leads to a saw tooth shaped queue utilization (see Figure 6.5). When the network link is fully utilized, the router queue level increases constantly to the maximum. If no more packets can be stored in the queue, a packet has to be dropped. TCP reacts to this packet loss by reducing its congestion window and sends no new packets till the number of non-acknowledged packets is less than the new congestion window size. Because no new packets are sent to the router, the router queue is drained with the serialization rate of the

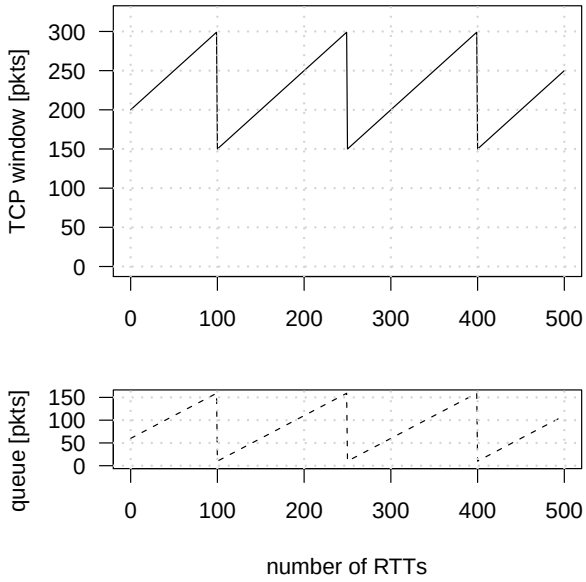



Figure 6.5: TCP window size and router queue utilization for a TCP flow through a router with a queue size equal to the bandwidth-delay product ($l_q = BW * MAXRTT$) [8].

network link. As a result, the queue utilization follows a saw tooth shape and the queuing delay of a network packet will be on average $t_q/2$. Because multiple TCP streams tend to self-synchronize [8], it is assumed that the average queuing delay is also valid for multiple TCP streams.

Of course, the average queuing delay will only be effective if all TCP streams fully utilize the network bandwidth. But request-response streams cannot fully utilize the network queue, if the chunk size is too small or the number of concurrent streams is too low. In this case, a data block can be transmitted in less than an RTT ($n_{rt} < 1$), leading to a under-utilized queue and reduced queuing delay. For that reason, the estimated queuing delay is reduced, if $n_{rt}$ is below one. The *average queuing delay* of a request-response stream $t_{qav}$ is estimated as [53]:

$$t_{qav} = min(n_{rt}, 1) \cdot t_q/2 \tag{6.5}$$

Using this knowledge the *estimated transfer duration of one chunk* $t_{ch}$ can be defined as:

$$t_{ch} = \lceil n_{rt} \rceil (RTT + t_{qav}) \tag{6.6}$$

Following the idea of Equation 6.4, the *average achieved throughput without packet loss* $r_{rr}$

of the request-response streams can be defined as:

$$r_{rr} = n_c \left( \frac{l_{ch}}{t_{ch} + t_{gap}} \right) \tag{6.7}$$

Equation 6.7 shows that, like in the initial simple model (Equation 6.4), it is possible to tune the throughput by means of $n_c$, $l_{ch}$, and $t_{gap}$. The main difference is now that this model takes the queue size of the bottleneck router and the bottleneck bandwidth into account, resulting in a more realistic model for the transmission time of a chunk (the simple model took only the RTT into account).

A direct result of Equation 6.1 is that the amount of data which TCP can transport within a single RTT is limited by the packet loss rate. Because request-response streams are based on TCP, $l_{rrloss}$ is defined as the average amount of data which can be transported within one RTT. The amount of data is now either restricted by the bottleneck router queue size or the limitations of TCP in case of packet loss. Note that the same packet loss pattern as in the TCP throughput estimation (see Equation 6.1) is used.

$$l_{rrloss} = min(l_{rr}, \frac{MSS}{\sqrt{p}}) \tag{6.8}$$

The number of round trips needed to successfully deliver a chunk is not only dependent on the bottleneck router's bandwidth and queue size, it is also affected by the packet loss. As a result, the number of transmissions is redefined as $n_{rtloss} = l_{ch}/l_{rrloss}$. Consequently, also the queuing delay is affected by the packet loss, because the packet loss restricts TCP's window size and therefore the router queue utilization. In general, the queuing delay is directly related to queue utilization. For that reason, the proposed model defines a simple relation between the utilized queue size and the queuing delay. As a consequence, the queuing delay begins to decrease, if the queue size in the router sinks below a certain level. The *average queuing delay under packet loss* $t_{qavloss}$ is estimated as follows [53]:

$$t_{qavloss} = \begin{cases} \frac{l_{rrloss}}{l_{rr}} \cdot t_q/2 & \text{if } l_{rrloss} < \frac{l_{rr}}{2} \\ min(n_{rtloss}, 1) \cdot t_q/2 & \text{otherwise} \end{cases} \tag{6.9}$$

The estimated transfer duration of one chunk under packet loss $t_{chloss}$ can be calculated as follows:

$$t_{chloss} = \lceil n_{rtloss} \rceil (RTT + t_{qavloss}) \tag{6.10}$$

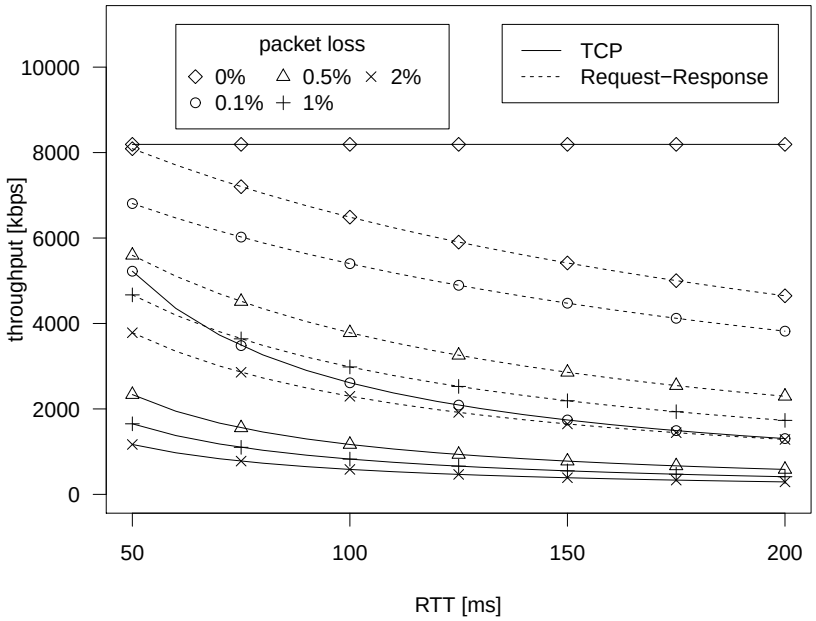



Figure 6.6: Modeled throughput performance of a single TCP connection (TCP) $r_{tcp}$ and the request-response streams $r_{rr}$ ($MSS = 1460\,bytes$, $l_{ch} = 160\,kB$, $n_c = 5$, $t_{gap} = 210\,ms$) at a fixed bottleneck bandwidth $BW = 8192\,kbps$ and packet loss rate $p$ [53].

Finally, the throughput under packet loss for the request-response streaming system $r_{rrloss}$ can be estimated as:

$$r_{rrloss} = n_c \left( \frac{l_{ch}}{t_{chloss} + t_{gap}} \right) \tag{6.11}$$

This enhanced model makes use of the throughput estimation for TCP (see Section 6.1) and the knowledge about the bottleneck router to estimate the performance of the request-response streams. Packet loss and queuing delay on the bottleneck router are the main factors responsible for limiting the throughput performance. While the number of request-response streams and the chunk size can be used to increase the throughput, also the influence of the used system configuration on the TCP fairness and the computational effort needed to manage the multiple streams have to be considered.

Figure 6.6 shows the upper bounds for a single TCP connection and the request-response streams according to Equations 6.1 and 6.11, respectively. Similar to the simple model (see Figure 6.4), the decline of the throughput of the request-response streams with increasing RTT and packet loss is significantly reduced, while the performance of the single TCP connection highly depends on the packet loss and the RTT. The major difference of the

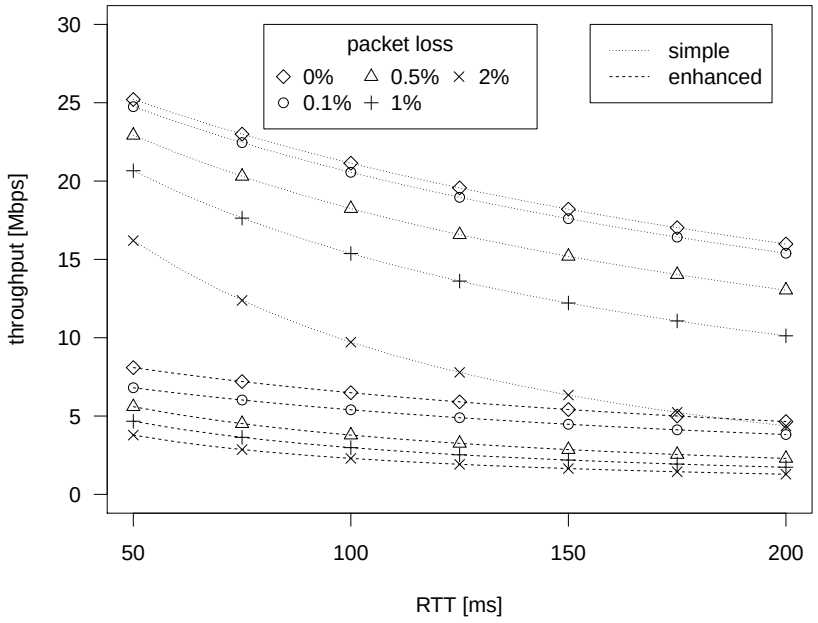



Figure 6.7: Throughput estimate of the simple and the enhanced model for the request-response streams $r_{rr}$ ($MSS = 1460\ bytes$, $l_{ch} = 160\ kB$, $n_c = 5$, $t_{gap} = 210\ ms$) at a fixed bottleneck bandwidth $BW = 8192\ kbps$ and packet loss rate $p$.

simple and the enhanced model can be noticed in Figure 6.7. Because the enhanced model is aware of the network's characteristics, using more parallel streams than necessary will not enhance the estimated throughput. The maximum throughput is limited by the router queue size, which is split between the single streams. Although larger queues would allow for larger TCP windows of the request-response streams, they would also increase the queuing delay because of the fixed bandwidth. For that reason, the estimated throughput is limited to the maximum bandwidth, which also reflects the real-world behavior. On the other hand, the simple model is not aware of network congestion and estimates the throughput beyond the maximum network bandwidth.

In Chapter 7, the pros and cons of the request-response system will be discussed in detail and a comparison of the enhanced model with real measurements will be presented. In the next section, an HTTP-based request-response streaming system will be presented. It is based on priority streaming and H.264/SVC (see Section 4.3 and 5.2.3) and uses multiple request-response streams to transport the video data. This streaming system will also be used for the evaluation.

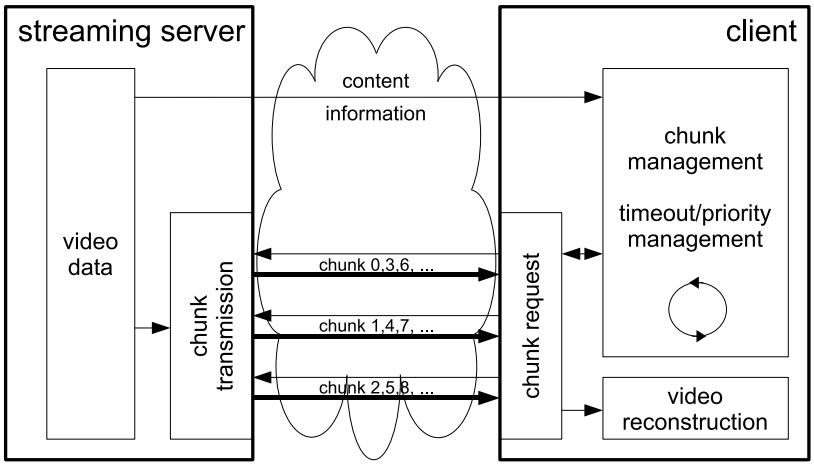



Figure 6.8: Request-response-based client-driven streaming system [53].

## 6.3 Adaptive Video Streaming with Parallel HTTP-based Request-Response Streams

The request-response streams' ability to avoid transmission stalls and therefore jerky playback makes them a good choice for video streaming. Parallel request-response streams can be used to stabilize the throughput and therefore reduce quality fluctuations, while still providing TCP-friendliness. Priority streaming can be used to improve the timeliness of delivery, because the priority-reordered data can be truncated any time (see Section 4.3). A combination of both will be used as a basis for the presented architecture.

The *HTTP-based request-response streaming system* is based on the client-driven adaptive streaming principle described in Section 4.4.2. The architecture of such a system is depicted in Figure 6.8. Since the system is based on HTTP, easy deployment, reuse of existing infrastructure (HTTP server, client, encryption, etc.) and application-layer multicast through HTTP proxies is enabled. In order to minimize the overhead of the TCP connection setup, HTTP's persistent connections (as defined in HTTP/1.1 [23]) are used for establishing the TCP connections.

As a prerequisite of priority streaming, the video is split into video fragments, comprising the same play-out duration $t_{frag}$. Each video fragment is rearranged according to the priority of its video syntax elements (see Section 4.3). The rearranged video fragment is

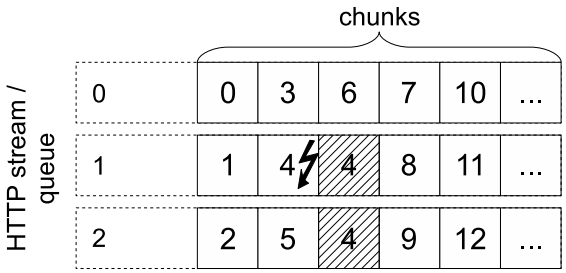



Figure 6.9: Three HTTP-based request-response streams/queues on the client with priority management [53].

called a *video segment*.

The streaming system can be configured using three different system parameters, namely the *chunk size* $l_{ch}$, the *number of concurrent streams* $n_c$ and the *inter-request gap* $t_{gap}$. Each 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, which manages $n_c$ HTTP-based request-response streams. The client schedules the downloads of the different chunks by using separate queues for each stream (see Figure 6.9). Each chunk is retrieved according to the order within the queue. In order to avoid self-congestion and to provide TCP-friendliness, each stream inserts an inter-request gap $t_{gap}$ between consecutive chunk downloads.

Priority streaming usually accounts for a fixed amount of time for the transmission of a video segment. Usually, the play-out duration of the video fragment is used as time limit (see also Section 4.3). In this time frame, as many video chunks as possible are transferred via the request-response streams. For that reason, the time used for downloading a segment is monitored. If the time limit is reached, the client stops downloading chunks of the current segment and switches to chunks of the next segment. All chunks belonging to the same segment are aggregated to a possibly truncated video segment and the video is reconstructed.

For priority streaming, in-order transmission is of most importance, because it heavily relies on the priority order within the video segments. However, transmitting video chunks over multiple request-response streams usually breaks the in-order transmission, if only

standard multiplexing techniques are applied. For that reason, the presented streaming system uses *timeout and priority management* to optimize the in-order throughput and to prevent transmission stalls.

Listing 6.1: Timeout calculation (pseudo code).

```
durations = []
timeout_min = 1000
timeout_max = 5000
timeout = 3000

WHILE receiving:
  FOR EACH suc_transfer IN successful_transfers:
    durations.append(suc_transfer.duration)

  FOR EACH exp_transfer IN expired_transfers:
    durations.append(1.5*timeout_max)

  IF durations.size > 20:
    timeout = 1.3*durations.moving_average(20)

  IF timeout > timeout_max: timeout = timeout_max
  IF timeout < timeout_min: timeout = timeout_min
```

**Timeout Management:** A chunk is considered stalled, if its transmission time exceeds a certain *timeout duration*, which can vary between 1000 ms and 5000 ms. Assuming that the maximum RTT is about 200 ms (a common value for Internet video streaming [81]), at most 25 round trips ($25 * 200\ ms \approx 5000\ ms$) should be needed to transmit the video chunk. Very large chunks may exceed this limit, but using such chunk sizes is regarded as unreasonable for adaptive video streaming, because they will lead to an unreasonable increase of the buffer needed on the client. The lower bound of the timeout duration is set to 1000 ms, to account for retransmissions and momentary stalls in congested networks. The initial value of the timeout duration is 3000 ms. The *transfer duration* is the time needed for downloading a chunk and is monitored for each chunk. To detect transmission stalls, the timeout duration is constantly updated with a moving average over the last 20 transfer durations plus a tolerance of 30%. Expired transfers are considered with an additional penalty in the moving average as well. In Listing 6.1, the pseudo code for the calculation of the timeout duration is shown.

**Priority Management:** The client coordinates the in-order transmission of the chunks based on the priority of the chunks (order within the video segment). Priority management improves the timeliness of the delivery by prioritizing video chunks required in the near future. For priority streaming, the chunks need to arrive in priority order (as shown in Figure 6.9). If the transmission of a chunk is stalled (see original chunk 4 in Figure 6.9), it will be requested twice again (see hatched chunks). The idea is based on the assumption that the probability of a successful download increases by using two concurrent streams. An important aspect of priority management is that it does not change the TCP-friendliness of the streaming system because the number of concurrent request-response streams is kept constant. Only the queues used for managing the download order of the chunks are updated, while the congestion control is ensured by the underlying TCP implementation of each request-response stream.

In the following chapter, an in-depth analysis of the request-response streaming system will be presented. The influence of the system parameters on the throughput performance and the ability to provide TCP-friendliness in congested networks will be evaluated.

CHAPTER

# 7 Performance of Parallel HTTP-based Request-Response Streams

Request-response streams are seen as a good choice for Internet video streaming, because they can recover fast from connection aborts and show a favorable behavior in presence of packet loss [52, 53]. The HTTP-based streaming process can be configured by several system parameters, which change the behavior of the request-response streams in terms of throughput performance and TCP-friendliness. The contribution of this chapter is an in-depth analysis of the streaming performance of the HTTP-based request-response video streaming system. The system is based on the architecture presented in Section 6.3. The streaming performance will be evaluated regarding the achievable throughput and the TCP-friendliness in different congestion scenarios.

## 7.1 Evaluation Methodology

To validate the enhanced model of the request-response streams (see Section 6.2.2) for the Internet video streaming use case, several factors have to be taken into account. Because the model is not only based on the intrinsic parameters of the request-response streams, but also considers the network's characteristics, different network scenarios were defined. In addition, the range of the test content's bit rates has to match the bandwidth of the bottleneck link to enable adaptive streaming. In the following, the choice of system parameters and network scenarios will be explained in detail.

### 7.1.1 System Parameters

The performance of a request-response streaming system depends on the system parameters and network's characteristics (see Equation 6.11). To evaluate the influence of the *chunk size* on the streaming performance, the size of the chunks is varied from very small to medium. Large chunks were omitted, because it is very unlikely to transmit such large chunks within a transfer duration of 5000 ms (see Section 6.3). Very large timeout durations (and consequently chunk sizes) should be avoided, because in this case the request-response streams lose their favourable characteristics (see Section 6.2). Five different chunk sizes ($l_{ch}$) were used in the evaluation, namely 20, 40, 80, 160 and 320 kBytes (kB).

While the chunk size basically determines the time the request-response stream continuously transmits data, the *number of concurrent request-response streams* $n_c$ can be used to efficiently utilize the available bandwidth. Compared to an infinite-source TCP connection, which always transmits data, a request-response stream has idle times, where no data is transmitted. To be able to compete over the fair share with other TCP data flows in a congested network, multiple concurrent request-response streams have to be used. For that reason, the number of request-response streams $n_c$ was varied in the range of 1 to 5.

The TCP-friendliness of a request-response streaming system usually depends on the chunk size and the number of concurrent streams. Because these two parameters cannot be changed arbitrarily, the *inter-request gap parameter* $t_{gap}$ was introduced in [52], which enables to steer the TCP-friendliness in a fine-grained manner. On the other hand, the inter-request gap increases the transmission latency, therefore $t_{gap}$ was limited to be 220 ms at maximum.

### 7.1.2 Network Scenarios

HTTP-based request-response streaming targets the Internet streaming use case. For that reason, two different network scenarios are considered important. First, the video data is transmitted over an *uncongested network link*, where the streaming system should be able to make use of all the bandwidth. While in the absence of packet loss TCP theoretically has no throughput limit due to its additive-increase algorithm (the congestion window can theoretically grow infinitely; see also Equation 6.1 with $p = 0$), request-response streams cannot scale infinitely. Because the system parameters fix the maximum achievable throughput

(at most $n_c * l_{ch}$ bytes can be transferred per RTT; see Equation 6.11), it is important to investigate which system parameters are suited for certain network characteristics (like the bottleneck bandwidth).

The second network scenario focuses on *congested network links* with different congestion levels, which may occur due to simultaneous Internet access over the same bottleneck link. A single request-response stream exhibits congestion control, because it works on top of a TCP connection. Multiple request-response streams are potentially unfair to concurrent TCP connections, because they are aggregated and used in a single application. To investigate the TCP-friendliness of the request-response streams, different congestion levels are emulated by 1 to 4 concurrent TCP downloads.

The performance of the request-response streams relies on the RTT and the amount of data which can be transferred per RTT. While the RTT is usually defined by the locations of the server and the client, the amount of data transferred in an RTT is restricted by the chunk size and the bottleneck router configuration. Two different bottleneck network bandwidths (4096 and 8192 kbps) were investigated and the bottleneck router is configured to allow only a maximum queuing delay of 200 ms by setting the router queue to a fixed size for a certain bandwidth ($l_q = BW * t_q$). Queuing delays larger than 200 ms are not considered because they render real-time applications like VoIP useless. To show the influence of the RTT on the throughput performance and the TCP-friendliness, RTTs ranging from 50 to 200 ms were evaluated.

Given all evaluated network and system parameters, a full evaluation run varying all parameters consists of 8000 streaming experiments. Each streaming experiment lasted for 500 seconds. The evaluation runs were repeated three times to cancel out variations imposed by the test environment.

## 7.2 Video Content

Priority streaming requires the video content to be encoded in a scalable fashion. For that reason, the test sequences were encoded using the H.264/SVC codec. The video codec is provided by the Joint Scalable Video Model (JSVM) [40] 9.18 software, offering a lot of configuration options. Because adaptive streaming based on scalable media relies on

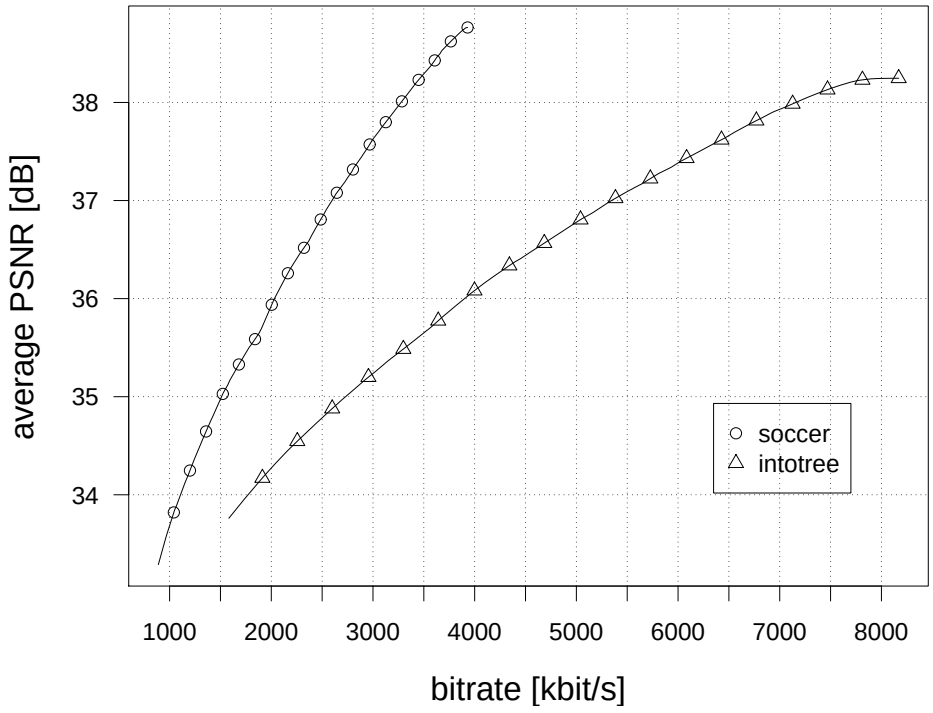



Figure 7.1: Rate-distortion curves of the test sequences [53].

the inherent scalability of the video bit stream, also the network scenarios (defined in Section 7.1.2) have to be considered in the encoding process. The range of the bit rates, which can be extracted out of the scalable bit stream, has to be aligned with the bandwidth of the bottleneck link.

The video sequence *soccer* was encoded in 4CIF resolution at 30 fps. The quantization parameters were chosen in such a manner that the bit rates range from approximately 1 to 4 Mbps (see Figure 7.1) and therefore match the requirements of adaptive streaming on the 4 Mbps bottleneck link. To fully utilize the 8 Mbps link, the *in to tree* sequence was encoded in 720p resolution at 50 fps. For this sequence, the bit rates range from approximately 1.5 to 8 Mbps. Each encoded video sequence comprises an H.264/AVC backward compatible base layer and one MGS quality enhancement layer. Within the MGS quality enhancement layer, the 16 transform coefficients are uniformly partitioned into four NAL units (4x4), resulting in five different quality representations for each video frame (base layer plus four quality enhancement layers). Because in MGS the quality can be changed on frame level, it is possible to extract more than five quality representations out of the scalable video

bit stream (theoretically up to $1 + n_{frames} * n_{enhlayers}$). The Quality Level Assigner tool (JSVM) was used to assign 64 different PRIDs to the NAL units based on rate-distortion values. Figure 7.1 shows the rate-distortion values of the video sequences, ranging from the lowest bit rate (highest priority) to the full bit rate (lowest priority).

Each test sequence has a play-out duration of 10 seconds, which is also considered to be a good video fragment size. Because each streaming experiment lasts for 500 seconds, the content is streamed 50 times in a loop.

## 7.3 Performance Metrics

The goal of adaptive Internet video streaming is to maximize the stream-out rate, while still providing TCP-friendliness. This section briefly reviews the main performance metrics used for characterizing Internet video streaming systems.

**Streaming Performance:** A commonly used metric for evaluating the streaming performance is the *average throughput*. It is measured for each video fragment (in this case every 10 seconds). In addition, the *download duration of the chunks* is measured. To be able to compare the performance values of each system parameter set, the throughput and the download duration are averaged over the different RTTs.

**TCP-friendliness:** The Internet is a best-effort network, the stability of which heavily depends on the use of congestion control. TCP is the main Internet protocol featuring congestion control. As a result, TCP-friendliness is seen as a required characteristic of a video streaming system, because video streaming will amount for a large share of the Internet's traffic [55, 94].

$$\frac{RR}{TCP} = \frac{r_{rr}}{\frac{1}{n_{tcp}} \sum_{i=1}^{n_{tcp}} r_{tcpi}} \tag{7.1}$$

The *TCP fairness ratio* $RR/TCP$ of the request-response streams and concurrent TCP streams (see Equation 7.1) is an indicator for how good the bandwidth is shared between concurrent data streams. In a congested network, it should give a good idea of how to choose the system parameters to achieve TCP-friendliness for a given bottleneck bandwidth of the last-mile link. A fairness ratio of $RR/TCP = 1$ indicates that the request-response streams

only use their fair share of the network's 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 use less than their fair share. Like for the throughput, the fairness ratio is averaged for the different congestion levels (number of competing downloads), in order to get a single value for a specific system parameter set.

**Comparison with Model:** The enhanced model (see Equation 6.11) will be validated by comparing the measured results with the model with respect to the average throughput and the download duration of the chunks.

**Video Quality:** Finally, the video quality will be evaluated with a parameter set providing good performance and TCP-friendliness. The network was configured to have a fixed bottleneck bandwidth of $BW = 8192kbps$ and a maximum queuing delay of 200 ms. For this network characteristics a system parameter set of $l_{ch} = 160\ kB$, $n_c = 5$ and $t_{gap} = 210\ ms$ was chosen, which is considered a good choice for a real-world deployment. To show the influence of over-provisioning on the video quality, also the test sequence *soccer* (with 4 Mbps maximum rate) was evaluated in this test. When looking at Figure 7.1, 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.

## 7.4 Test Setup

As depicted in the envisioned Internet video streaming use case (see Figure 1.1), the streaming process on the client is pulling the video data from the server via the Internet and the last mile network, which also is seen as the bottleneck link. The test setup is similar to the one in Section 5.3.1. Figure 5.6 shows the network topology. The main difference of this evaluation is that additionally packet loss is emulated in the access network.

The implementation of the request-response streaming system is based on Python and the HTTP library *libcurl* [90]. In all streaming experiments, Server 1 delivers the video data chunks requested by Client 1. The network congestion is emulated by concurrent HTTP downloads from Server 2 initiated by Client 2. The Apache HTTP Server [27] was used on

the servers for delivering the video chunks and the concurrent downloads.

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

## 7.5 Results

The evaluation of video streaming with HTTP-based request-response streams comprises a broad range of experiments. To rate the performance of this streaming solution, the average throughput, the average download duration of the chunks, and the TCP-friendliness was measured. As explained in Section 7.3, the results will be averaged over the different RTTs, to show the findings in a concise manner.

### 7.5.1 System Parameters

In the first part of this section, the influence of the three system parameters on the throughput performance will be discussed. The system parameters are the chunk size $l_{ch}$, the number of concurrent request-response streams $n_c$ and the inter-request gap $t_{gap}$. In Figures 7.2 and 7.3, the average throughput performance and the average download duration of a chunk are shown for the bottleneck bandwidths $BW = 4096\ kbps$ and $BW = 8192\ kbps$, respectively. There was no additional traffic on the test network, to allow the request-response streams to fully utilize the available bandwidth. The results show that a single request-response stream cannot utilize all of the available bandwidth, because of the nature of the request-response paradigm and the inter-request gap. The influence of the system parameters is discussed in the following.

#### Chunk size $l_{ch}$

When comparing the throughput performance for different chunk sizes, it becomes evident that larger chunk sizes utilize the available network bandwidth more efficiently. For example, this can be noticed in Figure 7.3, when directly comparing the chunk sizes 20 kB and 160 kB. While the 20 kB chunks only use about half of the network's link capacity with 5 streams, the 160 kB chunks are able to utilize all of the available bandwidth even with 3 streams. A single request-response stream using large chunks behaves similarly to an infinite-source TCP stream. While this leads to an increased throughput performance,

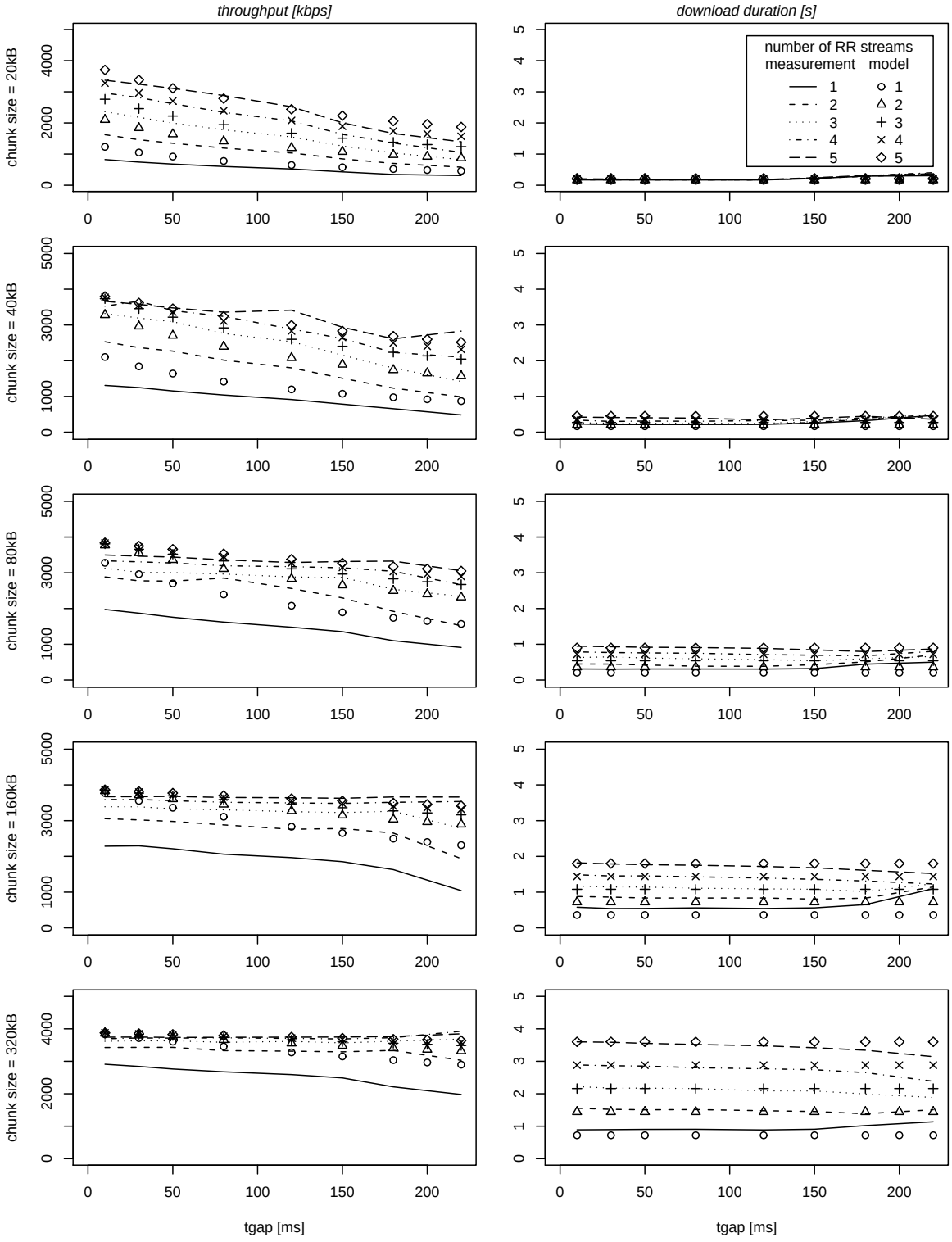



Figure 7.2: Measured and modeled throughput performance $r_{rr}$ and download duration of a chunk $t_{ch}$ for the request-response streams. The results are shown for a fixed bottleneck bandwidth of $BW = 4096\ kbps$, a maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs to provide a single performance value. [53]

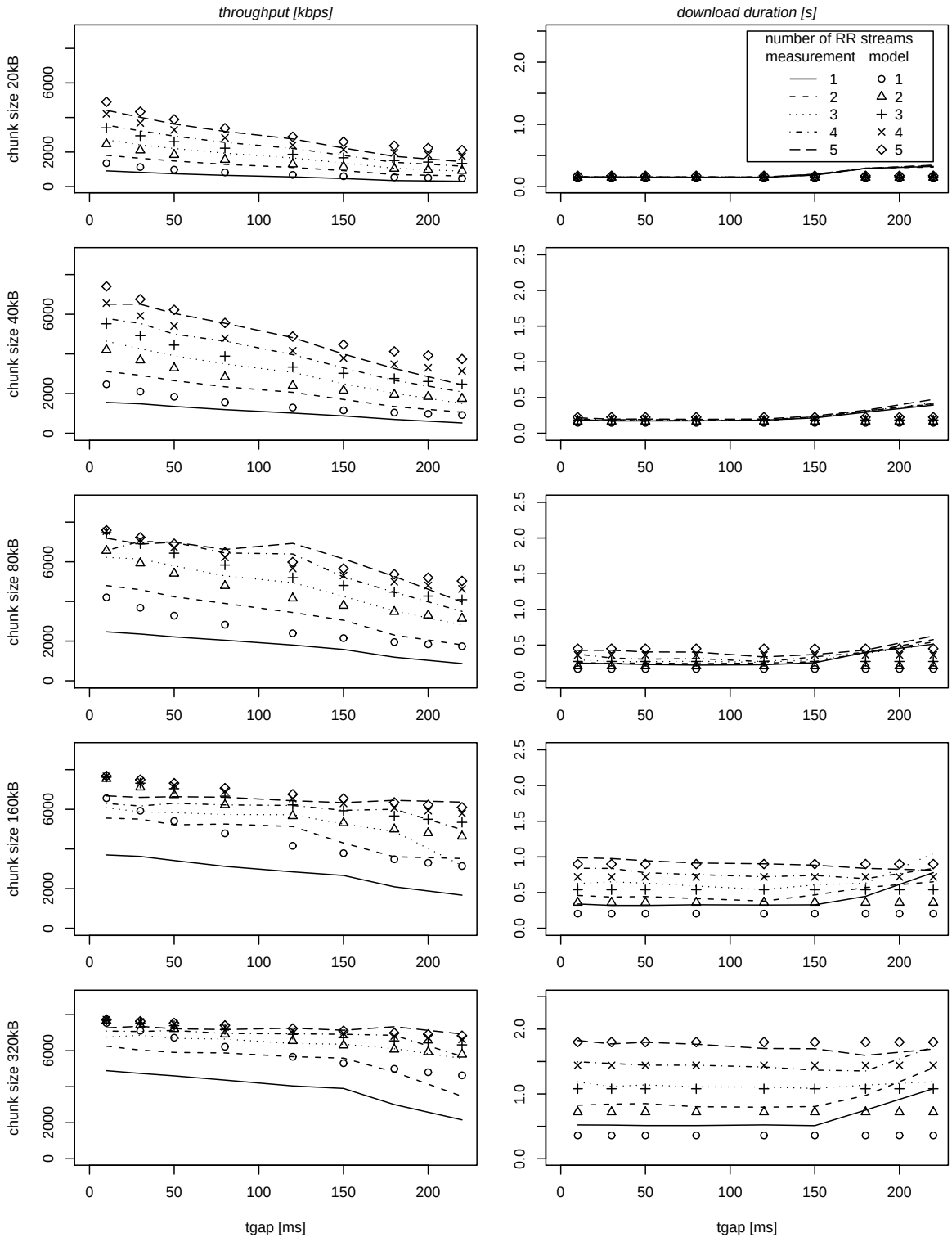



Figure 7.3: Measured and modeled throughput performance $r_{rr}$ and download duration of a chunk $t_{ch}$ for the request-response streams. The results are shown for a fixed bottleneck bandwidth of $BW = 8192\ kbps$, a maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs to provide a single performance value. [53]

also the congestion in case of multiple streams increases. In uncongested networks, the download duration increases linearly with the chunk size, because the available bandwidth $BW$ is fixed to a certain value (e.g., with a single stream $t_{ch} = l_{ch}/BW$). However, if the available bandwidth is exhausted, enlarging chunks will only increase the download duration without any improvement of the throughput.

#### Number of concurrent streams $n_c$

The performance of a request-response system can also be boosted by the number of streams used to transmit the data. The number of streams shows a behavior similar to the chunk size. When looking at the throughput performance of chunks with a size of 20 kB in Figure 7.2, a higher number of streams leads to a higher throughput. In general, multiple streams are also more error resilient (see Section 6.2), so it is good to increase the number of streams as much as possible. On the other hand, a high number of concurrent streams tends to generate self-congestion, because of the high number of TCP connections (basis of each HTTP-based request-response stream), which all try to maximize their throughput. It can be assumed that self-congestion takes place, if increasing the number of streams does not lead to a substantial throughput increase (see chunk size 320 kB in Figure 7.2). As a consequence, also the download duration increases with the number of streams. A finding of this evaluation is that the number of streams can be safely increased as long as the download duration does not increase proportionally.

#### Inter-request gap $t_{gap}$

While the two aforementioned system parameters are used to increase the throughput and the error resilience, the sole purpose of the inter-request gap parameter is to accomplish TCP-friendliness. It has obviously no positive effect on the throughput or download duration, because the artificial gap between the requests only increases the time the request-response streams are not transporting data. As a consequence, larger inter-request gaps lead to decreased throughput. Usually, the inter-request gap has no influence on the download duration, because the inter-request gap is applied between the downloads of the chunks. In case of self-congestion, large inter-request gaps may reduce the download duration by lowering the amount of congestion. This can be noticed at large chunk

sizes and a high number of streams, like in Figure 7.2 with $l_{ch} = 320kB$ and $n_c = 5$. Nevertheless, this effect is limited, because in this evaluation $t_{gap}$ is 220 ms at maximum. In addition, Figure 7.2 reveals that the performance of small chunks is more influenced by the inter-request gap (larger slope), because the inter-request gap is applied between consecutive downloads of chunks and smaller chunks need less time to be transferred.

Additionally, the values of the enhanced model for the estimated throughput are shown in Figures 7.2 and 7.3. The enhanced model shows a good correlation with the measured values and can give a quite realistic estimation on the achievable throughput. Large chunk sizes and heavy self-congestion usually lead to an increased amount of stalls in the data transmission. While the error of the modeled throughput increases because the model is not aware of these stalls, the model still predicts the upper bound of the throughput. A similar behavior can be noticed for the download duration.

### 7.5.2 TCP-friendliness

For the investigation of TCP-friendliness, one to four HTTP downloads were started concurrently to the request-response streaming system. By changing the number of concurrent downloads, it is possible to adjust the network's congestion to different levels. The throughput values are recorded for both, the HTTP downloads and the streaming system, and the *TCP fairness ratio* $RR/TCP$ is calculated (see Equation 7.1). To present a single value for the TCP-friendliness of a specific parameter set, the fairness ratios were averaged over the different RTTs and congestion levels. In Figure 7.4, the TCP-friendliness in terms of throughput fairness for the three system parameters is shown. The value $RR/TCP = 1$ is marked, because it shows under which conditions the request-response streaming system achieves TCP-friendliness. The measurements show that all three parameters have an influence on the TCP-friendliness. While the TCP-friendliness decreases ($RR/TCP$ increases) with the number of concurrent streams and the chunk size, an increasing inter-request gap is able to increase the TCP-friendliness. For system configurations which generate self-congestion (very large chunks and low bandwidth, e.g., 320 kB at $BW = 4096$ *kbps*), it becomes increasingly difficult to find parameter sets with multiple streams which exhibit TCP-friendliness. This is because the inter-request gap is usually limited (in this case to

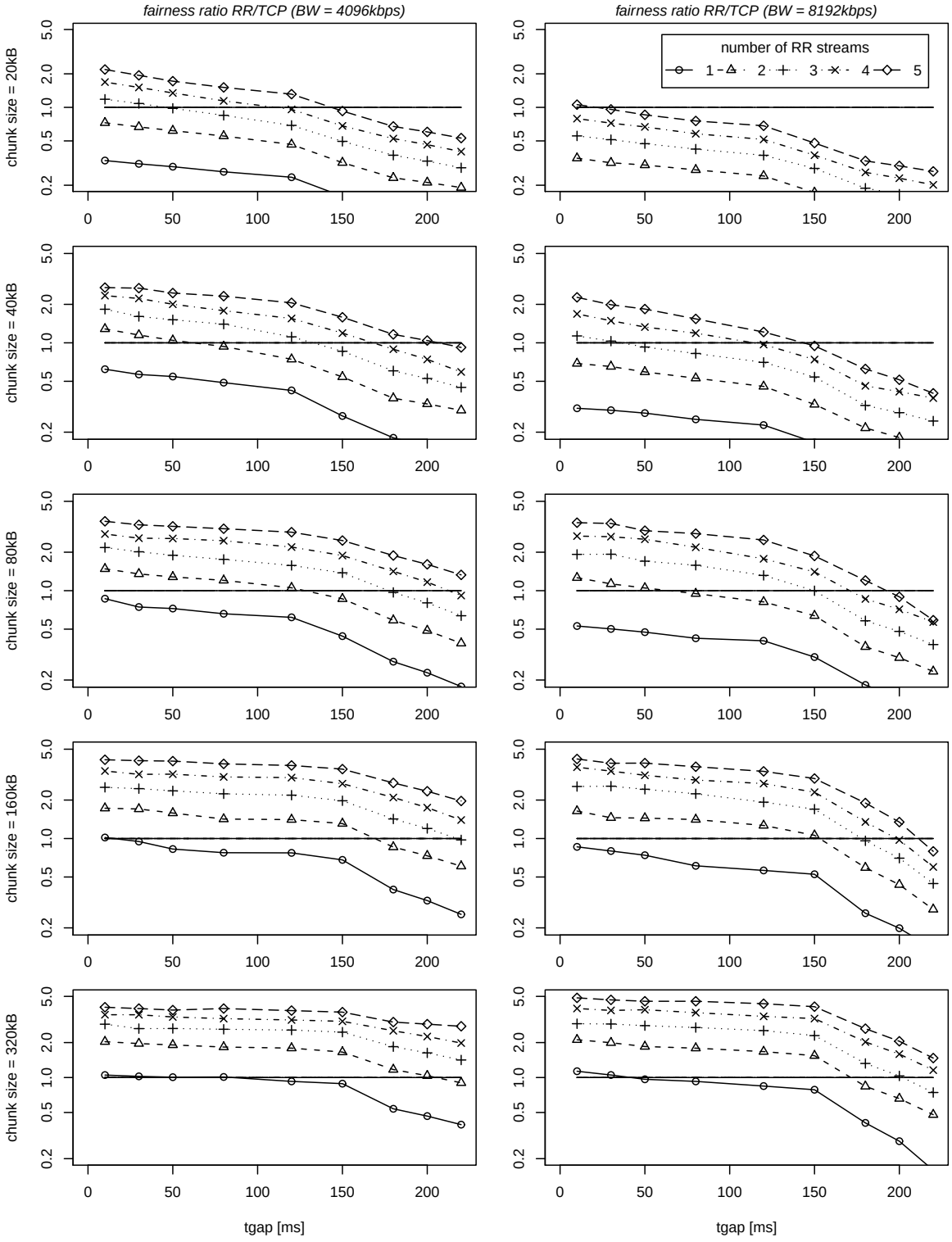



Figure 7.4: Measured TCP fairness ratio $RR/TCP$ of the request-response streams compared to 1 ... 4 concurrent HTTP downloads. The results are shown for two bottleneck bandwidths ($BW = 4096\ kbps$ and 8192 $kbps$ and maximum queuing delay of $t_q = 200\ ms$ and averaged over the RTTs and different congestion levels to provide a single performance value. [53]

220 ms). Large inter-request gaps also call for large chunk sizes to compensate for the increased idle times. But the chunk size is usually restricted to small and medium, because otherwise all favourable characteristics of the request-response streams get lost (see Section 6.2).

Another interesting finding can be noticed when directly comparing the fairness ratio plots for $BW = 4096\ kbps$ and $BW = 8192\ kbps$ (see Figure 7.4). The values of the fairness ratio of a certain chunk size in $BW = 4096\ kbps$ are similar to the values of the doubled chunk size in $BW = 8192\ kbps$. Assuming that the time needed to download a chunk is directly proportional to $BW$, it becomes quite clear that for these two combinations the download duration $t_{chloss}$ should be similar (see Equation 6.10). The TCP-friendliness is usually dependent on the ratio $t_{chloss}/t_{gap}$ and the number of concurrent streams, because this ratio defines the amount of time a request-response stream is active (download time vs. idle time). In this case, the ratio $t_{chloss}/t_{gap}$ is similar and 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 changing TCP-friendliness.

The main use case for request-response streams is Internet video streaming. For that reason, a request-response streaming system should provide TCP fairness ($RR/TCP = 1$). Because usually there is more than a single system parameter set which provides TCP-friendliness, a good trade-off between throughput performance and TCP-friendliness has to be found. A large number of request-response streams improves the performance in case of packet loss or transmission stalls. For that reason, medium sized chunks are seen as a good choice, because they do not introduce too much overhead like small chunks and inflict less damage on the in-order throughput than large chunks. Of course, the chunk size classification should be always seen relative to the bandwidth of the bottleneck link. The most important factor is the estimated download duration of a chunk $t_{chloss}$ (see Equation 6.10), which should be similar for all chunks within a class. Therefore, a system parameter set with a large number of streams and a download duration of $t_{ch} \approx 1 second$ at maximum RTT is seen as most appropriate, because also the request-response streaming system assumes the download duration to be one second in case of no congestion (minimum

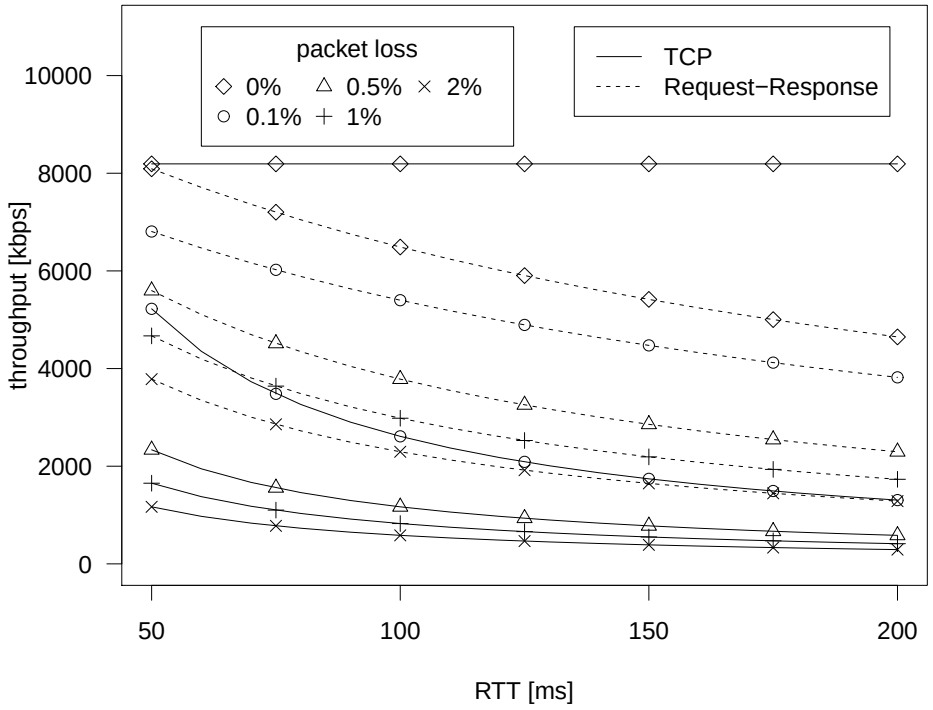
timeout duration is 1000 ms; see Section 6.3).

### 7.5.3 Comparison with Enhanced Model

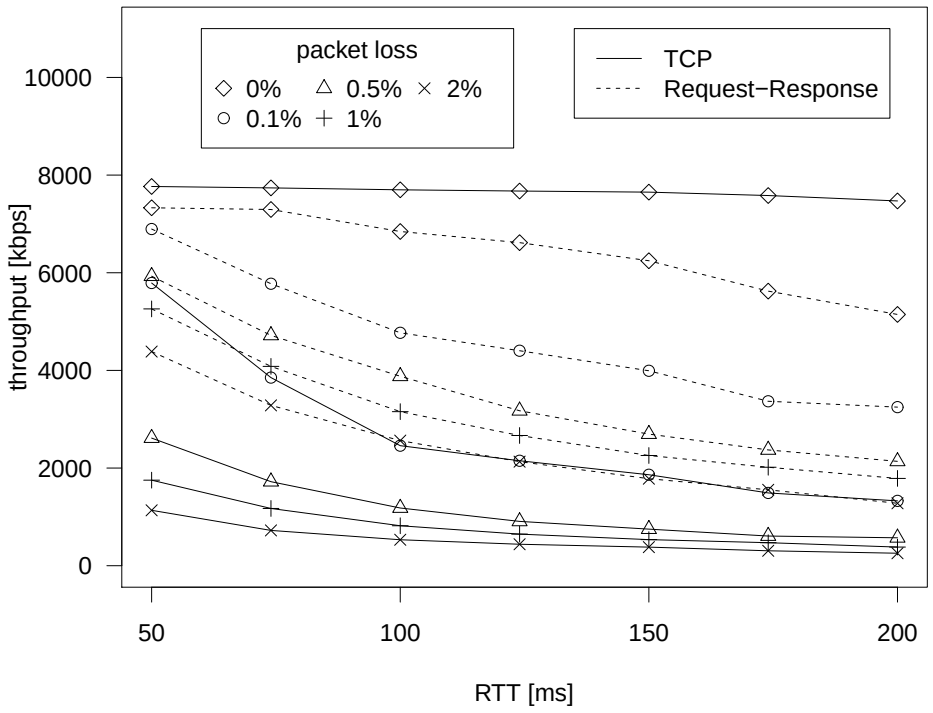
The measured throughput performance for a fixed bottleneck bandwidth of $BW = 8192$ *kbps* is shown in Figure 7.5(b). For the transmission of the video data, a chunk size of 160 kB was selected and 5 concurrent request-response streams with an inter-request gap of 210 ms were used, resulting in an estimated download duration of 1.2 seconds ($RTT = 200ms$). In direct comparison with the measured throughput of a single TCP connection, the benefit of the multiple streams becomes evident. The throughput of the single TCP connection rapidly decreases with increasing packet loss, while the request-response streams are able to mitigate the effects. In addition, the request-response streams are able to utilize the available bandwidth in an uncongested network with no packet loss ($p = 0$ %), but of course not as good as TCP, because of the request-response paradigm and the HTTP overhead. When comparing the values of the enhanced model with the measurements (see Figures 7.5(a) and 7.5(b)), a good correlation between the model and the real measured values can be observed. Currently, also the enhanced model is a simplification of the real world problems, but it can be used to estimate the throughput performance of the request-response streams (with and without packet loss).

### 7.5.4 Video Quality

The influence of the request-response streams on the video quality is similar to the throughput performance. The video quality in terms of PSNR for the test sequences *soccer* and *in to tree* is shown in Figures 7.6(a) and 7.6(b), respectively. If no packet loss is present, TCP and the request-response streams perform well, with an advantage for TCP in terms of network utilization. In case of packet loss, this fact changes dramatically. At high packet loss rates, the throughput of TCP falls below 1.5 Mbps. As a result, the single TCP connection cannot even transmit the base layer of the HD video. High RTTs and the HD video *in to tree* inflict difficulties on the request-response streams as well. It is quite evident, though, that the request-response streams can better utilize the available bandwidth, while the throughput performance of the single TCP stream is limited by the

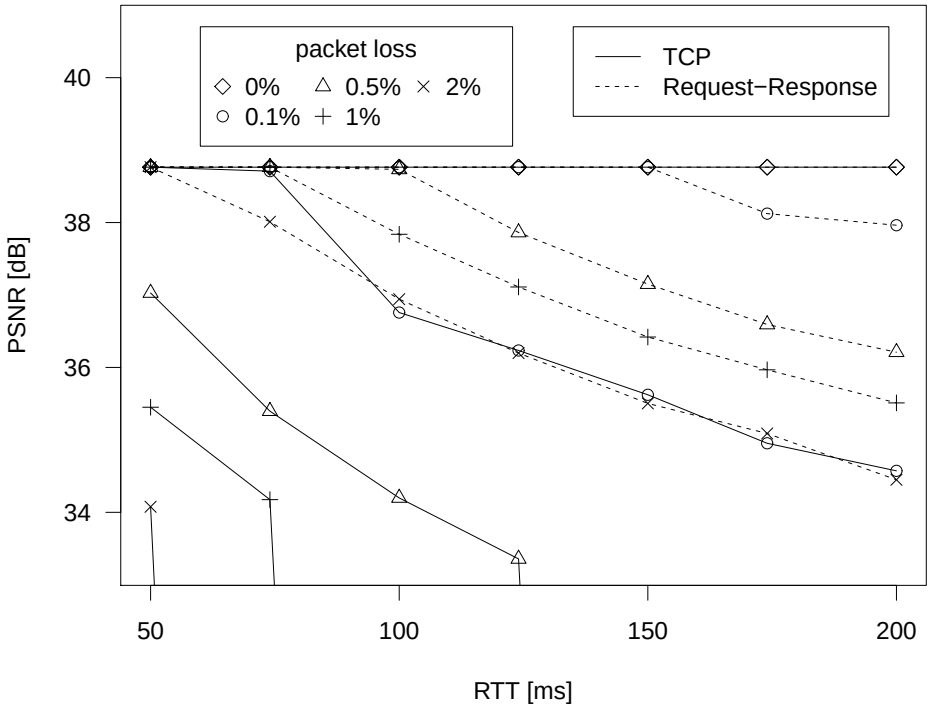



(a) Modeled throughput

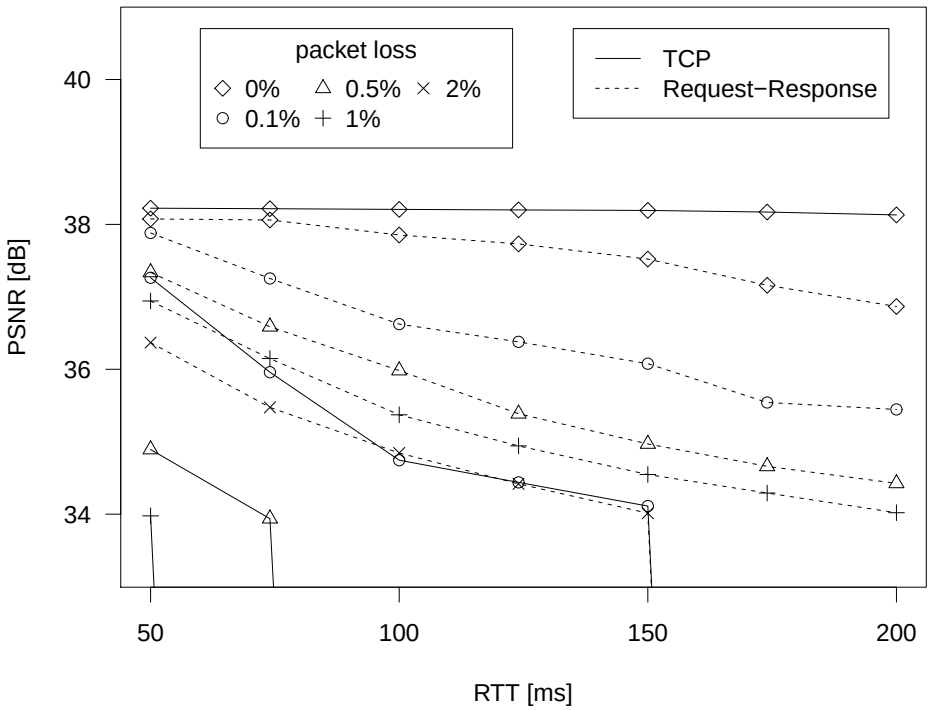



(b) Measured throughput

Figure 7.5: Measured and modeled 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 6.6. [53]




(a) Test sequence *soccer*




(b) Test sequence *in to tree*

Figure 7.6: Measured video quality of the streamed test sequences 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 $p$. [53]

packet loss (see Equation 6.1). Unfortunately, the difference in terms of PSNR can only be calculated for a packet loss rate of $p = 0.1$ % and the test sequence *soccer*, which is 2.04 dB on average. In the other network scenarios with packet loss ($p > 0.1$ %) and for the whole sequence *in to tree*, TCP cannot deliver the video at every given RTT. So in presence of packet loss, the main advantage of the request-response streaming system compared to a single TCP connection is that it can deliver the video continuously, while TCP may not be capable of delivering the video at all.

The presented results indicate that request-response streams can achieve a good performance in wired networks. To assess the performance of the request-response streams in a mobile video streaming scenario, an evaluation in a cellular network with high bandwidth fluctuations will be presented in the following chapter.

CHAPTER

# 8 Mobile Video Streaming using Request-Response Streams

In recent years, IP-based over-the-top services (e.g., services on top of an Internet connection) have become increasingly important. The reason for that is the easy deployment of such services, because they are not tailored to certain types of networks (unlike DVB-H [3], for example). Usually, TCP or HTTP is used for transmitting the data, because these protocols are most likely to be supported by any IP-based network. With the advent of 3rd generation (3G) wireless networks like UMTS, also high speed data services emerged (e.g., HSDPA), which enable over-the-top services like VoD in cellular networks. Figure 8.1 illustrates a mobile video streaming use case, where - in addition to video streaming in wired networks - also cellular networks are considered (e.g., streaming video to a user

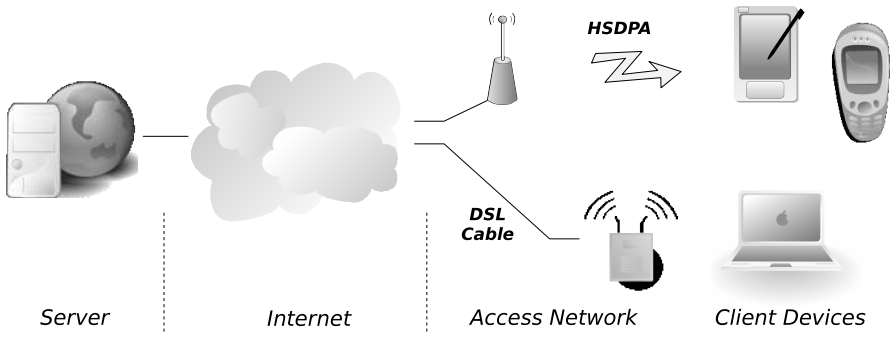



Figure 8.1: Mobile video streaming use case.

in a moving car). Cellular networks behave quite differently compared to DSL or cable networks. They usually exhibit high RTTs and bandwidth fluctuations. In addition, the connectivity between the mobile device and the base station may be temporally interrupted, resulting in times when no data can be transmitted at all.

A performance evaluation of the request-response streaming system in a cellular network scenario will be the contribution in this chapter. The streaming system will have to cope with high bandwidth fluctuations and RTTs. In the following, the cellular network traces used in the evaluation and the evaluation methodology will be presented in detail.

## 8.1 Cellular Network Traces

To emulate the cellular network in the evaluation environment, three existing traces of a GPRS/HSDPA network of the Austrian cellular network provider A1 were used. In [66], three different experiments were conducted while driving on the Austrian freeway A2 on different tracks. The throughput of an infinite-source TCP download was measured, resulting in three throughput traces of 601, 575 and 599 seconds length.

- *Trace 1 (601 seconds):* Driving on the freeway A2, passing by the city of Villach in the direction to Klagenfurt.

- *Trace 2 (575 seconds):* From the Klagenfurt University on the freeway A2 to the service area near Techelsberg.

- *Trace 3 (599 seconds):* From the service area near Techelsberg on the freeway A2 to the exit of Klagenfurt.

Figure 8.2 depicts the recorded throughput traces. While driving along the freeway, changes in the reception quality led to high bandwidth fluctuations in the cellular network. A low reception quality consequently results in a low physical transmission rate. In addition, link layer retransmissions may reduce the achievable throughput.

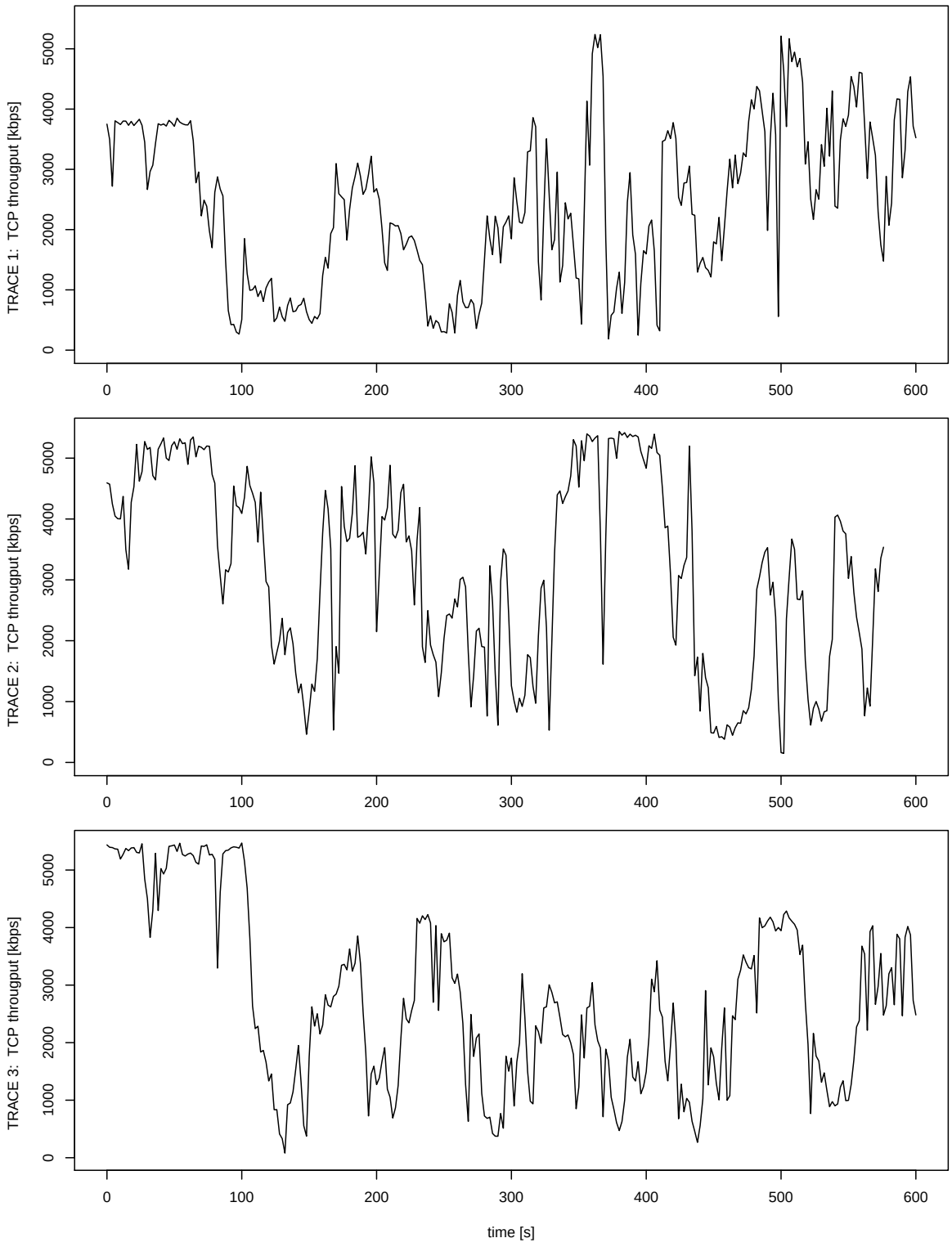



Figure 8.2: Network TCP throughput traces of the cellular network.

## 8.2 Evaluation Methodology

The evaluation setup is similar to the setup in Section 5.3.1. Figure 5.6 shows the evaluation network topology. The main difference in this evaluation is that the bandwidth of the access network is emulated dynamically based on throughput traces. The streaming process on the client is pulling the video data from the server via the Internet and the access network (cellular network), which also is seen as the bottleneck link. The request-response streaming system is implemented in Python, using the HTTP library *libcurl* [90]. Server 1, running the Apache HTTP Server, delivers the video data chunks requested by Client 1. Three different throughput traces are used to emulate the cellular network (see Section 8.1). In the emulation, the bandwidth of the access network is adjusted every two seconds.

For this evaluation, the video sequence *big buck bunny* [1] was encoded in 480p resolution using the H.264/SVC video codec. The first 600 seconds of the video sequence were considered in the tests. Hence, the length of the test sequence is $t_{video} = 600$ seconds. Before encoding, the video sequence was split into video fragments of different size. To determine the impact of the video fragment size on the streaming performance, three different fragment sizes $t_{frag}$, 10, 20 and 30 seconds, were evaluated.

The encoded video comprises an H.264/AVC backward compatible base layer and one MGS quality enhancement layer. Within the MGS quality enhancement layer, the 16 transform coefficients are uniformly partitioned into four NAL units (4x4), resulting in five different quality representations for each video frame (base layer plus four quality enhancement layers). Because in MGS the quality can be changed on frame level, it is possible to extract more than five quality representations out of the scalable video bit stream (theoretically up to $1 + n_{frames} * n_{enhlayers}$). Finally, the Quality Level Assigner tool (JSVM) was used to assign 64 different PRIDs to the NAL units based on rate-distortion values. To be able to compare the performance of the request-response streaming system with the results presented in [66], the H.264/SVC encoder was configured to provide video bit rates ranging from 200 kbps (base layer) to 4500 kbps (highest quality).

The parameter sets were derived from Figure 7.4 for a bottleneck bandwidth of $BW = 8192\ kbps$, because it is assumed that the maximum bandwidth of the HSDPA

Table 8.1: Parameters sets for test sequences

| parameter set | chunk size $l_{ch}$ [kb] | number of rr streams $n_c$ | inter-request gap $t_{gap}$ [ms] |
|---|---|---|---|
| 1 | 40 | 5 | 140 |
| 2 | 80 | 4 | 170 |
| 3 | 160 | 3 | 180 |
| 4 | 160 | 5 | 210 |

network is 7.2 Mbps. A suitable parameter set utilizes the available bandwidth efficiently and provides TCP-friendliness in case of congestion. Three different chunk sizes $l_{ch}$ (40, 80 and 160 kB) with different numbers of connections (ranging from 3 to 5) were investigated. The request-response streaming system was configured using only fair parameter sets. Table 8.1 shows the used system parameter sets.

In [66], several HTTP-based adaptive video streaming systems based on stream switching were evaluated. The metrics used for comparing the measured results were the following.

- *Bit rate of the streamed video $br_{video}$:* The bit rate of the video, which is reconstructed at the client should be maximized.
- *Number of quality switches $n_{switch}$:* Changes in the video quality are usually recognized by the user. As a result, they should be kept low.
- *Buffer level on the client $t_{buf}$:* The buffer level on the client gives insights into the behavior of the adaptive algorithm.
- *Average unsmoothness $t_{unsmooth}$:* The time in seconds the playback of the video is stalled.

To be able to directly compare the results of this evaluation with the ones in [66], these metrics will be also investigated in this evaluation.

## 8.3 Results

In this evaluation, the performance of the request-response streaming system is investigated in a cellular network scenario. Three different network traces are used to emulate the cellular

network, with bottleneck bandwidths ranging from 200 kbps to 5.5 Mbps. In addition, the parameters sets and video fragment sizes are varied.

Figures 8.3, 8.4 and 8.5 show the performance in terms of video bit rate for the parameter set 1 and video fragment sizes of 10, 20 and 30 seconds, respectively. Similar, the results for the parameters sets 2, 3 and 4, are shown in Figures 8.6, 8.7 and 8.8, Figures 8.9, 8.10 and 8.11, and Figures 8.12, 8.13 and 8.14, respectively. In the following, the influence of the video fragment size and system parameters will be discussed.

**Video Fragment Size:** When directly comparing the results for different video fragment sizes, it can be noticed that larger fragment sizes can be utilized to smooth out the time-varying bandwidth. Priority streaming can bridge times with low throughput by averaging the video bit rate over a larger period of time (see also Section 4.3).

Larger video fragment sizes can also increase the effective throughput. Chunks which are fetched near the end of the video fragment's time slot are potentially lost, because they do not arrive on time. While the number of chunks near the end of the video fragment's time slot is usually constant (approximately $n_c$ chunks), the overall number of fetched chunks increases with the video fragment size. Hence, the transmission efficiency improves with increasing video fragment sizes. As a result, the throughput and, consequently, the video bit rate is usually higher for larger video fragments.

Because of priority streaming, large fragment sizes additionally reduce the number of quality changes without a penalty in terms of video bit rate (see Section 5.2). But large video fragments usually increase the startup-delay.

**System Parameters:** While in Section 7.5 a static bottleneck bandwidth was assumed, in this cellular network scenario, the bottleneck bandwidth varies significantly. This has also to be considered in the selection process of a parameter set suited for cellular networks, because the request-response streaming system defines a maximum download duration $timeout_{max}$ of 5000 ms. Because the download duration of a chunk depends only on the chunk size (see Figure 7.3), it is important to know the expected minimal bottleneck bandwidth to be able to check if a chunk can be transmitted within the maximum download duration. Assuming a minimal bottleneck bandwidth of BW=200 kbps, the maximum chunk size which can be transmitted would be $l_{chmax} = BW * timeout_{max} \approx 120\ kb$. When looking at the results

with chunk sizes of 160 kb, it becomes clear that the performance of the system significantly deteriorates in times with low bandwidth. In some cases not even the base layer can be transmitted within the given time frame. As a consequence it is not recommended to use chunk sizes larger than 80 kB for these cellular network scenarios.

Nevertheless, it can be noticed that large chunk sizes adapt better to high network bandwidths. This is because large chunk sizes have in general a better throughput performance than small chunks with the same TCP-friendliness (see Figure 7.3 and 7.4).

**Performance Metrics:** Figure 8.15 summarizes the results of this evaluation by averaging the performance values for the three cellular network traces. Larger chunk sizes can slightly increase the average video bit rate, but at the cost of unsmoothness which rapidly increases with the chunk size (see video fragments of size 10 seconds). In general, the best performance can be achieved with large video fragments, because priority streaming averages the video bit rate over the used time slot. As a result, unsmoothness is unlikely to happen.

The maximum number of quality switches is dependent on the play-out duration of the video fragments $t_{frag}$ and the play-out duration of the whole test sequence $t_{video}$. When considering equal play-out durations for all streamed video fragments within an evaluation run, the maximum number of quality switches can be defined as $n_{switchmax} = t_{video}/t_{frag}$, which is for the evaluated fragment sizes 10, 20 and 30 seconds and a sequence length of 600 seconds, 60, 30 and 20, respectively. Nevertheless, the real number of quality switches is usually even lower and can be seen in Figure 8.15. In priority streaming, the received video fragment is buffered till the last data packet is retrieved (see Section 4.3). For that reason, the client buffer level can be considered constant as one video fragment size, which is 10, 20 and 30 seconds, respectively.

As mentioned before, when coping with high bandwidth fluctuations and low bandwidth, it is recommended to stick to small chunk sizes and large video fragments. For these mobile traces, the parameter set 1 and the video fragment size 30 seconds is most likely the best combination. Because even video fragments of size 20 seconds feature good performance for parameter set 1, it can be assumed that there is a sufficiently large safety margin in case of bad network quality.

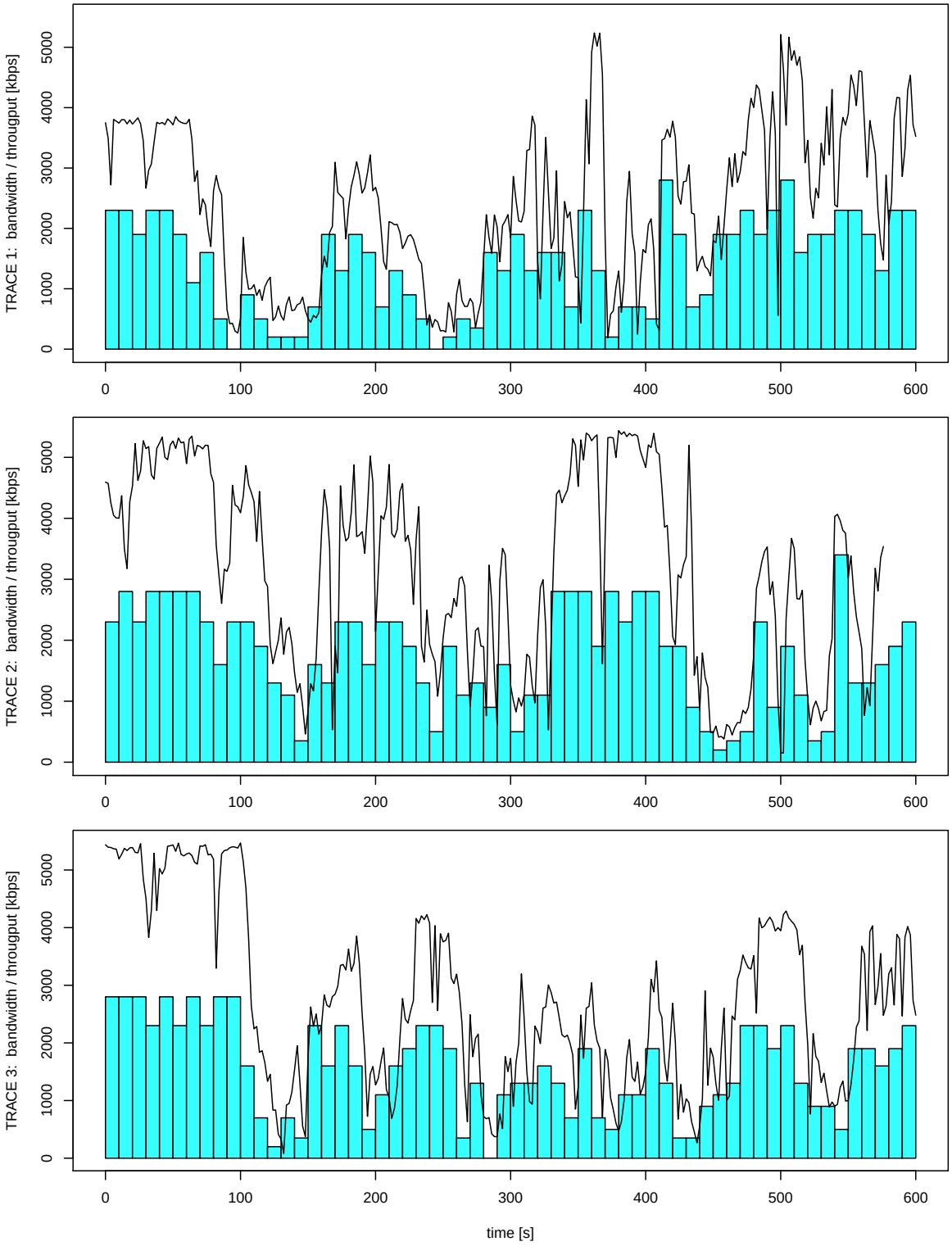



Figure 8.3: Parameter set 1: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 10s, l_{ch} = 40\ kB$, $n_c = 5$, $t_{gap} = 140\ ms$) at a fixed RTT of 150 ms.

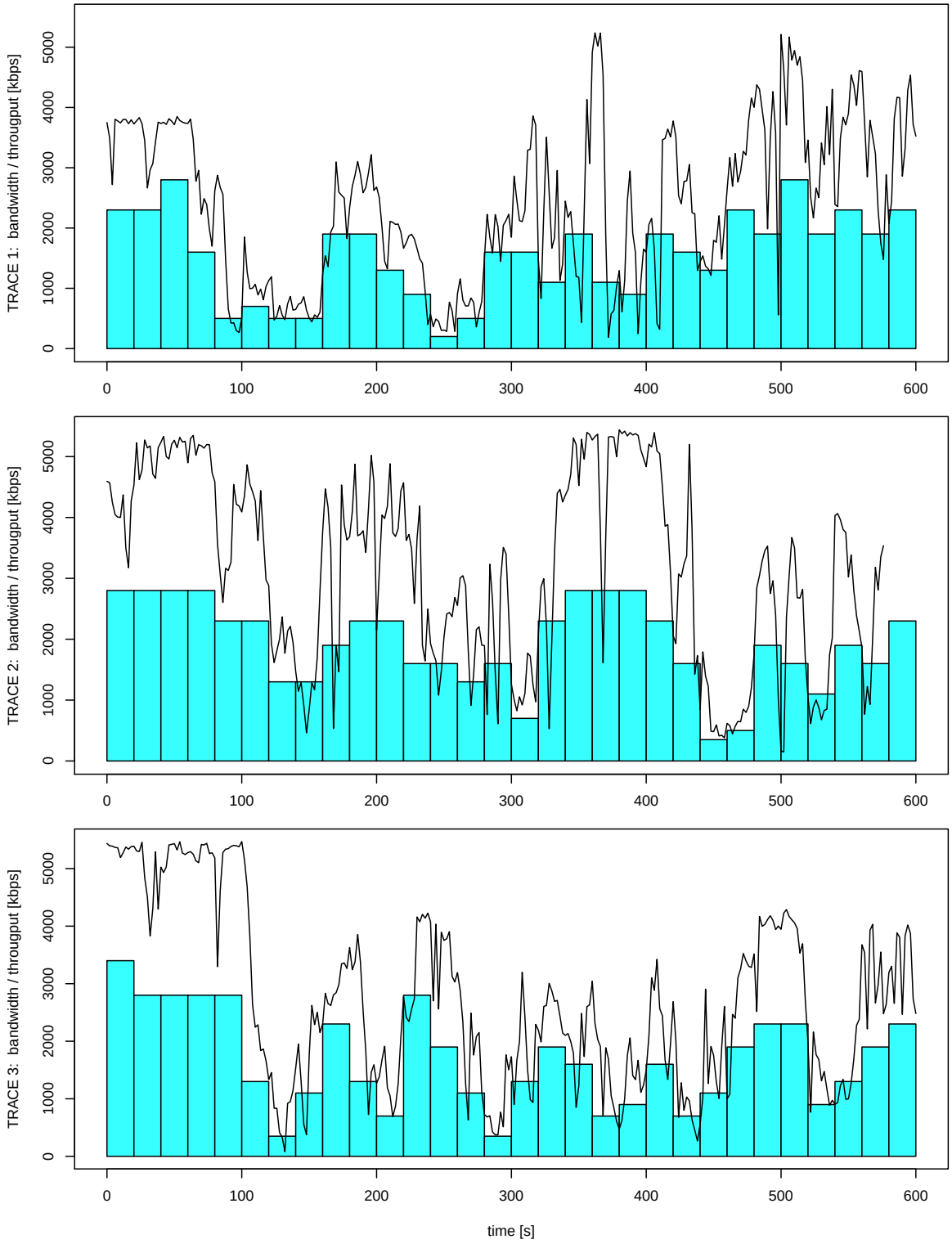



Figure 8.4: Parameter set 1: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 20s, l_{ch} = 40\ kB$, $n_c = 5$, $t_{gap} = 140\ ms$) at a fixed RTT of 150 ms.

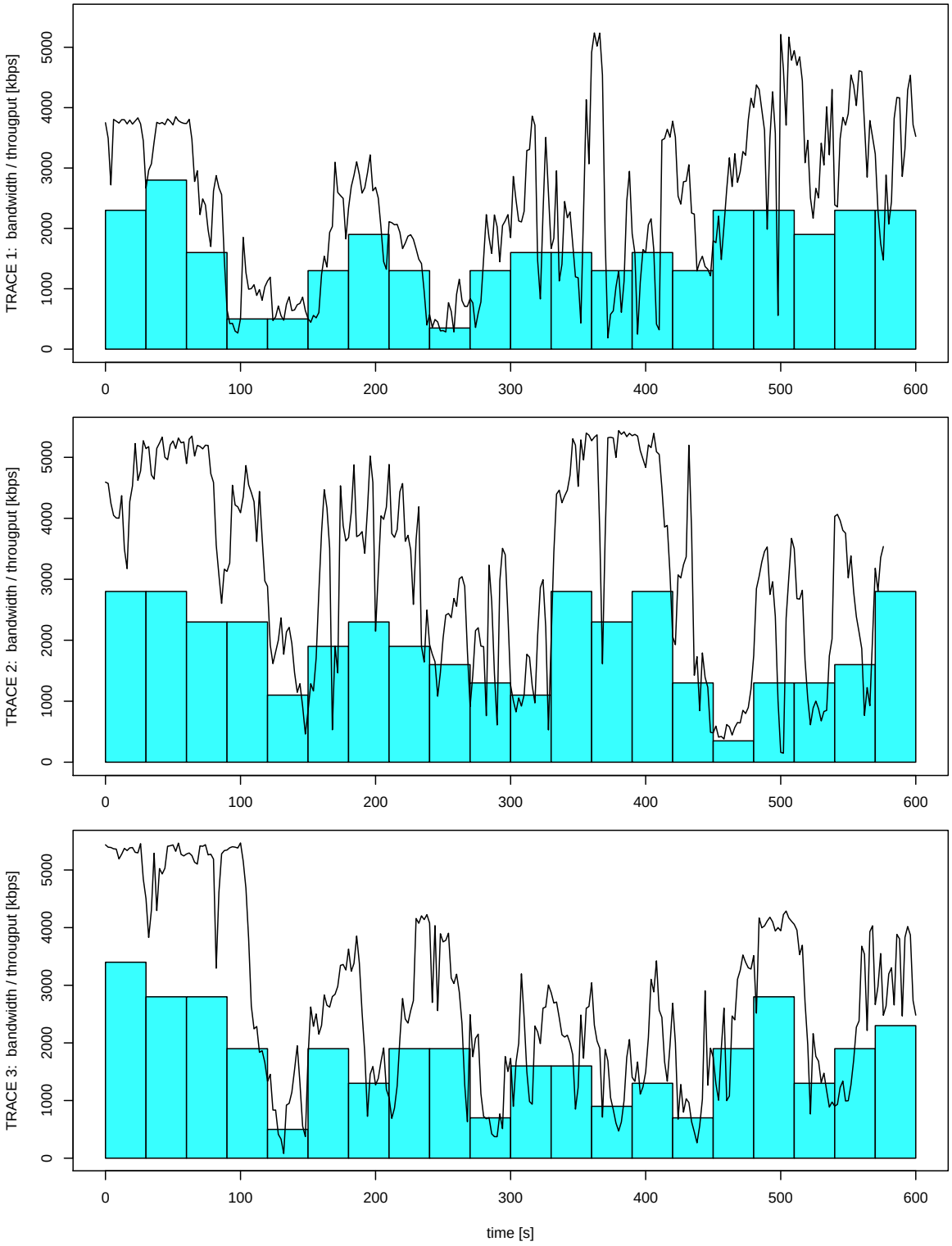



Figure 8.5: Parameter set 1: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 30s, l_{ch} = 40\ kB$, $n_c = 5$, $t_{gap} = 140\ ms$) at a fixed RTT of 150 ms.

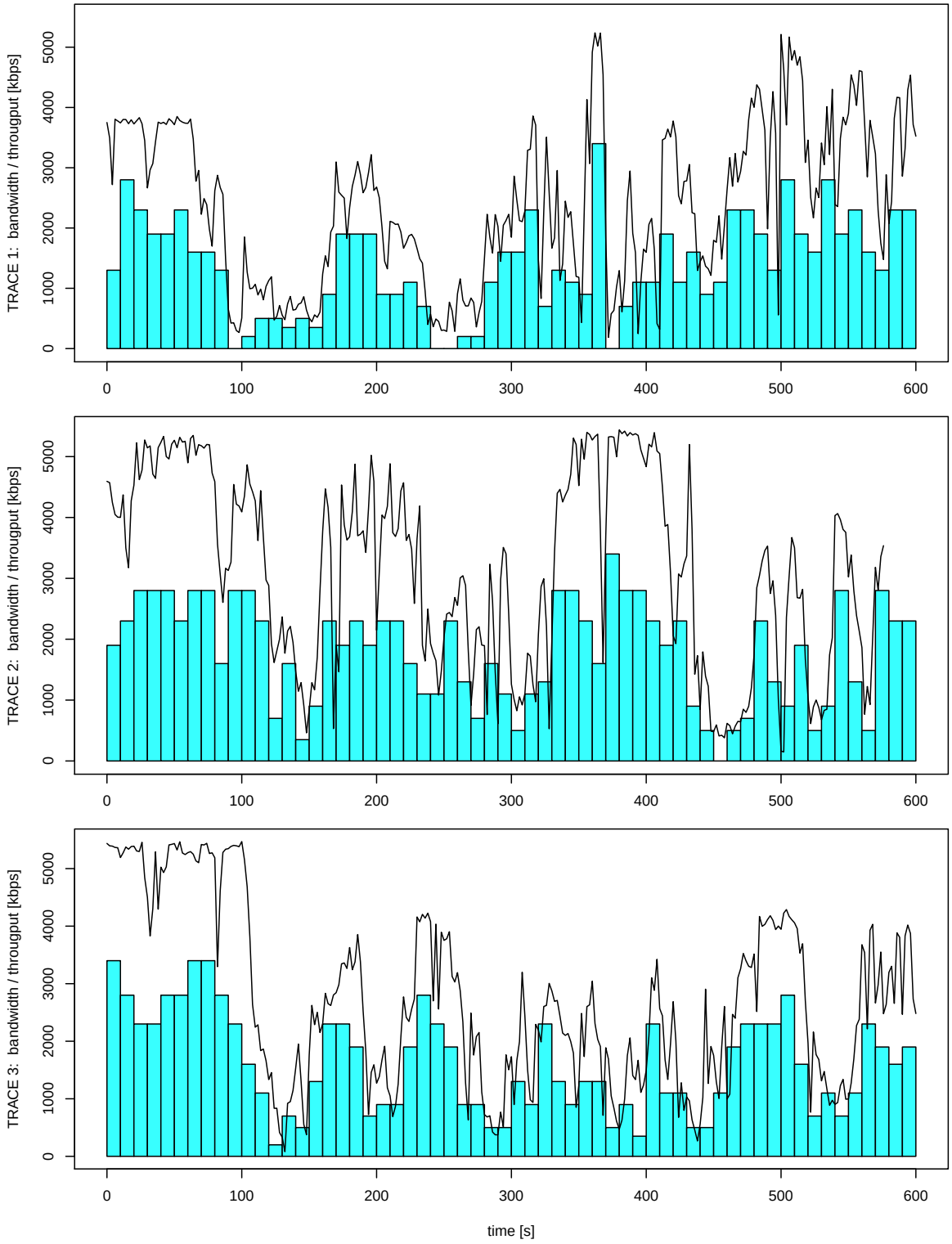



Figure 8.6: Parameter set 2: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 10s, l_{ch} = 80\ kB$, $n_c = 4$, $t_{gap} = 170\ ms$) at a fixed RTT of 150 ms.

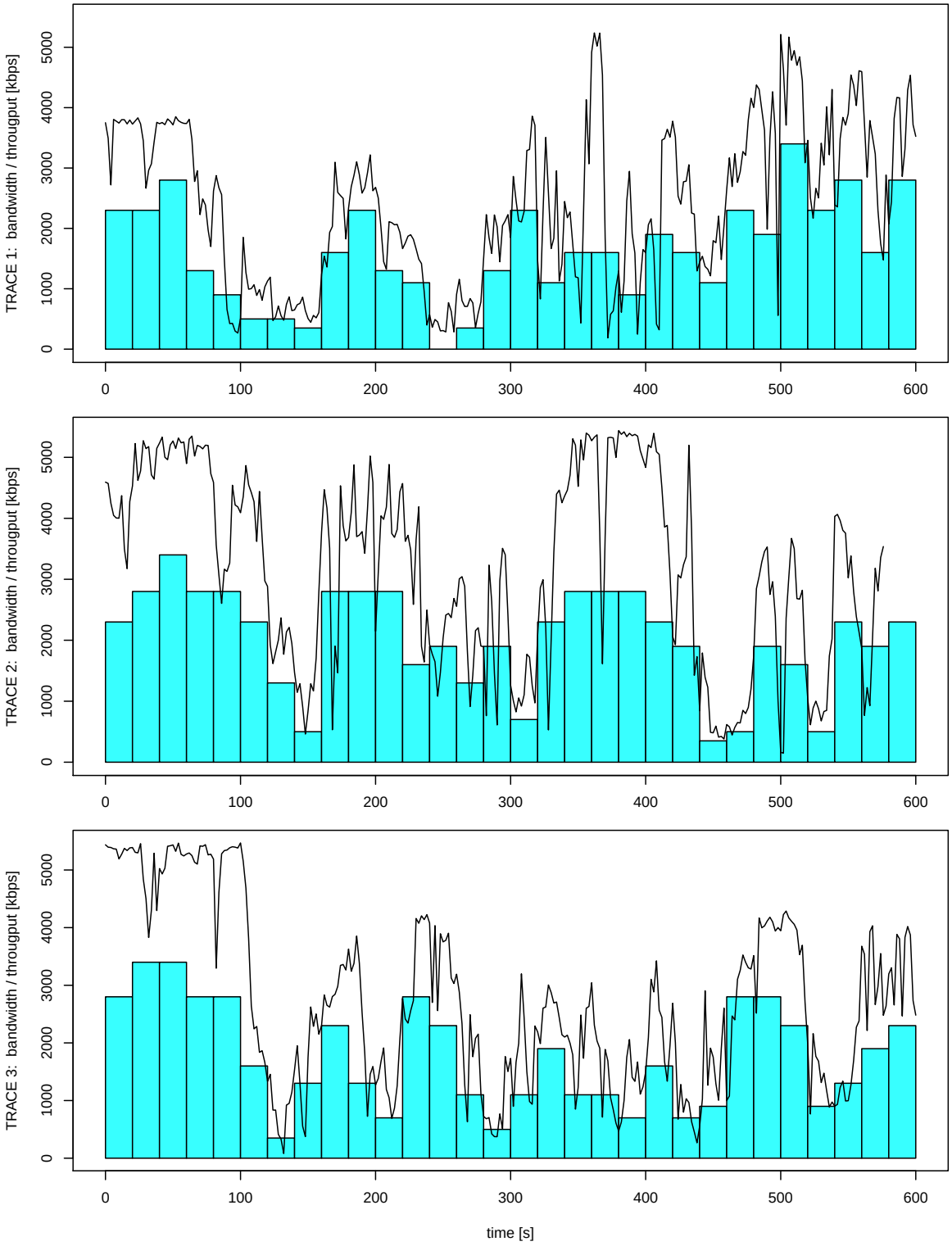



Figure 8.7: Parameter set 2: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 20s, l_{ch} = 80\ kB$, $n_c = 4$, $t_{gap} = 170\ ms$) at a fixed RTT of 150 ms.

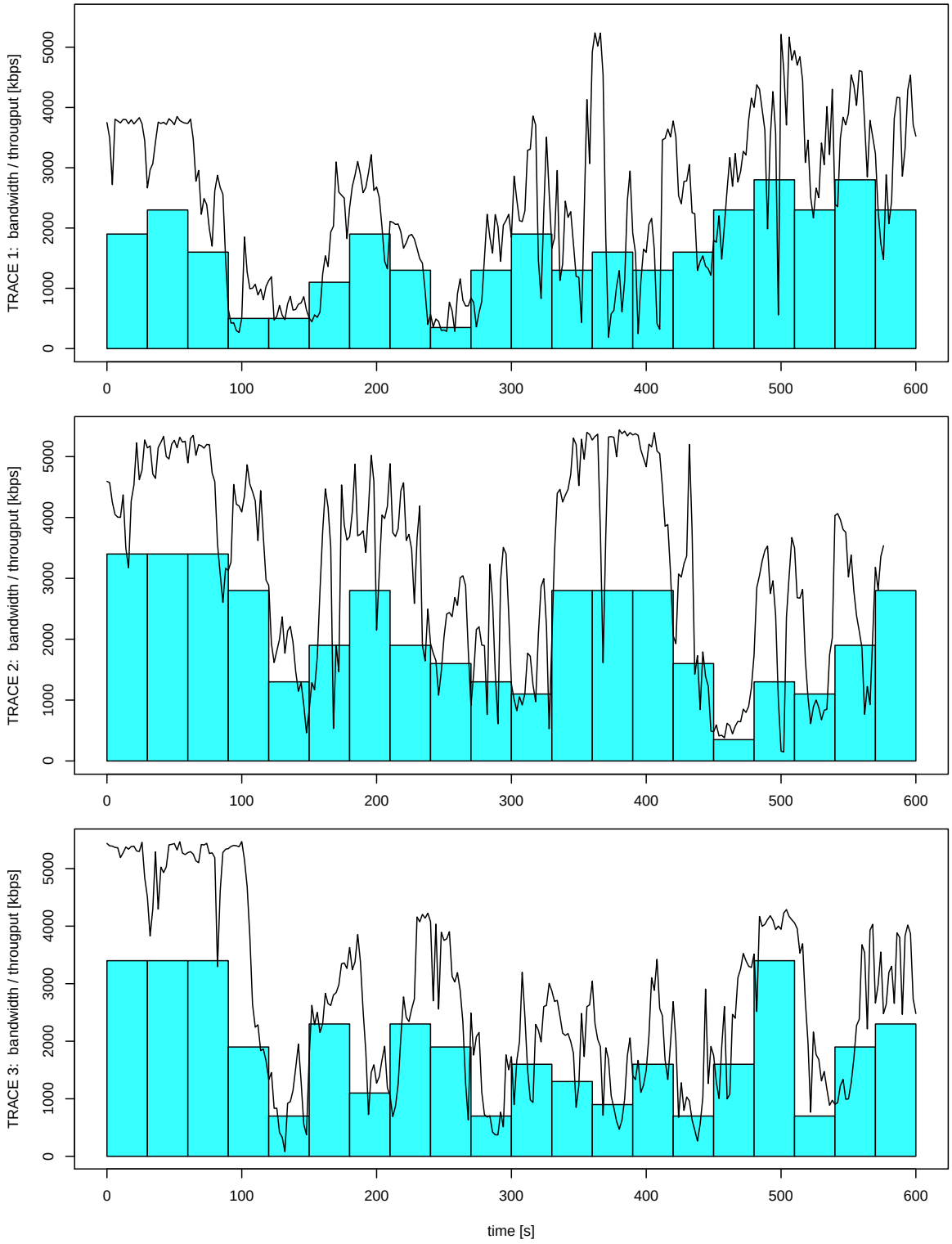



Figure 8.8: Parameter set 2: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 30s, l_{ch} = 80\ kB$, $n_c = 4$, $t_{gap} = 170\ ms$) at a fixed RTT of 150 ms.

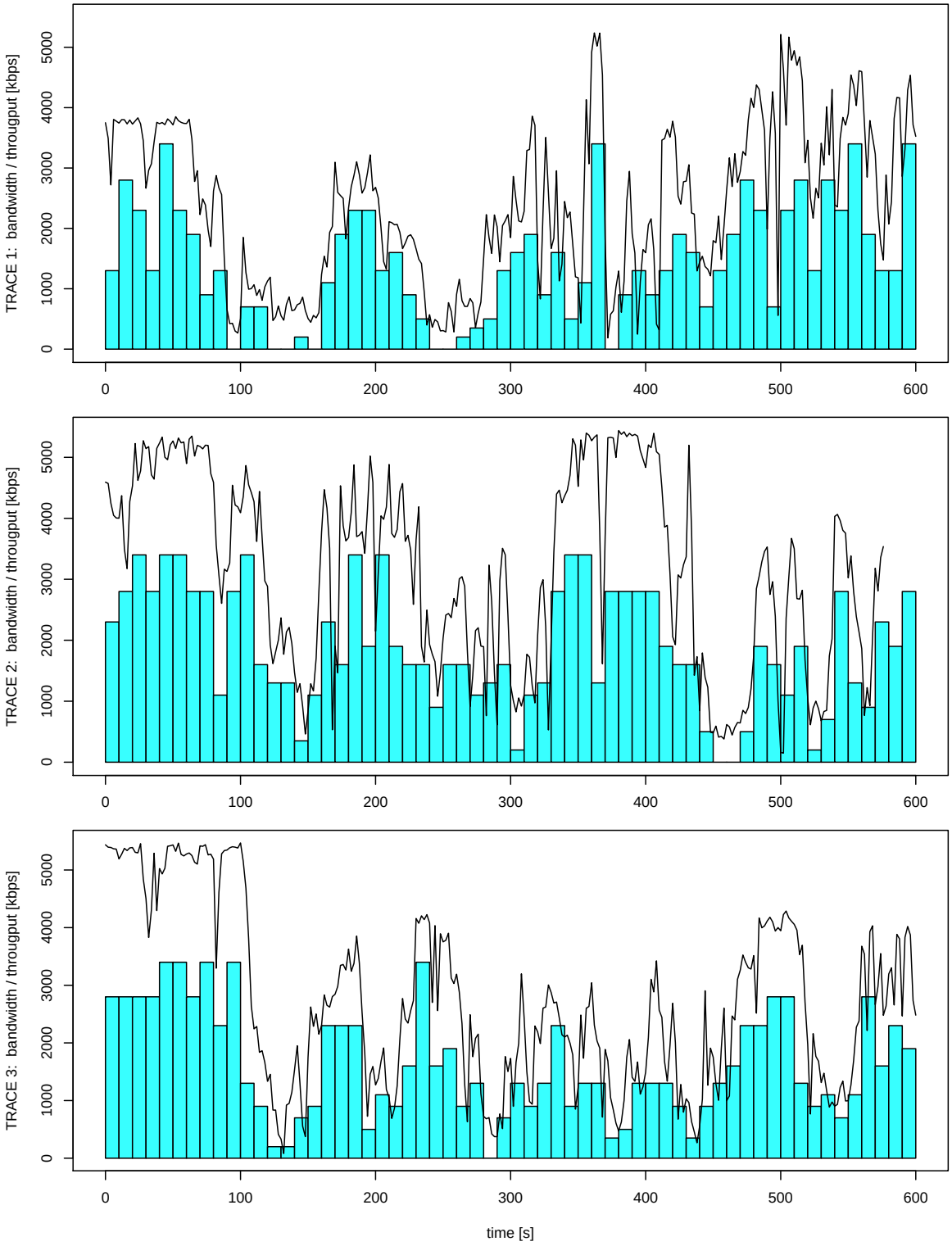



Figure 8.9: Parameter set 3: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 10s, l_{ch} = 160\ kB, n_c = 3, t_{gap} = 180\ ms$) at a fixed RTT of 150 ms.

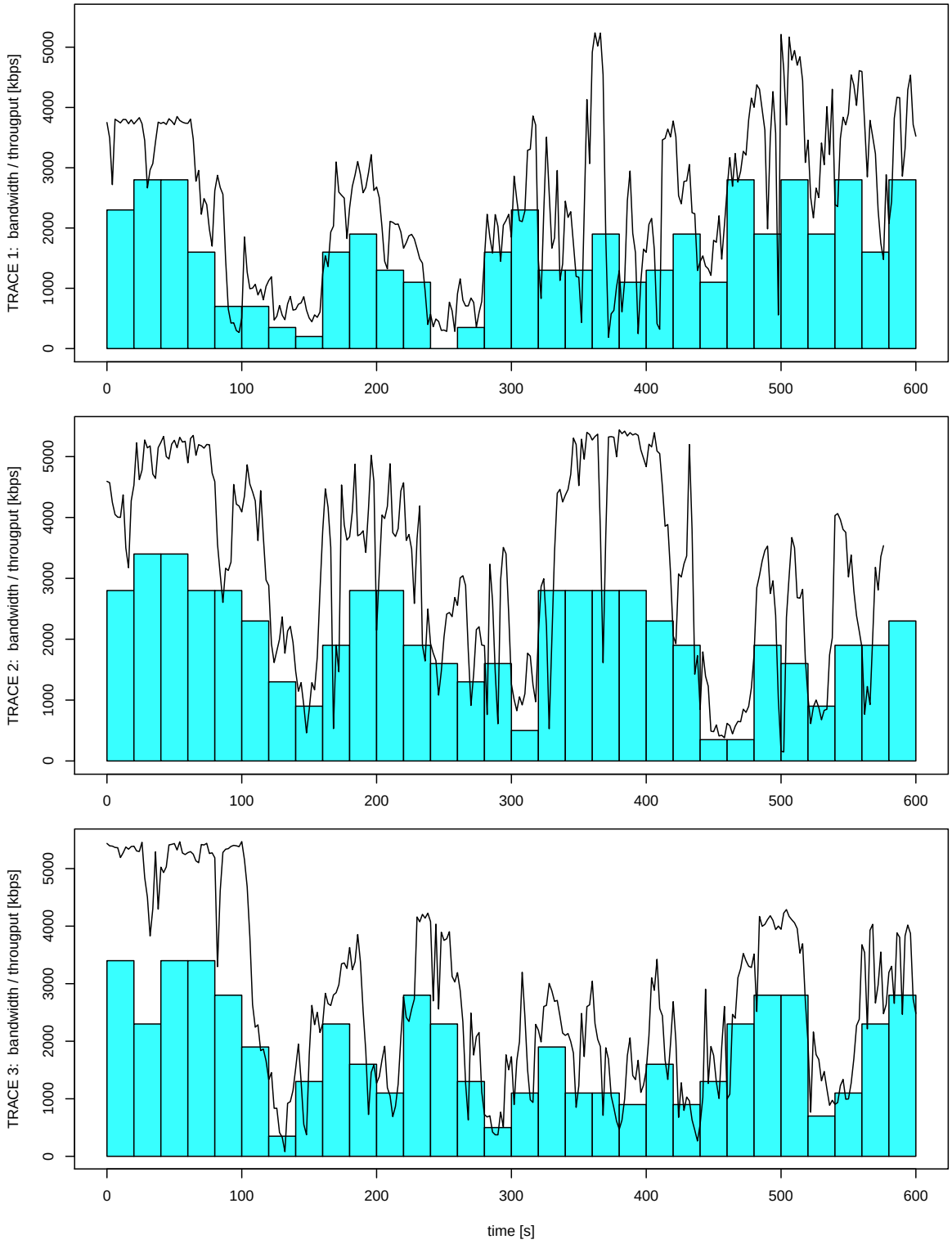



Figure 8.10: Parameter set 3: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 20s, l_{ch} = 160\ kB$, $n_c = 3$, $t_{gap} = 180\ ms$) at a fixed RTT of 150 ms.

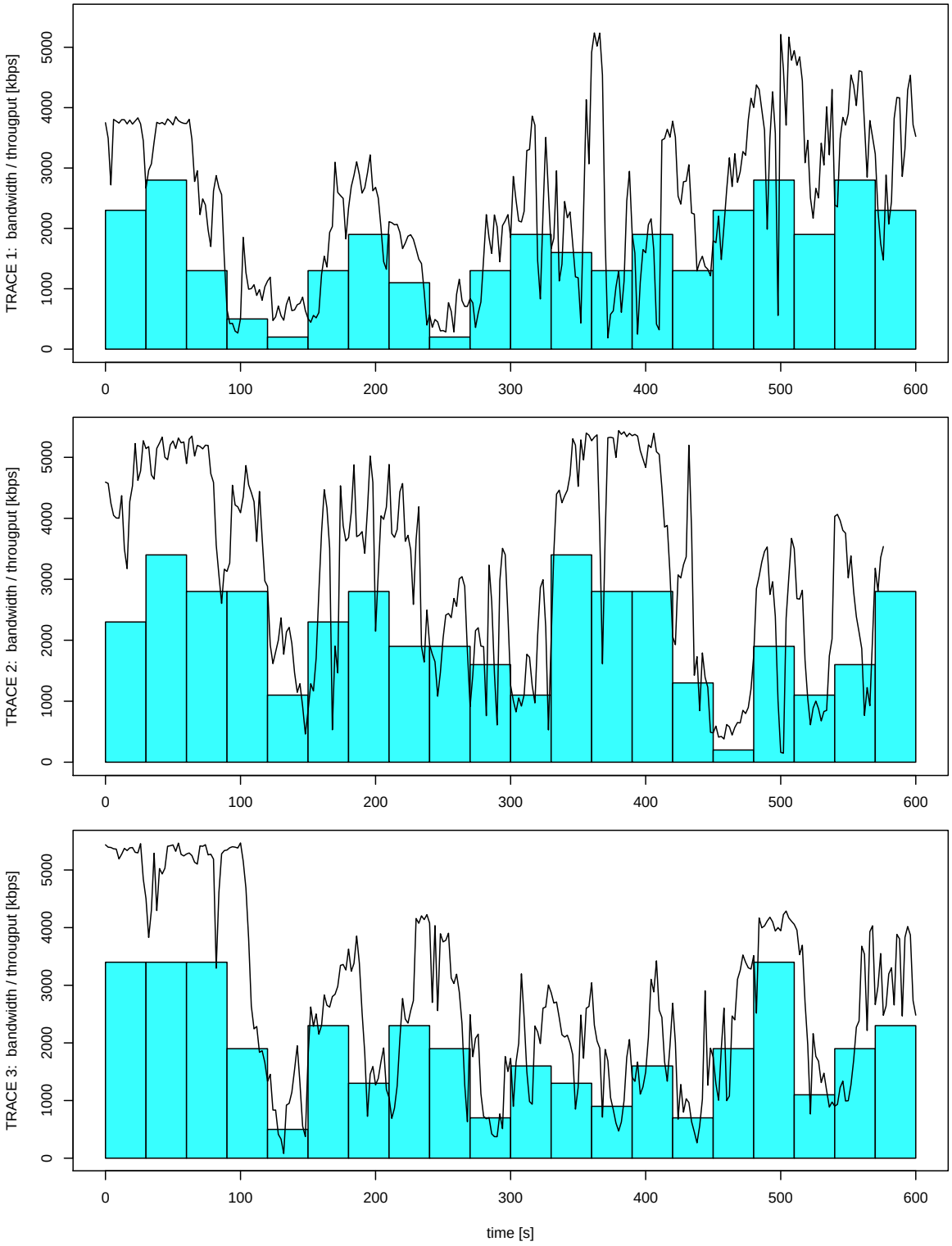



Figure 8.11: Parameter set 3: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 30s, l_{ch} = 160\ kB, n_c = 3, t_{gap} = 180\ ms$) at a fixed RTT of 150 ms.

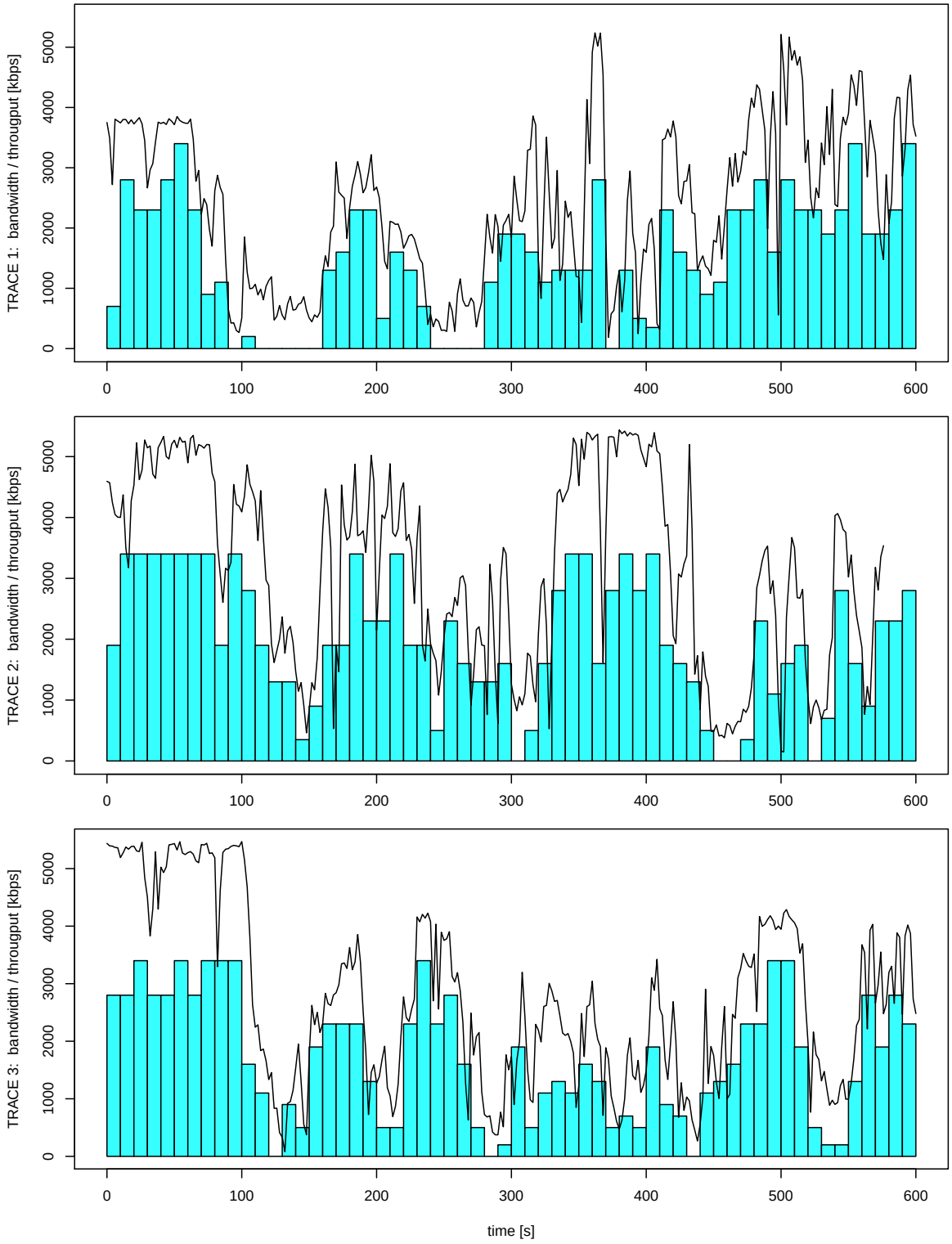



Figure 8.12: Parameter set 4: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 10s, l_{ch} = 160\ kB, n_c = 5, t_{gap} = 210\ ms$) at a fixed RTT of 150 ms.

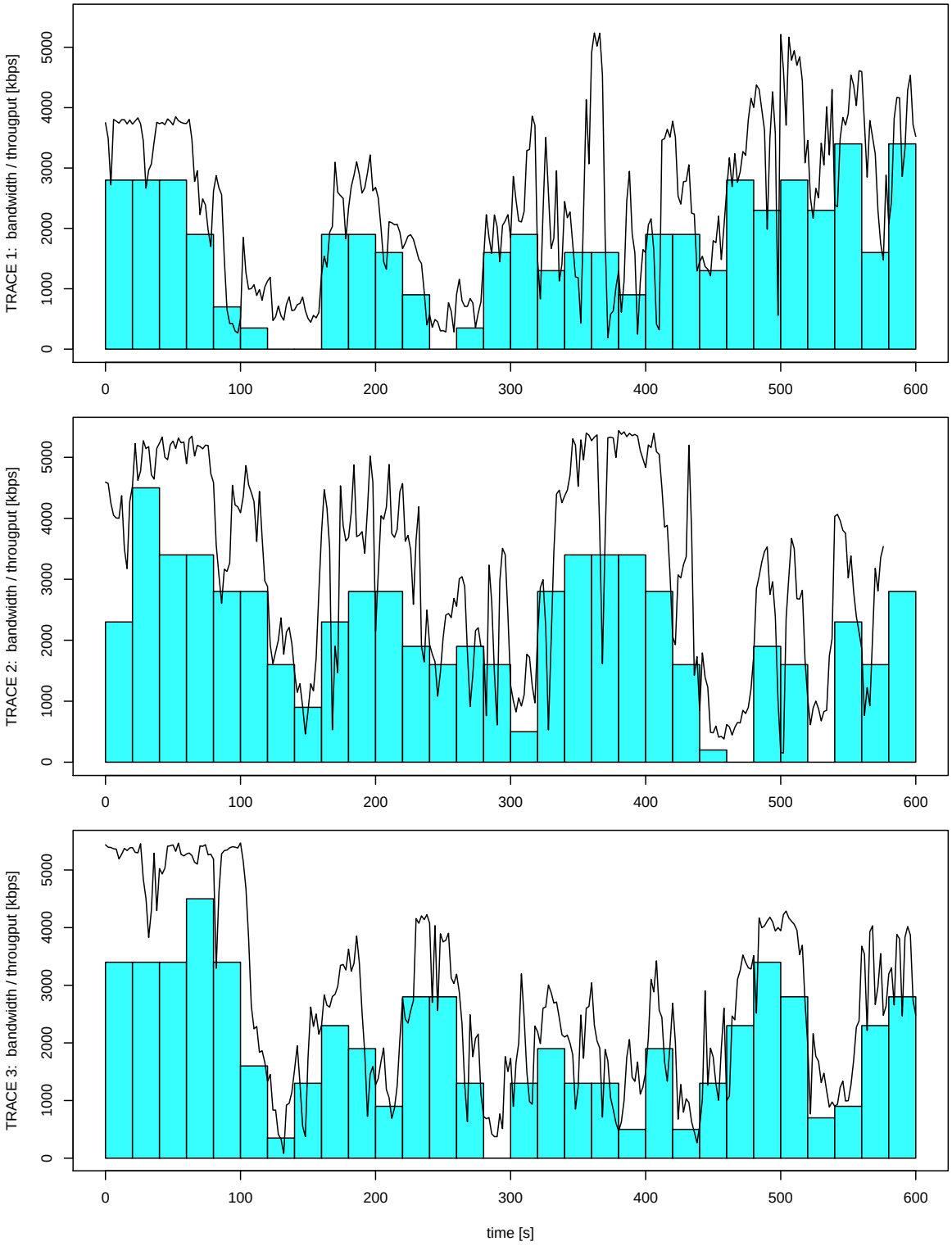



Figure 8.13: Parameter set 4: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 20s, l_{ch} = 160\ kB, n_c = 5, t_{gap} = 210\ ms$) at a fixed RTT of 150 ms.

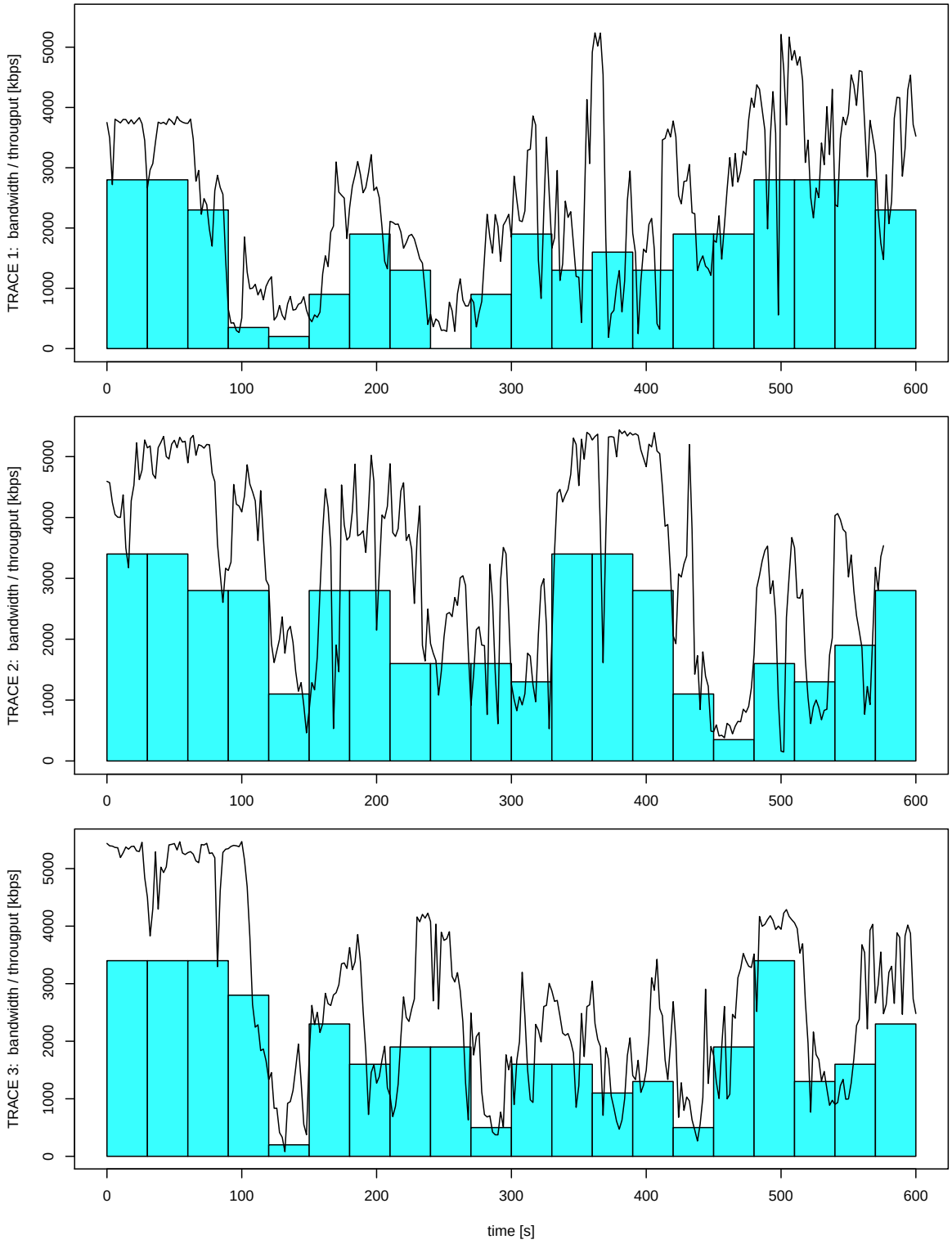



Figure 8.14: Parameter set 4: Network bandwidth (line) and video bit rate (bars) of the request-response streaming system ($t_{frag} = 30s, l_{ch} = 160\ kB, n_c = 5, t_{gap} = 210\ ms$) at a fixed RTT of 150 ms.

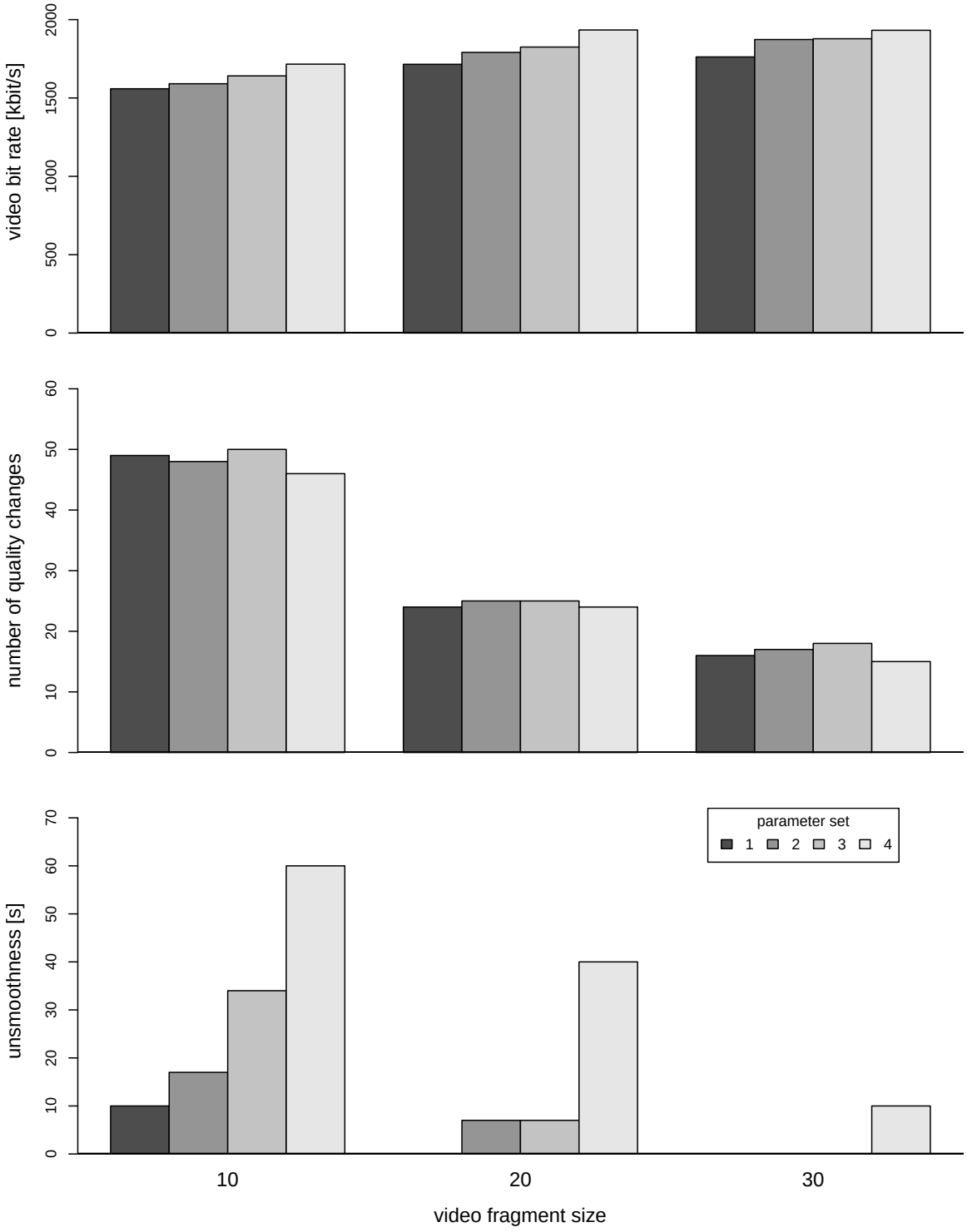



Figure 8.15: Average video bit rate, number of quality switches and unsmoothness of the request-response streaming system for different video fragment sizes ($t_{frag} = 10,\ 20$ *and* $30$ *seconds*) and parameter sets, averaged over all network traces.

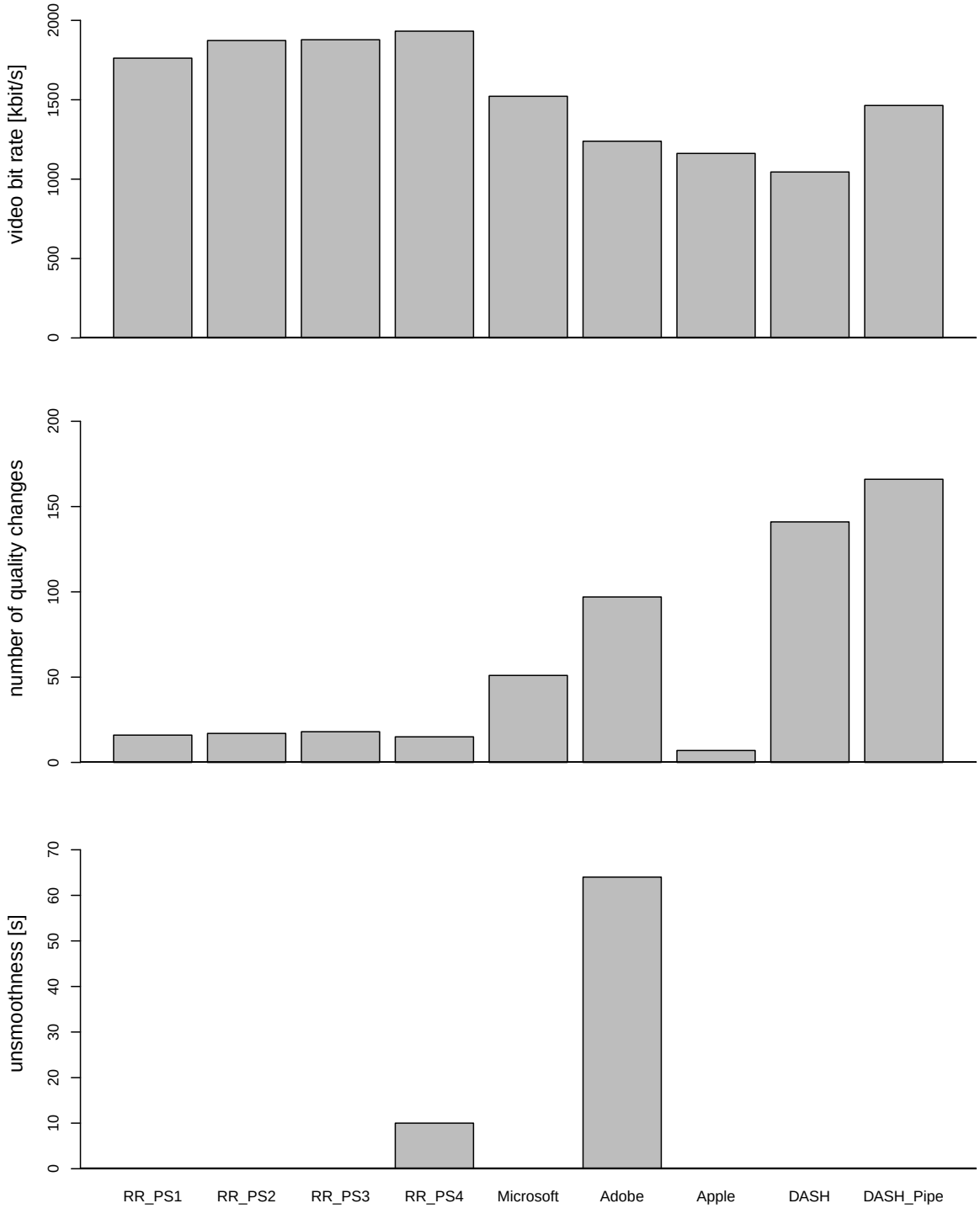



Figure 8.16: Average video bit rate, number of quality switches and unsmoothness of the request-response (RR) streaming system ($t_{frag} = 30$ *seconds*) and adaptive streaming systems evaluated in [66], averaged over all network traces.

**Comparison with Stream Switching:** A direct comparison of a priority-based streaming approach with stream switching is always difficult (see Chapter 4 and Chapter 5). While stream switching is usually based on bandwidth estimation and client-side buffering, priority streaming averages the video quality over a period of time. As a consequence, video fragment sizes in stream switching and priority streaming have to be selected differently.

In stream switching, the video fragment size has to be selected carefully to prevent buffers from draining, because bandwidth estimation errors usually reduce the buffer level (see Section 5.2). For the cellular network scenario, in [66], short video fragments (2 seconds) were used to cope with high bandwidth fluctuations and a client buffer of approximately 30 seconds video data was used to overcome bandwidth shortages. In contrast to stream switching, priority streaming does not use a classical client-side buffer. Nevertheless, priority streaming always buffers one video fragment, which is in this evaluation between 10 and 30 seconds. Another difference is that in case of high bandwidth fluctuations, the performance of stream switching deteriorates with increasing video fragment size, while for priority streaming the opposite is the case. For that reason, comparing the two adaptive streaming approaches with the same video fragment size is not possible. Hence, in the following comparison, a video fragment size of 30 seconds will be considered for priority streaming, because it is similar to the 30 second client-side buffer in the stream switching approaches.

While the test setup is similar to the one in [66], the evaluations were conducted in different test systems. Hence, the results should be interpreted with some caution. Figure 8.16 shows a direct comparison of the performance values of the different streaming approaches. The request-response streaming system with parameter set 1 (RR PS1) and a video fragment size of 30 seconds features an increased video bit rate of approx. 15% compared to the stream switching approaches. In addition, the number of quality switches can be kept low, because priority streaming can utilize large video fragments without performance penalties. The average unsmoothness is zero, except for the parameter set 4, which utilizes large chunk sizes and a high number of concurrent request-response streams. The detailed results of the single tests are shown in Table 8.2.

Table 8.2: Averaged results over all network traces of the request-response streams (RR) with different parameter sets (PS) and the approaches evaluated in [66].

| name | fragment size [s] | video bit rate [kbps] | quality switches | unsmoothness [s] |
|---|---|---|---|---|
| RR PS1 | 10 | 1559 | 49 | 10 |
| RR PS1 | 20 | 1715 | 24 | 0 |
| RR PS1 | 30 | 1762 | 16 | 0 |
| RR PS2 | 10 | 1591 | 48 | 17 |
| RR PS2 | 20 | 1791 | 25 | 7 |
| RR PS2 | 30 | 1873 | 17 | 0 |
| RR PS3 | 10 | 1641 | 50 | 34 |
| RR PS3 | 20 | 1825 | 25 | 7 |
| RR PS3 | 30 | 1878 | 18 | 0 |
| RR PS4 | 10 | 1716 | 46 | 60 |
| RR PS4 | 20 | 1934 | 24 | 40 |
| RR PS4 | 30 | 1932 | 15 | 10 |
| Microsoft | 2 | 1522 | 51 | 0 |
| Adobe | 2 | 1239 | 97 | 64 |
| Apple | 2 | 1162 | 7 | 0 |
| DASH | 2 | 1045 | 141 | 0 |
| DASH Pipelined | 2 | 1464 | 166 | 0 |

CHAPTER

# 9 Conclusion

Internet video streaming has gained a lot of momentum in recent years. Social video portals, like YouTube, became very popular and video-on-demand services emerged with the availability of broadband last-mile networks. Because video streaming generates a considerable amount of the Internet's network traffic [55, 94], it needs to be congestion-aware, to avoid a congestion collapse. TCP is currently the de facto standard protocol for congestion-aware and reliable data transmission. But the Internet offers only a best-effort service. As a result, the available bandwidth may change over the time and constant bit rate video streaming based on TCP is usually only possible with high over-provisioning [104]. Adaptive video streaming based on TCP/HTTP is certainly the key to Internet video streaming and a lot of systems were proposed (see Section 4.5).

While the most criticized property of TCP is its lack of control of the timeliness of delivery, little precaution was taken to prevent video data from arriving too late. The standard way to cope with large arrival time deviations is the use of large buffers, which introduce also high start-up delays. The evaluation of TCP-based adaptive streaming revealed that the streaming performance suffers from difficult network conditions. Usually, the client buffer would drain rapidly, resulting in jerky playback, if no measures like buffer control were deployed.

In contrast to classical buffering at the client, priority streaming averages the video quality over a period of time. The evaluation showed that priority streaming has a significant advantage to other adaptive algorithms based on bandwidth estimation. The buffer usage is well controlled and predictable, resulting in rather low start-up delays even

under congestion.

In this thesis, the basic properties of HTTP streaming were analyzed. By introducing HTTP-based request-response streams, the system parameters of HTTP-based streaming were identified. HTTP-based request-response streams are usually more robust against changing network conditions. This fully client-driven approach is basically state-less on the server-side and allows for fast resume of the streaming session in case of connection resets or stalls. In addition, request-response streams can mitigate the effects of packet loss, which cannot be handled by a single TCP connection. The additional knowledge about the bottleneck router led to an enhanced model for the estimated throughput of the request-response streams. The model performance of the request-response streams indicates that they can utilize the available bandwidth in an efficient manner.

To validate the model, an adaptive streaming system using HTTP-based request-response streams was designed and implemented. The architecture is based on the findings of the evaluation on TCP-based adaptive video streaming. The use of priority streaming proved to be beneficial, resulting in low client buffer usage. Paired with timeout and priority management, it enhances the timeliness of delivery significantly.

The performance of the request-response streams was evaluated in terms of throughput and TCP-friendliness. The throughput can scale with the available bandwidth by increasing the chunk size or the number of concurrent streams. However, the system parameters have to be chosen carefully, in order to avoid congestion or under utilization of the network. The proposed model for the throughput uses knowledge of the bottleneck router to improve the estimation. It shows a good correlation with the experimental results for diverse system configurations and network scenarios. The TCP-friendliness of the request-response streams is dependent on the bottleneck bandwidth and the system parameters. The evaluation showed that there were several combinations of system parameters which exhibit TCP-friendliness. By means of a temporal inter-request gap, it is possible to adjust the fairness of a certain chunk size / number of streams combination to a fair level. The video streaming performance in terms of video quality was evaluated for such a fair system parameter set. The impact of the network's RTT and packet loss on the video quality was investigated. As

expected, the single TCP connection performs better than the request-response streams, if no packet loss is present. However, with increasing packet loss, the performance of the single TCP connection deteriorates rapidly, while the request-response streams can still maintain a reasonable quality.

The ability of the request-response streaming system to adapt to the bandwidth fluctuation of a cellular network was evaluated by using traces of an HSDPA network. By using priority streaming, which averages the available bandwidth over a period of time (10 to 30 seconds), the system was able to cope with the high bandwidth fluctuations. Compared to classical stream switching approaches the average video bit rate could be increased, while still providing a low number of quality switches.

This thesis provided experimental evidence that multiple HTTP-based request-response streams are a good alternative to classical TCP and HTTP streaming. They can efficiently make use of the available bandwidth, while still being less prone to packet loss. Because the request-response streams are based on HTTP, the scalability of the system and the network utilization can be improved. Especially the reuse of existing infrastructure helps to reduce costs, which makes the approach interesting for real world deployments. System parameter sets can be tuned to provide both good throughput and TCP-friendliness, making the approach viable for the use in Internet video streaming.

## Future Work

The results of the evaluation indicate that the request-response streams can make use of additional bandwidth and scale by increasing the chunk size. Because in this thesis only bottleneck bandwidths of 4 and 8 Mbps were evaluated, further investigations beyond a bottleneck bandwidth of 8 Mbps should confirm this hypothesis.

Another finding of this thesis was that also the TCP-friendliness seems to be kept intact when scaling the bandwidth and the chunk size proportionally. This was a rather unexpected result, but is somehow consistent with the assumption that the download duration should be similar, if the bandwidth and the chunk size are increased by the same factor. Evaluating the TCP-friendliness of the request-response streams in different congestion levels for different bottleneck bandwidths would give more insights on this issue.

Currently, the TCP-friendliness of the request-response streams can only be measured. A model of the TCP-friendliness would make measurements unnecessary and therefore be very useful in real-world deployments.

To minimize the factors which could effect the results, this work focuses on request-response streams using fixed parameter sets. An investigation of the performance of dynamic parameter sets and their TCP-friendliness may enable this approach to be deployed for other use cases than adaptive video streaming, because it would increase the possible throughput range where it can work efficiently.

# Bibliography


[1] Big Buck Bunny Movie. `http://www.bigbuckbunny.org`. Last accessed on 2012-04-15.

[2] ISO/IEC 13818-2:2000 - Information technology – Generic coding of moving pictures and associated audio information: Video, 2000.

[3] Digital Video Broadcasting (DVB);Transmission System for Handheld Terminals (DVB-H). ETSI EN 302 304, V1.1.1, Nov 2004.

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

[5] Transparent End-to-end Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH). ETSI TS 126 247, V10.0.0, June 2011.

[6] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSYS 2011)*, pages 157–168, 2011.

[7] John G. Apostolopoulos, Wai-Tian Tan, and Susie J. Wee. Video streaming: Concepts, algorithms, and systems. Technical report, HP Laboratories, 2002.

[8] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of the SIGCOMM '04 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 281–292, 2004.

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

[10] Ali C. Begen. Error Control for IPTV over xDSL Networks. In *Proceedings of the 5th IEEE Consumer Communications and Networking Conference (CCNC 2008)*, pages 632–637, Jan 2008.

[11] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379, 5884, 6093.

[12] Ian Burnett, Rob Koenen, Fernando Pereira, and Rik Van de Walle, editors. *The MPEG-21 Book*. Wiley, 2006.

[13] Shih-Fu Chang and Anthony Vetro. Video Adaptation: Concepts, Technologies, and Open Issues. *Proceedings of the IEEE*, 93(1):148–158, January 2005.

[14] Gao Chen, Yong-dong Zhang, Shou-xun Lin, and Feng Dai. Efficient block size selection for MPEG-2 to H.264 transcoding. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (MULTIMEDIA 2004)*, pages 300–303, 2004.

[15] Luca De Cicco and Saverio Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *Proceedings of the 6th International Conference on HCI in Work and Learning, Life and Leisure: Workgroup Human-Computer Interaction and Usability Engineering (USAB 2010)*, pages 447–464, 2010.

[16] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSYS 2011)*, pages 145–156, 2011.

[17] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871.

[18] S.E. Deering. Host extensions for IP multicasting. RFC 1112 (Standard), August 1989. Updated by RFC 2236.

[19] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC 2007)*, pages 43–56, 2007.

[20] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking (TON)*, 12(6):963–977, 2004.

[21] Michael Eberhard, Riccardo Petrocco, Hermann Hellwagner, and Christian Timmerer. Comparison of Piece-Picking Algorithms for Layered Video Content in Peer-to-Peer Networks. In *Proceedings of the Consumer Communication & Networking Conference 2012*, pages 1–5, Jan 2012.

[22] Wu-Chi Feng, Ming Liu, Brijesh Krishnaswami, and Arvind Prabhudev. A Priority-Based Technique for the Best-Effort Delivery of Stored Video. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference 1999 (MMCN 1999)*, pages 286–300, Jan 1999.

[23] 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.

[24] 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.

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

[26] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard), September 2008.

[27] The Apache Software Foundation. The Apache HTTP Server Project. `http://httpd.apache.org`. Last accessed on 2012-04-15.

[28] The Linux Foundation. netem - Network Emulation. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`. Last accessed on 2012-04-15.

[29] Ladan Gharai. RTP with TCP Friendly Rate Control. Internet Draft draft-ietf-avt-tfrc-profile-10, 2007.

[30] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC 2007)*, pages 15–28, 2007.

[31] Ashvin Goel, Charles Krasic, Kang Li, and Jonathan Walpole. Supporting low latency TCP-based media streams. In *Proceedings of the Tenth IEEE International Workshop on Quality of Service*, pages 193–203, Aug 2002.

[32] Ashvin Goel, Charles Krasic, and Jonathan Walpole. Low-latency adaptive streaming over TCP. *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP 2008)* , 4:20:1–20:20, September 2008.

[33] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[34] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566, July 2006.

[35] Pai-Hsiang Hsiao, H. T. Kung, and Koan-Sin 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 2001)*, pages 199–208, 2001.

[36] Adobe Systems Incorporated. HTTP Dynamic Streaming on the Adobe Flash Platform. `http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf`. Last accessed on 2012-04-15.

[37] ISO/IEC DIS 23009-1.2. Information Technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. July 2011.

[38] ITU - International Telecommunication Union. ITU-R BT.1359-1: Relative Timing of Sound and Vision for Broadcasting. Recommendation, 1998.

[39] ITU - International Telecommunication Union. ITU-T G.114: One-way transmission time. Recommendation, 2003.

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

[41] Hari Kalva. Issues in H.264/MPEG-2 video transcoding. In *Proceedings of the First IEEE Consumer Communications and Networking Conference (CCNC 2004)*, pages 657–659, Jan 2004.

[42] A.M. Kaplan and M. Haenlein. Users of the world, unite! the challenges and opportunities of social media. *Business Horizons*, 53(1):59–68, 2010.

[43] R. P. Karrer, J. Park, and J. Kim. TCP-ROME: Performance and fairness in parallel downloads for Web and real-time multimedia streaming applications. Technical report, Deutsche Telekom Labs, September 2006.

[44] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. Obsoleted by RFC 5321, updated by RFC 5336.

[45] I. Kofler, M. Prangl, R. Kuschnig, and H. Hellwagner. An H.264/SVC-based adaptation proxy on a WiFi router. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2008)*, pages 63–68, May 2008.

[46] Ingo Kofler, Robert Kuschnig, and Hermann Hellwagner. Implications of the ISO Base Media File Format on adaptive HTTP streaming of H.264/SVC. In *Proceedings of the 9th IEEE Consumer Communications and Networking Conference (CCNC)*, page 5, jan 2012.

[47] Ingo Kofler, Christian Timmerer, Hermann Hellwagner, Andreas Hutter, and Francesc Sanahuja. Efficient MPEG-21-based Adaptation Decision-Taking for Scalable Multimedia Content. In *Proceedings of the 14th SPIE Annual Electronic Imaging Conference – Multimedia Computing and Networking (MMCN 2007)*, Jan 2007.

[48] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340 (Proposed Standard), March 2006. Updated by RFCs 5595, 5596.

[49] Charles Krasic, Kang Li, and Jonathan Walpole. The Case for Streaming Multimedia with TCP. In *Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS 2001)*, pages 213–218, 2001.

[50] Charles Krasic, Jonathan Walpole, and Wu-Chi 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 2003)*, pages 112–121, 2003.

[51] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. An Evaluation of TCP-based Rate-Control Algorithms for Adaptive Internet Streaming of H.264/SVC. In *Proceedings of the First Annual ACM Conference on Multimedia Systems (MMSYS 2010)*, February 2010.

[52] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response Streams. In *Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC 2010)*, page 5, Jan 2010.

[53] Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. Evaluation of HTTP-based request-response streams for internet video streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSYS 2011)*, pages 245–256, Feb 2011.

[54] Robert Kuschnig, Ingo Kofler, Michael Ransburg, and Hermann Hellwagner. Design options and comparison of in-network H.264/SVC adaptation. *Journal of Visual Communication and Image Representation*, 19(8):529–542, December 2008.

[55] Neal Leavitt. Network-Usage Changes Push Internet Traffic to the Edge. *Computer*, 43:13–15, 2010.

[56] Bo Li and Jiangchuan Liu. Multirate video multicast over the Internet: an overview. *IEEE Network*, 17(1):24–29, jan 2003.

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

[58] Huadong Ma and Kang G. Shin. Multicast Video-on-Demand services. *ACM SIGCOMM - Computer Communication Review*, 32(1):31–43, January 2002.

[59] Kevin J. Ma, Radim Bartos, Swapnil Bhatia, and Raj Nair. Mobile video delivery with HTTP. *Communications Magazine, IEEE*, 49(4):166–175, April 2011.

[60] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *ACM SIGCOMM - Computer Communication Review*, 27(3):67–82, 1997.

[61] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM '96 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 117–130, 1996.

[62] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010.

[63] D. Miras, M. Bateman, and S. Bhatti. Fairness of High-Speed TCP Stacks. In *Proceedings of the International Conference on Advanced Information Networking and Applications (IANA)*, pages 84–92, 2008.

[64] Move Networks. Move Adaptive Stream - Product Sheet. `http://www.movenetworks.com`. Last accessed on 2012-04-15.

[65] Christopher Müller and Christian Timmerer. A test-bed for the Dynamic Adaptive Streaming over HTTP featuring session mobility. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSYS 2011)*, pages 271–276, 2011.

[66] Christopher Müller, Stefan Lederer, and Christian Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proceedings of the*

*Fourth Annual ACM SIGMM Workshop on Mobile Video (MoVid12)*, pages 37–42, feb 2012.

[67] A. Nafaa, T. Taleb, and L. Murphy. Forward error correction strategies for media streaming over wireless networks. *IEEE Communications Magazine*, 46(1):72–79, Jan 2008.

[68] Jens-Rainer Ohm. Advances in Scalable Video Coding. *Proceedings of the IEEE*, 93(1):42–56, Jan 2005.

[69] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP throughput: a simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 303–314, 1998.

[70] R. Pantos. HTTP Live Streaming. Internet Draft draft-pantos-http-live-streaming-07, 2011.

[71] F. Pereira and I. Burnett. Universal multimedia experiences for tomorrow. *IEEE Signal Processing Magazine*, 20(2):63–73, March 2003.

[72] Fernando Pereira, John Smith, and Anthony Vetro. Special Section on MPEG-21. *IEEE Transactions on Multimedia*, 7(3), June 2005.

[73] C. Perkins. RTP and the Datagram Congestion Control Protocol (DCCP). RFC 5762 (Proposed Standard), April 2010.

[74] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[75] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[76] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093.

[77] Martin Prangl, Ingo Kofler, and Hermann Hellwagner. Towards QoS Improvements of TCP-Based Media Delivery. In *Proceedings of the Fourth International Conference on Networking and Services (ICNS 2008)*, pages 188–193, 2008.

[78] Zhang Qian, Guo Quji, Ni Qiang, Zhu Wenwu, and Zhang Ya-Qin. Sender-adaptive and receiver-driven layered multicast for scalable video over the Internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(4):482–495, April 2005.

[79] Iain E. G. Richardson. *Video Codec Design: Developing Image and Video Compression Systems*. Wiley, 202.

[80] Haakon Riiser, Pal Halvorsen, Carsten Griwodz, and Dag Johansen. Low Overhead Container Format for Adaptive Streaming. In *Proceedings of ACM Multimedia Systems (MMSYS 2010)*, pages 193–198, Feb 2010.

[81] Mohit Saxena, Umang Sharan, and Sonia 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 2008)*, pages 39–44, 2008.

[82] T. Schierl, S. Johansen, A. Perkis, and T. Wiegand. Rateless scalable video coding for overlay multisource streaming in MANETs. *Journal of Visual Communication and Image Representation*, 19(8):500 – 507, 2008.

[83] T. Schierl and S. Wenger. Signaling Media Decoding Dependency in the Session Description Protocol (SDP). RFC 5583 (Proposed Standard), July 2009.

[84] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard), July 2003.

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

[86] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, April 1998.

[87] H. Schwarz, D. Marpe, and T. Wiegand. Analysis of Hierarchical B Pictures and MCTF. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2006)*, July 2006.

[88] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[89] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 315–323, 1998.

[90] Daniel Stenberg. cURL and libcurl - cURL groks URLs. `http://curl.haxx.se/`. Last accessed on 2012-04-15.

[91] Thomas Stockhammer. Dynamic adaptive streaming over HTTP – standards and design principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSYS 2011)*, pages 133–144, Feb 2011.

[92] Thomas Stockhammer, Guenther Liebl, and Michael Walter. Optimized H.264/AVC-based bit stream switching for mobile video streaming. *EURASIP J. Appl. Signal Process.*, 2006:1–19, 2006.

[93] Yeping Su, Jun Xin, A. Vetro, and Huifang Sun. Efficient MPEG-2 to H.264/AVC intra transcoding in transform-domain. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2005)*, volume 2, pages 1234–1237, May 2005.

[94] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2010-2015. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`. Last accessed on 2012-04-15.

[95] S. Takeuchi, H. Koga, K. Iida, Y. Kadobayashi, and S. Yamaguchi. Performance evaluations of DCCP for bursty traffic in real-time applications. In *Proceedings of the Symposium on Applications and the Internet*, pages 142–149, Jan 2005.

[96] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, pages 1–12, April 2006.

[97] S. Thulasidasan, Wu chun Feng, and M.K. Gardner. Optimizing GridFTP through dynamic right-sizing. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 14–23, June 2003.

[98] C. Timmerer, S. Devillers, and M. Ransburg, editors. *ISO/IEC 21000-7:2007 Part 7: Digital Item Adaptation 2nd Edition*. International Standardization Organization, 2007.

[99] Sunand Tullimas, Thinh Nguyen, Rich Edgecomb, and Sen-ching Cheung. Multimedia streaming using multiple TCP connections. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 4:12:1–12:20, May 2008.

[100] Bobby Vandalore, Wu chi Feng, Raj Jain, and Sonia Fahmy. A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia. *Real-Time Imaging*, 7(3):221 – 235, 2001.

[101] A. Vetro, C. Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: an overview. *IEEE Signal Processing Magazine*, 20(2):18–29, March 2003.

[102] Anthony Vetro, Charilaos Christopoulos, and Touradj Ebrahimi. Special Issue on Universal Multimedia Access. *IEEE Signal Processing Magazine*, 20(2), March 2003.

[103] Anthony Vetro and Christian Timmerer. Digital Item Adaptation: Overview of Standardization and Research Activities. *IEEE Transactions on Multimedia*, 7(3):418–426, June 2005.

[104] Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. *ACM Transactions on Multimedia Computing, Communications and Applications*, 4(2):16:1–16:22, 2008.

[105] Y. Wang, 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.

[106] S. Wenger, M.M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP Payload Format for H.264 Video. RFC 3984, February 2005.

[107] S. Wenger, Y. Wang, and T. Schierl. Transport and Signaling of SVC in IP Networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1164–1173, September 2007.

[108] S. Wenger, Y.-K. Wang, T. Schierl, and A. Eleftheriadis. RTP Payload Format for Scalable Video Coding. RFC 6190 (Proposed Standard), May 2011.

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

[110] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[111] XSL Transformations (XSLT) 1.0. W3C Recommendation, 16 November 1999. URL: `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[112] Jun Yao, Salil Kanhere, Imran Hossain, and Mahbub Hassan. Empirical Evaluation of HTTP Adaptive Streaming under Vehicular Mobility. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *Proceedings of the 10th International IFIP TC 6 Conference on Networking (NETWORKING 2011)*, volume 6640 of *Lecture Notes in Computer Science*, pages 92–105. 2011.

[113] Esma Yildirim, Dengpan Yin, and Tevfik 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 2009)*, pages 21–30, 2009.

[114] A. Zambelli. IIS Smooth Streaming Technical Overview. Technical report, Microsoft Corporation, March 2009.

[115] Michael Zink, Carsten Griwodz, Jens Schmitt, and Ralf Steinmetz. Scalable TCP-friendly Video Distribution for Heterogeneous Clients. In Ragunathan Rajkumar, editor, *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2003)*, pages 102–113, 2003.

# List of Figures