Adaptation Algorithms for HTTP-Based Video Streaming

vorgelegt von
Dipl.-Ing.
Konstantin Miller
geb. in Murmansk

von der Fakultät IV – Elektrotechnik und Informatik –
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr.-Ing. Thomas Sikora
Gutachter:	Prof. Dr.-Ing. Adam Wolisz
Gutachterin:	Prof. Klara Nahrstedt, Ph.D.
Gutachter:	Prof. Dr.-Ing. Carsten Griwodz
Gutachter:	Assoc. Prof. Priv.-Doz. Dr. Christian Timmerer

Tag der wissenschaftlichen Aussprache: 16. November 2016

Berlin 2016

Acknowledgments

This thesis would not have been possible without the help of many people. First of all, I would like to thank my advisor Prof. Adam Wolisz for his encouragement and support. Being a passionate researcher, he loves to get to the bottom of things, and has mastered the art of asking the right questions. Being initially more inclined towards mathematics and theory, I was guided by Prof. Wolisz into the spirit of engineering research, which means to make mathematics relevant to engineering problems through a precise modeling and a thorough validation of conclusions.

I want to thank Emanuele Quacchio for the internship opportunity at the Advanced System Technology group at STMicroelectronics, Milano, Italy. This internship was the start of my work on adaptive streaming technologies that finally has led to this thesis. Further, I would like to thank Prof. Giuseppe Caire for being my host and mentor during my internship at the University of Southern California, CA, USA. His willingness to dive into mathematical problems, and his vast expertise in the area of wireless communication, have made working with him a stimulating experience. I am particularly grateful to Prof. Tobias Harks for his support and fruitful discussions.

I believe that without intensive discussions an efficient and creative intellectual work is hardly possible. Thus, I would like to acknowledge all the interlocutors who have inspired me, by explicitly using the pronoun "we" instead of "I" throughout the thesis.

Last but not least, I thank my family for their love, support and encouragement in this challenging endeavor I took up. This thesis would not have been possible without them.

Abstract

Ever since the invention of the cinematography, there has been a growing demand for high-quality video content. Recently, the broad availability of high-speed wireless Internet access, complemented by the pervasiveness of mobile, computationally powerful devices with high-resolution screens, have made the video delivery over the open Internet the technology of choice for both video on demand and live streaming services. Due to the best-effort nature of the Internet, however, ensuring a high quality of experience is challenging. A state-of-the-art approach to address this challenge is adaptive streaming, designed to continuously adjust the characteristics of the streamed media to dynamically varying network conditions, leading to a smoother viewing experience with less playback interruptions and a more efficient utilization of the available network resources. Despite the ongoing efforts, however, recent studies suggest that the challenge has not yet been successfully resolved. One of the open issues is the design of efficient adaptation algorithms, that are among the primary factors determining the overall performance of a streaming service. In this thesis, I present several contributions to this area of research, that are outlined in the following.

In order to cope with the wireless traffic increase expected over the next years, it will be necessary to increase the density of the deployed wireless infrastructure. In my first contribution, I focus on a simultaneous delivery of a *large number of unicast video on demand streams in a dense wireless network*. I jointly consider the problem of wireless transmission scheduling and video quality selection, and develop a distributed approach based on control theory. The conducted performance evaluation shows that the presented approach is able to serve an up to twice as large number of users completely without interruptions, as compared to a baseline approach. Simultaneously, it allows to reduce the number of quality transitions by up to 50%, without reducing the average video quality. In addition, the unfairness among the individual streaming sessions is reduced by up to a factor of 4.

Even though the majority of the video content being streamed over the Internet is video on demand, the amount of live streaming is growing rapidly. In my second contribution, I focus on a particularly challenging use case of *low-delay live streaming*. I develop a novel adaptation algorithm that is leveraging throughput predictions to provide a high quality of experience over wireless links, with a latency bound on the order of a few seconds. It heuristically maximizes the average video quality at an operating point defined by the live latency, amount of playback interruptions, and number of quality transitions. A comparative evaluation reveals that at the individual operating points,

the developed algorithm provides an average video quality which is by up to a factor of 3 higher than the quality achieved by the baseline approach. Furthermore, it is able to reach a broader range of operating points, and can thus be more flexibly adapted to the user profile and service provider requirements.

In my third contribution, I develop a *universal adaptation algorithm for video on demand*, that can operate over a broad range of network conditions, and that has a flexible configuration that can be adjusted to the particular service and user requirements. It uses the playback buffer level information and the past throughput information to meet its adaptation decisions. It does not rely on a cooperation with the network nor on cross-layer information, and is therefore suitable for a standalone deployment in any network environment, and on a broad range of platforms. Moreover, it minimizes the start-up delay, which is particularly important for services, where users tend to frequently start new video sessions. I evaluate the approach against a baseline and against an *omniscient client that computes optimal adaptation trajectories* by solving a series of optimization problems. The evaluation reveals that the proposed algorithm allows to efficiently avoid playback interruptions, provides a smooth viewing experience by avoiding excessive video quality fluctuations, achieves a high level of network resource utilization, and provides a fair resource allocation in a multi-user environment. In particular, in the network environment used for the evaluation, the developed algorithm achieves an average video bit rate which is by up to 35% higher than that of the baseline approach, and within up to 85% of the optimum, with an up to an order of magnitude smaller total duration of interruptions. It is worth mentioning that the omniscient client developed in the course of this work can not only serve as a reliable benchmark for streaming clients but also allows to evaluate the influence of various media and network properties on the achievable streaming performance.

Last but not least, based on my experience with implementing streaming client prototypes and simulation models, I develop a *streaming client architecture* that is modular, extendible, and platform-independent, and efficiently supports distributed operation of the individual functional blocks.

Zusammenfassung

Seit der Erfindung der Kinematographie steigt der Bedarf, hochwertige Videoinhalte jederzeit und überall abrufen zu können, stetig an. Die allgegenwärtige Verfügbarkeit von drahtlosem Hochgeschwindigkeitszugang zum Internet, ergänzt durch die Verbreitung von mobilen internetfähigen Endgeräten mit hoher Rechenkraft und hochauflösenden Bildschirmen, machte das Internet zur Technologie der Wahl sowohl für Video-on-Demand- als auch für Livestreaming-Dienste. Allerdings macht die minimalistische Dienstgütezusicherung im Internet das Erreichen einer hohen Erlebnisqualität zu einer Herausforderung. Ein dem Stand der Technik entsprechender Ansatz ist adaptives Streaming, das die technischen Charakteristiken der übertragenen Inhalte kontinuierlich an die Netzwerkdynamik anpasst und damit für ein gleichmäßigeres Betrachtungserlebnis mit weniger Wiedergabeunterbrechungen und einer effizienteren Netzwerkauslastung sorgt. Allerdings belegen Studien, dass diese Herausforderung, trotz der anhaltenden Anstrengungen, noch nicht erfolgreich bewältigt ist. Eins der ungelösten Probleme ist der Entwurf von performanten Adaptationsalgorithmen – einem der wichtigsten bestimmenden Faktoren, der die Gesamtqualität eines Streamingdienstes beeinflusst. In der vorliegenden Dissertation stelle ich mehrere Beiträge zu diesem Forschungsfeld vor, die ich im Folgenden umreiße.

Um den in den kommenden Jahren erwarteten Anstieg des drahtlosen Datenverkehrs zu bewältigen, muss die vorhandene Funknetzeinfrastruktur verdichtet werden. In meinem ersten Beitrag befasse ich mich mit der gleichzeitigen Übertragung einer *großen Zahl von Video-on-Demand-Strömen in einem dichten Drahtlosnetzwerk*. Ich betrachte gemeinsam die beiden Probleme der Übertragungssteuerung im Funknetz und der Videoqualitätsadaptation und entwickele einen verteilten auf Kontrolltheorie basierenden Ansatz. Die durchgeführte Leistungsbewertung zeigt, dass der vorgestellte Ansatz im Vergleich zum verwendeten Basisverfahren eine bis zu zweimal so große Nutzermenge ohne Wiedergabeunterbrechungen bedienen kann. Gleichzeitig erlaubt er, die Anzahl der Qualitätsübergänge um bis zu 50% zu reduzieren, ohne die durchschnittliche Videoqualität zu beeinträchtigen. Außerdem wird die Unfairness zwischen den einzelnen Videoströmen um bis zu viermal reduziert.

Obwohl Video-on-Demand den Großteil der über das Internet übertragenen Videoinhalte ausmacht, steigt der Anteil der Liveinhalte rapide. In meinem zweiten Beitrag widme ich mich dem besonders anspruchsvollen Fall des *Livestreamings mit niedriger Verzögerung*. Der entwickelte neuartige Adaptationsalgorithmus setzt Durchsatzvorhersagen ein, um selbst über Drahtlosverbindungen eine hohe Erlebnisqualität mit auf

wenige Sekunden beschränkter Latenz zu erreichen. Der Algorithmus maximiert heuristisch die durchschnittliche Videoqualität am Arbeitspunkt, der durch die Latenz, Dauer der Wiedergabeunterbrechungen und Anzahl der Qualitätsübergänge definiert ist. Eine vergleichende Leistungsbewertung zeigt, dass der entwickelte Algorithmus in den einzelnen Arbeitspunkten eine bis zu dreimal höhere durchschnittliche Qualität als der Basisansatz erreichen kann. Außerdem ist er in der Lage, eine breitere Spanne von Arbeitspunkten anzusteuern und kann daher flexibler an den Nutzerprofil und an die Anforderungen des Dienstanbieters angepasst werden.

In meinem dritten Beitrag entwickle ich einen *universellen Adaptationsalgorithmus für Video-on-Demand*, der in einem breiten Spektrum von Netzwerkumgebungen einsetzbar ist und flexibel an die speziellen Anforderungen einzelner Dienste und Nutzer angepasst werden kann. Der entwickelte Ansatz verwendet für seine Entscheidungen den Abspielbuffer-Füllstand sowie den gemessenen Durchsatz. Er ist weder auf eine Kooperation mit dem Netzwerk noch auf Informationen aus unteren Protokollschichten angewiesen und daher für den Einsatz in beliebigen Netzwerkumgebungen und auf einer Vielzahl von Plattformen geeignet. Darüber hinaus minimiert er die Startverzögerung, was insbesondere für Dienste, bei denen die Nutzer zum häufigen Starten neuer Videoströme tendieren, wichtig ist. Ich evaluiere den Ansatz im Vergleich zu einem Basisansatz und einem *allwissenden Clienten*, der durch das Lösen einer Reihe von Optimierungsproblemen optimale Adaptationstrajektorien berechnet. Die Leistungsbewertung zeigt, dass der entwickelte Algorithmus effizient Wiedergabeunterbrechungen vermeidet, eine gleichmäßige Betrachtungsqualität durch das Vermeiden übermäßiger Qualitätsschwankungen erzielt, eine hohe Netzwerkauslastung erreicht und eine faire Resourcenverteilung in Mehrnutzerumgebungen herstellt. Der entwickelte Algorithmus erreicht in den getesteten Netzwerkumgebungen eine um bis zu 35% höhere durchschnittliche Medienbitrate als der Basisansatz bzw. bis zu 85% des Optimums, mit einer um bis zu einer Größenordnung kleineren Gesamtdauer der Wiedergabeunterbrechungen. Es ist erwähnenswert, dass der im Rahmen dieser Arbeit entwickelte allwissende Client nicht nur einen zuverlässigen Bewertungsmaßstab darstellt, sondern es auch erlaubt, den Einfluss verschiedener Medien- und Netzwerkcharakteristiken auf die erreichbare Streamingleistung zu untersuchen.

Schließlich habe ich, basierend auf den beim Implementieren von Streamingclient-Prototypen und -Simulationsmodellen gesammelten Erfahrungen, eine *Streamingclient-Architektur* entwickelt, die modular, erweiterbar und plattformunabhängig ist und einen verteilten Betrieb der einzelnen Funktionsblöcke ermöglicht.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans have always been predominantly visual creatures. "Bread and circuses", that's how a Roman satirical poet ironically described the main cares of the Roman populace 2000 years ago. Not much has changed since that time. In fact, since the beginning of the 20st century, when cinematography made it possible to detach the visual observation of an event from its actual happening, technological progress has gone a long way to enable people to consume video content at any time and any place, selecting from a sheer unlimited supply.

After the cinematography, in the middle of the 20st century, television achieved a further step by bringing the content directly to people's homes. In addition, it enabled *live* transmission, that is, watching an event *as it happens*. At that time, however, the supply consisted exclusively of a fixed, prescheduled menu of content distributed through a handful of over-the-air broadcast television stations, soon followed by cable and satellite distribution companies. Users' choices were therefore limited to switching to another station.

The next step took place in the late seventies, when Videocassette Recorders (VCR's) became affordable for the home use, and videocassette capacity reached several hours. For the first time, consumers obtained the possibility to time-shift video programming, untethering it from the schedules determined by the broadcasters. In addition, VCR's broadened the amount of the available content by enabling consumption of movies through the sale or rental of prerecorded tapes. Following the demand, cable companies also began offering Video on Demand (VoD) services that enabled viewers to watch broadcast or cable network programming or movies *on demand* for a limited time.

And then the Internet has arrived.

The ubiquitous availability of broadband Internet access, complemented by the advances in media compression technologies and miniaturization, as well as the increasing

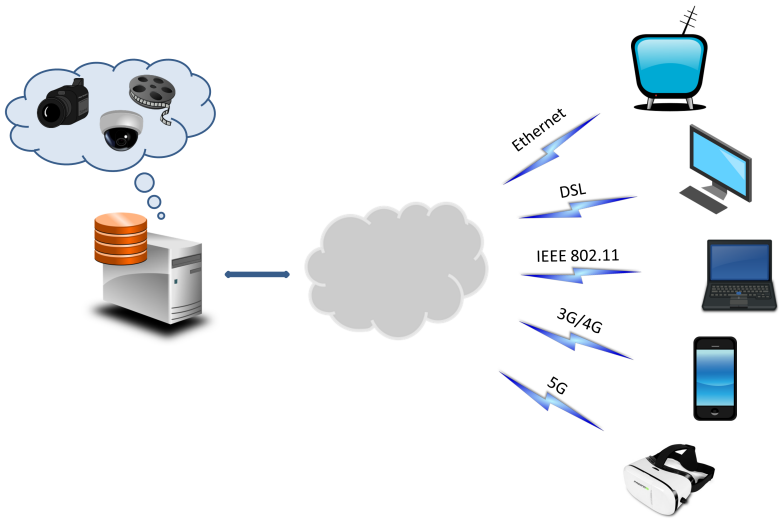

Figure 1.1: Consumers are engaging with digital media across an increasing number of platforms and network technologies.

processing power of electronic devices, produced a new mindset: watch what I want, when I want, and where I want. A multitude of devices provide access to a vast sea of video content at any time and location: smartphones, tablets, PC's, game consoles, and, of course, Hybrid TV sets – the next generation of the TV's that have been connected to the Internet and are now offering a multitude of interactive applications (see Figure 1.1 for an illustration). Wearable devices such as "smart watches" and "smart glasses" are rapidly gaining popularity, and will take the digital media landscape further to a whole new level. All these devices empower the user to watch their favorite content on the best screen available at a particular moment, and not at the behest of the content provider. Moreover, people are increasingly using multiple types of devices to access the content, often simultaneously. Thus, in 2013 for the first time multi-platform users became the majority of the U.S. audience [41].

In addition to TV-like services, applications such as surveillance, telepresence, Virtual Reality (VR), or tele-immersion are leveraging the Internet as the communication platform to an ever increasing extent [218]. All in all, according to the forecasts, video traffic will be 79% of all consumer Internet traffic in 2018, up from 66% in 2013 [37]. Notably, this trend is being accompanied by a shift towards the usage of wireless and mobile networks. In 2013, wired devices still accounted for the majority of Internet traffic at 56%. The status quo, however, is rapidly changing. Traffic from wireless and mobile devices will exceed traffic from wired devices by 2018, accounting for 61% of the total Internet traffic.

The Internet, however, was not designed to stream video. First of all, traditionally, a broadcaster exclusively used a communication channel, such as a radio frequency, to broadcast to everyone within reach. Digital Video Broadcasting (DVB) networks (terrestrial, cable, or satellite) can be named as an example. In contrast, with Internet-based streaming or, as it is often called, Over-the-Top (OTT) streaming a separate,

unicast, data stream is transmitted to every single receiver, which is much less efficient, and the medium is shared among many users.

Moreover, the Internet is a so-called best-effort network that does not provide any Quality of Service (QoS) guarantees. It is thus a very challenging medium for multimedia distribution since, prior to starting the streaming session, it is not known which network conditions one can expect.

Thus, supporting such an enormous amount of video traffic with an appropriate Quality of Experience (QoE), which is a concept introduced with the goal to evaluate human perception of multimedia content in an objective manner [90], places a huge burden on state-of-the-art communication networks technology, and requires novel solutions in the areas of content distribution [121], wireless and mobile networking, and video streaming.

One example is the enormous heterogeneity of the network throughput. On the one side, there still exist locations with Internet access speeds of tens of kilobits per second, e.g., in rural areas, on highways, in trains, or in the underground, especially under unfavorable link conditions that can arise, e.g., due to mobility. On the other side there are Digital Subscriber Line (DSL), cable, or Ethernet access links that can provide tens or hundreds of megabits per second. This heterogeneity makes it impossible to transmit the same media to each user. Once encoded into a certain media representation, a piece of video content poses certain requirements on the network path used for its transmission. A mismatch between these requirements and the network properties can lead to a severe degradation of the QoE. One typical consequence are playback interruptions that arise when a piece of content is not received in time for its playback due to a low network throughput. Another one is a long initial delay, during which the streaming client prebuffers enough data to sustain sudden throughput drops and link outages.

As a consequence, we lately have been observing a period of high interest in adaptive streaming technologies that are able to continuously adjust the characteristics of the streamed media to dynamically varying network conditions, leading to a smoother viewing experience with less playback interruptions and a more efficient utilization of the available network resources. In particular one technology has become the *de facto* standard for Internet streaming: HTTP-Based Adaptive Streaming (HAS) [183].

HAS has several advantages, as compared to the traditional OTT streaming technologies such as the Real-Time Transport Protocol (RTP)/Real-Time Control Protocol (RTCP)/Real-Time Streaming Protocol (RTSP) suit. HAS uses Hypertext Transfer Protocol (HTTP), which was developed in the early days of the World Wide Web as the main application layer protocol used by a client device to fetch web pages from a server [15, 79]. By using HTTP, and thus standard web servers, HAS forgoes the necessity to maintain specialized video servers and pay for their licenses, thus reducing the operating costs. Using HTTP, HAS is leveraging an ubiquitous and highly optimized delivery infrastructure, originally created for the web traffic, which includes, e.g., Content Delivery Networks (CDNs), caches, and proxies. In addition, HTTP is typically allowed to traverse middleboxes, such as Network Address Translation (NAT) devices and firewalls. HAS has good scalability properties due to the stateless nature of HTTP, and the client-based control logic. The reliable transmission, provided by the underlying Transmission Control Protocol (TCP), enables usage of efficient video compression

technologies that are particularly sensitive to packet losses. The usage of HTTP/TCP also simplifies service and application development due to the TCP's built-in features, such as congestion and flow control, that otherwise would have to be implemented by the application itself. Last but not least, one of the enablers of the success of HAS was the open standard MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [50, 180].

Among the core factors determining the performance of an adaptive streaming client is its adaptation logic, which continuously adjusts the media representation of the streamed video content to the dynamically varying network conditions. It typically pursues several partially conflicting goals, frequently including: providing the best possible QoE or the one satisfying the active Service-Level Agreements (SLA's), minimizing costs, maximizing fairness, satisfying latency constraints (in the case of live streaming), along with others.

The design of adaptation mechanisms is extremely challenging due to the often complex stochastic dynamics of the network conditions. Especially on wireless links, users are exposed to interference, cross-traffic, and fading effects, leading to continuously fluctuating QoS characteristics. These effects become even more severe when the users are mobile. The tight latency requirements in the case of low-delay live streaming make the task even harder.

To further complicate matters, expressing QoE in a way that facilitates objective measurements is itself an open research question. It must take into account human perception and cognitive processing – phenomena influenced by hard to measure factors. The number of factors influencing QoE is immense, and many of them have a high level of subjectivity that results in extremely complex modeling [162, 217].

Recent studies suggest that the challenges arising when delivering high-quality video content over the Internet has not yet been successfully addressed. In 2013, around 26.9% of streaming sessions on the Internet experienced playback interruption due to rebuffering, 43.3% were impacted by low resolution, and 4.8% failed to start altogether [43].

Consequently, in my work I have focused on the problem of developing efficient adaptation techniques for several application domains. In particular, I have considered the following three deployment scenarios:

- supporting a large number of parallel VoD unicast streaming sessions in dense wireless networks by performing joint transmission scheduling and video quality adaptation, in a decentralized way,
- low-delay live streaming in wireless networks, and
- VoD streaming targeting a broad spectrum of network environments.

In addition, I have developed a scheme based on solving a series of optimization problem that computes optimal adaptation trajectories for VoD from the perspective of an omniscient client which has the full knowledge of the future throughput.

Moreover, the iterative implementation of the developed adaptation approaches and their integration into streaming clients and network simulators has led to the design of a universal streaming client architecture that is modular and flexible, and thus can be used both for live streaming and VoD, is able to accommodate cross-layer and context information, and can be implemented in a platform-independent way.

Finally, I would like to mention two aspects that are not in the scope of the present work: multi-view streaming and interactive streaming. On the one hand, the presented approaches are agnostic of the media format. Consequently, they can be used to deliver 2D, 3D, 360°, or VR content. In fact, the algorithm TOBASCO which is presented in Chapter 7 has been used in an MPEG-DASH-based 3D streaming prototype jointly developed by STMicroelectronics and Fraunhofer HHI in 2011. On the other hand, however, transmitting multi-view content such as 360°or VR as if it were single-view content is inefficient due to the lack of spatial differentiation. While the spatial relationship descriptors in the MPEG-DASH standard allow for optimization by splitting the content in tiles and only transmitting subsets of tiles that are in the current view of the user, this is not considered in the present work. Also, interactive streaming, such as video conferencing or tele-immersion, is out of scope, since HTTP-based client-driven mechanisms are not designed to provide a low enough delay of 100 ms and less, as required by such services.

In the following, I outline the contributions in more details.

Joint Transmission Scheduling and Video Quality Selection in Dense Wireless Networks

It is well understood that the current trend of cellular technology (e.g., Long-Term Evolution (LTE) [174]) cannot cope with the traffic increase caused by the multitude of new video services, unless the density of the deployed wireless infrastructure is increased correspondingly. One solution candidate are very dense small cell networks (multiple nested tiers of smaller and smaller cells, possibly operating at higher and higher carrier frequencies) [26]. If supplied with sufficient storage capacity, they can also help reducing the load on the backhaul (i.e., the part of the network connecting the access network to the Internet), which have recently become a bottleneck in wireless networks [63].

Consequently, in my first contribution I have focused on the highly timely problem of efficiently supporting a large number of parallel unicast video streaming sessions in a dense wireless network. To achieve the best possible performance, the problems of wireless transmission scheduling and video quality selection are considered jointly – which is known to lead to the best performance, but which is also known to be challenging due to the distributed nature of the problem, as well as the different time scales of the individual subproblems.

The innovation of the proposed scheme, called JINGER (**J**oint Schedul**ing** and Quality Sel**e**ction in Dense Wi**r**eless Networks), is the usage of a control-theoretic framework which leverages Proportional-Integral-Derivative (PID) controller theory. The strength of the PID controllers lies in their analytical tractability, complemented by the ability to stabilize a dynamic system in the presence of model uncertainties (that is, the system parameters are not completely known and might be time-varying) and disturbances (unknown, potentially random, inputs to the system).

In the studied case, a PID controller is used to stabilize users' playback buffers around certain target values, in the presence of dynamically changing network conditions due to users arrival and departures, mobility and fading effects. I show how the usage of an anti-windup technique and heuristic strategies leads to a decentralized enforcement of

the desired overall control behaviour. In the developed scheme the wireless transmission scheduling is performed by a centralized network controller, while the video quality is selected by each client individually and asynchronously (on its own time scale) – leading however jointly to a stabilization of the playback buffer level dynamics, and heuristically maximizing the quality of experience.

The approach has been extensively evaluated by simulations in different deployment scenarios, such as long-term users with low user churn, short-term users with high user churn, and a mix of short-term and long-term users. It has been compared to a baseline approach, with respect to all the important factors influencing the quality of experience, such as rebuffering, average video distortion, and the number of quality level transitions. The baseline approach represents currently deployed systems, where the transmission scheduling is unaware of the application's requirements, and each streaming client is individually trying to optimize its own video quality. I have observed that JINGER consistentlly and significantly outperforms the baseline scheme. It has been shown to serve an up to twice as large number of users completely without rebuffering, which is the most important factor defining the performance. Simultaneously, my approach allows to reduce the number of quality transitions by up to 50%, without reducing the average video distortion. In addition – as a side effect, the unfairness among the individual streaming sessions is reduced by up to a factor of 4.

The approach is directly implementable. The client-side mechanisms use only locally available information and the mechanisms at the network controller require a minimum of communication that can be piggy-backed with the video segment requests. Candidate platforms include both cellular and Wireless Local Area Network (WLAN) technologies, or a combination of both, that allow multiple simultaneous Access Point (AP) associations, combined with a centralized network controller performing the coordination, and extended by the capability to simultaneously receive multiple downstream transmissions. The study thus paves the way for the development of highly scalable video streaming solutions that are able to deliver a large number of concurrent high bit rate video streams to wirelessly connected users.

Prediction-Based Low-Delay Live Streaming

Although currently the majority of the video content streamed over the Internet is VoD, the amount of live streaming is growing rapidly [187]. While current live streaming services can exhibit a latency of several tens of seconds, *low-delay* streaming refers to live streaming with a particularly low upper bound on the latency: a few seconds or less. Such a requirement is desirable for use cases such as the transmissions of sports events. Moreover, a low latency is absolutely necessary in the case of video conferencing and online gaming, where active participants have latency requirements on the order of hundreds of milliseconds [92], while permanently or temporarily passive participants may be served with a delay of a few seconds.

HAS, however, has been primarily developed to replace the progressive download of VoD content and therefore its application to low-delay streaming has received little attention in the research community. In many studies, typical buffer sizes used for the design and evaluation of HAS-based clients are on the order of tens of seconds. The

capability of the HAS approach to efficiently stream low-delay content, especially in wireless networks, is still an open question.

Consequently, in my second contribution, I have demonstrated that efficient HAS-based low-delay live streaming is possible by leveraging short-term TCP throughput predictions over multiple time scales, from 1 to 10 seconds, along with estimations of the relative prediction error distribution. I have designed a novel prediction-based algorithm called LOLYPOP (**Lo**w-**L**atenc**y** **P**redicti**o**n-Based Ada**p**tation) that supports QoE-based adaptation with a transport latency on the order of a few seconds.

The approach introduced in LOLYPOP jointly considers four QoE components: the live latency, the number of playback interruptions, the number of quality transitions, and the average video quality. Its goal is to maximize the average video quality as a function of the operating point defined by the other three components. The operating point is controlled by three input parameters: the target live latency, an upper bound on the number of quality transitions, and a parameter controlling the number of playback interruptions. Thus, LOLYPOP provides configurable QoE that can be adjusted to the nature of the video, the user context and preferences, or the service provider's business model.

At the core of LOLYPOP is an estimation of download success probabilities for the individual segments. To obtain these estimations, LOLYPOP leverages predictions of throughput distributions, computed from a time series prediction and an error estimation. I have evaluated several time series prediction methods using TCP throughput traces collected in IEEE 802.11 WLANs, including public hotspots (indoor and outdoor), campus hotspots, and access points in residential environments. I have observed, somewhat surprisingly, that taking the average over the previous T seconds as a prediction for the next T seconds provides the best prediction accuracy among the considered methods for all considered time scales. That is, taking into account the trend does not help to reduce the prediction error.

I have implemented a prototype of the algorithm and have evaluated it against FESTIVE [98], a well-known adaptation algorithm from the literature. I have limited the transport latency to 3 seconds, while still using a segment duration of 2 seconds. I have observed that LOLYPOP is able to reach a broad range of operating points and thus can be flexibly adapted to the user profile or service provider requirements. Furthermore, I have observed that at the individual operating points, LOLYPOP provides an average video quality which is by up to a factor of 3 higher than the quality achieved by the baseline approach.

VoD Adaptation Algorithm

My third contribution is an adaptation algorithm called TOBASCO (**T**hresh**o**ld-**B**ased **A**daptation **Sc**heme for **o**n-Demand Streaming), that is designed for buffer sizes typical for VoD, but may also be used for live streaming with moderate latency requirements. The proposed scheme uses both the buffer level information and past throughput information to meet its adaptation decisions. It has a flexible configuration and thus can be deployed in various network environments and with different user profiles. It can, e.g., be configured to provide a consistent video quality with a low amount of quality tran-

sitions, or it can allow for a higher number of quality transitions in order to maximize the video quality.

TOBASCO has been implemented as a plugin for the open source multimedia player VLC, and evaluated in an emulated WLAN against a commercial HAS implementation by Microsoft, as well as against optimal adaptation trajectories computed using the full knowledge about future throughput dynamics. The evaluation has shown that TOBASCO allows to efficiently avoid playback interruptions, provides a smooth viewing experience by avoiding excessive video quality fluctuations, achieves a high level of network resource utilization, and provides a fair resource allocation in a multi-user environment. Moreover, it minimizes start-up delays, which is particularly important for services, where users tend to frequently start new video sessions. In particular, in the network environment used for the evaluation, the developed algorithm achieves an average video bit rate which is by up to 35% higher than that of the baseline approach, and within up to 85% of the optimum, with an up to an order of magnitude smaller rebuffering duration.

Omniscient VoD Streaming Client

One open issue regarding the performance evaluation of adaptation algorithms is the lack of widely accepted benchmarks. To tackle this issue, I have developed an approach to computing optimal adaptation trajectories, given the complete information on the throughput process (that is, the amount of data that can be downloaded until time t, for each t). Furthermore, this approach can be used for studying the impact of media characteristics, such as the number of representations, and network properties, such as different throughput dynamics, on the optimal streaming performance.

Streaming Client Architecture

Last but not least, based on my experience with implementing streaming client prototypes and simulation models, I have developed a streaming client architecture that is modular, extendible, and platform-independent, and that supports distributed operation of the individual streaming components. The developed architecture facilitates the integration of various adaptation approaches, both for live streaming and VoD, including algorithms that leverage cross-layer or context information, and including distributed implementations of the adaptation logic.

Chapter 2

Background

In this chapter, we provide the necessary background information on the technologies used in the present thesis. We start with an overview of the broad area of Internet-based multimedia streaming in Section 2.1, including a detailed description of the operating principles of HTTP-Based Adaptive Streaming (HAS). In Section 2.2, we introduce the notion of the Quality of Experience (QoE) and describe the inherent challenges in objectifying and quantifying this concept, in particular for HAS. We provide an overview over small cell wireless networks, that are targeted by our cross-layer approach, in Section 2.3. We conclude this chapter with Chapter 4 that introduces the notation.

2.1 Internet-Based Video Streaming

In this section, we first highlight some of the major cornerstones in the historical development of Internet-based streaming, or, as it is frequently called, Over-the-Top (OTT) streaming. We proceed with a characterization of the streaming landscape by categorizing existing approaches based on a few core features. We then go into details on the important aspect of the choice of the transport protocol. Finally, we describe the technology of adaptive streaming, focusing mainly on HAS, which is the target of this thesis.

2.1.1 A Historical Overview

As early as in the 1970s, researchers started to study approaches to transmit multimedia content over packet-switched networks [61, 195]. At that time, the focus was on the voice traffic, since neither the network equipment nor the user terminals had the capacity and computational power to transmit and render video content.

In the beginning of the 1990s, multimedia technologies started to establish themselves on the desktop Personal Computers (PC's). Audio and video clips could be digitized, encoded, and stored as files, and then decompressed and rendered on the screen. The first natural extension to this process, enabled by the advance of the Internet, was to download a video file from a server to the local machine where it could then be played back. This, however, had the drawback that a user had to wait until the complete file was downloaded, which could take a long time. In addition, she or he had to store the video file on the local machine, which could be a problem due to a limited storage capacity. These drawbacks created the need for streamed media that could be watched while still being downloaded has emerged. Ideally, it should also offer the interactive functionality of a Videocassette Recorder (VCR) such as fast forward, rewind, pausing and indexed jumps [47]. An overview of the early trials of Video on Demand (VoD) services over packet-switched networks is presented in [39].

Consequently, in the 1990s and early 2000s a lot of effort was concentrated on the development of streaming solutions operating on top of the standard Internet protocol stack [216, 233]. One of them was the successful Real-Time Transport Protocol (RTP)/Real-Time Control Protocol (RTCP)/Real-Time Streaming Protocol (RTSP) suite [172, 173]. It provides the means to establish and control the streaming session, and to monitor the Quality of Service (QoS) in order to enable the application to react to delay variations (jitter) and packet reordering/losses.

Even though RTP was designed to be independent from the underlying protocols, it is typically used on top of the User Datagram Protocol (UDP) [158] (but see also [115]). UDP is a transport protocol that provides means to send data with a minimum of protocol mechanisms. It is transaction-oriented, that is, there is no overhead for connection establishment and teardown. It does not provide mechanisms to ensure reliability, that is, the delivery is not guaranteed. Moreover, it does not provide any mechanisms to avoid or react to network congestion. All such mechanisms, if required, have to be implemented by the higher protocol layers (see also Section 2.1.3 for details).

At about the same time, there were many efforts to revisit the original Internet communication pattern of unicast, that is one-to-one, data flows. With unicast, the same content is independently simultaneously streamed to many users, which is inefficient since a separate copy of the data is sent to each destination. IP multicast [46] was proposed as a solution that provides multipoint delivery directly at the network layer of the Open Systems Interconnection (OSI) model [233].

In its basic form, IP multicast delivers the same video stream to each user of a multicast group. This causes problems in the case of heterogeneous network conditions of the individual users that may, e.g., experience very different throughput. Approaches such as Receiver-Driven Layered Multicast (RLM) [140] were proposed to overcome this issue by creating multiple multicast groups for the same content, each group streaming a specific representation of the content (for example, encoded at a specific media bit rate). By tentatively joining and leaving groups, the client was supposed to find the right representation.

Unfortunately, despite the intensive research and large-scale pilot projects [53], and despite the fact that IP multicast implementations have been available in many end devices and routers, it has never been widely deployed on the Internet. The reasons

include architectural shortcomings, security and economic concerns [48], but also the challenges related to performing application-layer adaptation in a multicast stream [123].

Another approach to overcome the inefficiency of parallel unicasts was Application Layer Multicast (ALM) or Peer-to-Peer (P2P) streaming [8, 34, 44, 83, 137, 153, 178, 225, 227]. With P2P streaming, only a subset of the users obtain the data directly from the server(s). Those who have received certain fragments of the video content, pass them on to other users, resulting in one or multiple distribution trees or a distribution mesh, where most of the data exchange takes place directly between the end users, no longer involving the original source(s).

In addition, especially in mesh-based P2P networks, a technique called network coding can facilitate the exchange of content fragments. Using network coding, or a variant called random linear network coding [73], packets from a certain time window can be jointly encoded in such a way that allows a peer to reconstruct the original packets from any fixed-sized set of encoded packets mitigating the "rarest piece" problem [209].

The fact that the resulting overlay topology is typically created without taking into account the underlying physical network topology reduces the efficiency of the P2P communication paradigm. Still, the service provider is relieved from the burden to serve thousands or millions of parallel unicast streaming sessions, and the total upload capacity increases with the number of users participating in the overlay, making the system highly scalable.

Several P2P streaming systems have had a tremendous success in certain regional markets, e.g., PPLive[1] [207] and PPStream[2]. A recently established service called BitTorrent Live[3], which is based on an enhancement of the very popular P2P file sharing protocol BitTorrent [40], has still to establish itself and accumulate a critical mass of available content.

Unfortunately, the decentralized nature of P2P file sharing had enabled copyright infringement on a massive scale, and thus has severely discreded the P2P technology for many years to follow. The issue of lawfully streaming copyright-protected content via P2P networks has never been solved to the extent satisfying influential content owners. Another drawback of the technology is that P2P clients may generate a high amount of upstream traffic serving the downloaded content to other peers, and thus they may be restricted or even blocked by the network operators.

In the last years, the technological trend for OTT video delivery has taken a different direction. Namely, Hypertext Transfer Protocol (HTTP) has established itself as a *de facto* standard in this domain. HTTP was developed in the early days of the World Wide Web as the main application layer protocol used by a web client to fetch web pages from a server [15, 59, 79]. By using HTTP, streaming clients are leveraging the ubiquitous and highly optimized HTTP delivery infrastructure, including Content Delivery Networks (CDNs), caches, proxies, etc. The services providers benefit from lower operational costs due to the lack of necessity to maintain specialized video servers and pay for their licenses. HTTP is typically allowed to traverse middleboxes, such as Network Address Translation (NAT) devices and firewalls. Finally, HAS has good

[1] http://www.pplive.com
[2] http://www.ppstream.com
[3] http://www.bittorrent.com

Delivery technology	Properties
Cinematography	"Offline" delivery to selected locations; playback using expensive equipment
Classical broadcasting	Terrestrial/cable/satellite television; delivery via dedicated infrastructure; playback mostly using dedicated hardware
Videocassette, DVD, Blu-ray	"Offline" delivery; playback using dedicated hardware
File download, then playback	Delivery over the Internet (using File Transfer Protocol or Hypertext Transfer Protocol); playback using standard computing equipment
Internet Protocol multicast	Delivery over the Internet, or over dedicated Internet Protocol Television networks; playback using dedicated hardware (receivers, Set-Top Boxes), or standard computing equipment
Peer-to-Peer streaming	Delivery over the Internet (using dedicated application layer protocols such as SplitStream, or BitTorrent [137]); playback using standard computing equipment
Over-the-Top unicast streaming	Delivery over the Internet (typically using RTP/UDP or HTTP/TCP), typically enhanced by Content Delivery Networks; playback using standard computing equipment

Table 2.1: Video delivery evolution

scalability properties due to the stateless nature of HTTP, and since the control logic resides at the client. It is also worth noting that HTTP uses Transmission Control Protocol (TCP), which is a reliable transport protocol. It thus enables usage of efficient video compression technologies that are particularly sensitive to packet losses.

In fact, the idea of using HTTP for streaming is not new. Even some of the early streaming applications, such as VivoActive 1.0, were using HTTP to keep the design simple [42]. However, it was not until recently that HTTP-based streaming has reached a broad deployment. Partially, because the commercial success of Internet-based video streaming itself was not possible before the high-speed network infrastructure, including broadband wireless and mobile networks, has become ubiquitous. Another key enabler was the broad availability of hybrid TV's that are connected to the Internet, and of mobile devices with large high-resolution screens and enough computational power to stream high-quality video. Finally, the open standards MPEG-DASH (Moving Picture Expert Group Dynamic Adaptive Streaming over HTTP) [50, 180, 183] and Hypertext Markup Language (HTML) 5 have enabled the interoperability required for a wide sup-

port across hardware and software manufacturers, service and content providers.

Initially, the dominant type of HTTP-based streaming was the so-called progressive download. The client simply started the playback of a video file while the file was still being downloaded via a single HTTP request. This required certain support from the deployed video format, since all the information needed for decoding a video frame had to be stored within the file either before that frame or very shortly after. More recent approaches split the file into segments of a few seconds duration that are subsequently downloaded by the client [99]. Coupled with the possibility to provide each segment in different representations, this allows the client to adjust the stream to the network conditions. In addition, segmenting the video file has facilitated caching and ad insertion. We will introduce this technology in more details in Section 2.1.4.

An overview of the outlined evolution of video delivery technologies is presented in Table 2.1.

2.1.2 The Streaming Landscape

The diversity of approaches to OTT video delivery is enormous (see, e.g., [13, 14, 125] for an overview). In Table 2.2, we categorize them along five attributes: delay requirements, control location, distribution method, transport protocol, and adaptivity. In the following, we briefly describe each of the attributes and the typical values they can take.

Latency

One fundamental characteristic of a video streaming approach is the latency it is designed for. W.r.t. the latency, we distinguish between interactive streaming, live streaming, low-delay live streaming, and VoD. Examples for interactive streaming include video conferencing and telepresence. Examples for live streaming include transmissions of sports events or live transmission of video games[4]. An essential property of interactive and live services is that the content is being streamed while the event being recorded is taking place. Consequently, the delay of the transmitted sequence influences the value the sequence has for the receiver. In contrast, with VoD, the content is prerecorded and stored at the server, and thus, VoD streaming does not have any latency requirements. (It still has requirements on the start-up delay, though, especially in cases, when the user tends to watch short videos or frequently switch between channels.)

The latency of interactive services should not exceed 100 ms [88, 92]. The latency of OTT live streaming services can reach tens of seconds. Even though live services often benefit from a low latency, achieving it is challenging and may lead to a reduction in video quality. For some live services, however, the latency requirements are tighter. A live stream of a surveillance camera should have a smaller latency in order to enable shorter reaction times. Another example is passive participation in a conference or a lecture. While the (typically few) active participants have the latency requirements of interactive streaming, it is more efficient for the service provider to serve the passive participants with larger delays, potentially deploying a different streaming technology with a better scalability. However, since a passive participant may become active (e.g.,

[4] http://www.twitch.tv

Delay requirements	Interactive (up to 100 ms), low-delay (100 ms to 10 s), live (1 s to 30 s), VoD (not applicable)
Control location	Sender-driven, receiver-driven, network-driven, hybrid
Distribution method	Broadcast, multicast, P2P, unicast, hybrid
Transport protocol	UDP, RTP/UDP, TCP
Adaptivity	Adaptive, non-adaptive
Viewpoint	Single-view, multi-view

Table 2.2: Streaming landscape; note that control location refers to the control of the data flow not to session control in general.

by asking a question), his delay must be small enough to avoid a significant "time jump" when switching to the interactive mode. We call live streaming services with latency requirements ranging from hundreds of milliseconds to few seconds low-delay live streaming.

Control location

Another important attribute of a streaming solution is the location of the application-layer flow control. Traditionally, this functionality has been implemented at the server side. This, however, may lead to scalability issues, since the server has to keep state for each flow, and implement various mechanisms that control the streaming process. Moreover, information about the QoS of the network path has to be accumulated by the client and fed back to the server, creating additional delays in the control loop. Consequently, many of the current non-interactive streaming services that are deployed on a large scale are client-driven. There are ongoing efforts to create frameworks for hybrid – *network-assisted* or *server-assisted* client-driven – streaming services that can further optimize the network resource allocation, and thus improve efficiency and fairness. One example is the recent extension to the MPEG-DASH standard called Server and Network Assisted DASH (SAND). An example for a network-driven delivery method is a transcoding-based approach, where the quality of the stream is adapted based on each client's network path conditions at the CDN or proxy nodes [189].

Distribution method

We have already mentioned the inefficiency of simultaneously unicasting identical content to multiple receivers, and some of the alternative distribution methods developed to overcome this drawback, such as IP multicast and P2P streaming. In addition, some of the ongoing efforts are focusing on enabling IP-based broadcast over cellular (e.g., evolved Multimedia Broadcast Multicast Service (eMBMS)) or satellite links, or to combine broadcast and unicast distribution, for example by broadcasting the base layer of a

scalable video to all participants, and by unicasting enhancement layers to those clients who have a fast enough Internet connection, and a high-resolution screen.

Transport protocol

The choice of the transport protocol for OTT streaming was historically decided in favor of UDP, which provides a minimum of overhead and thus fits the needs of many real-time applications. However, later on TCP became the protocol of choice. In the dedicated Section 2.1.3, we describe the impact of the transport protocol choice in more details.

Adaptivity

An important property of a streaming technology is its capability to perform dynamic adaptation of the transmitted media format to the changing network conditions, device capabilities, or user context. If a streaming approach is non-adaptive, the network connectivity and the end devices of all users receiving the stream must satisfy the requirements of the media format used for the transmission. Thus, either the least common denominator is used, negatively impacting the "premium" users, or some of the users will not be able to receive an appropriate QoE, or even fail to start the streaming session. We will discuss adaptive streaming, which constitutes the focus of this thesis, in Section 2.1.4.

Viewpoint

Traditionally, the consumption of video content has been limited to a single perspective of a scene, corresponding to a location and angle of the camera, which was determined during the content production phase. Meanwhile, the advances in imaging and computing technology are bringing multi-view video to the mainstream, allowing the user to dynamically select the perspective she or he wants to see [192]. Combined with 3D video and the ability to control the perspective by head movements, multi-view systems are offering an immersive Virtual Reality (VR) experience that can greatly enhance the QoE in various application domains, including entertainment, videoconferencing [218], medicine (remote surgery), education, etc. In principle, multi-view video can be transmitted using the technologies designed for single-view video. However, this is inefficient due to the involved transmission of large amounts of dispensable content that is not perceived by the user. Instead, taking into account the current and predicted user's perspective allows to reduce the amount of transmitted data and greatly increase the QoE [29, 58, 154].

2.1.3 The Choice of the Transport Protocol

The two transport protocols dominating the Internet since its early days are UDP [158] and TCP [201]. Both serve quite different purposes.

TCP was designed to support a reliable and connection-oriented data transfer. Designed to operate over networks where individual packets might get lost or arrive out of order, it uses retransmission and receiver-side buffering to deliver the data to the

application completely and in strict order. In addition, it is equipped with congestion avoidance and congestion control mechanisms that adjust the sending rate based on the sender's estimation of the available network capacity, in order to ensure the stability of the network and a fair allocation of resources. Since its first design in 1974 [25], TCP has been subject to a considerable optimization and reshaping process, and is currently used to transport over 90% of the data on the Internet [100].

In contrast to TCP, UDP provides means to send data with a minimum of protocol mechanisms. It is transaction-oriented, that is, there is no overhead for connection establishment and teardown. It does not provide mechanisms to ensure reliability, that is, the delivery of the individual data packets is not guaranteed. Morever, it does not provide any mechanisms to avoid or react to network congestion. All such mechanisms, if required, have to be implemented by the upper protocols or by the application itself.

For the transmission of data that require a particularly low delay on the order of tens or hundreds of milliseconds, as, e.g., with real-time applications such as interactive streaming, UDP is clearly the preferred choice. For these services, the utility of the individual packets quickly decreases with the increasing delay, and becomes zero when it exceeds a certain threshold. TCP, however, sacrifices timeliness for reliability by retransmitting lost packets while delaying the delivery of all subsequent packets until all packets can be delivered in order, considerably increasing the delay in the presence of packet losses. In addition, many TCP flavors interpret packet losses and sporadic transmission delay peaks as congestion signals, and react by reducing their sending rate. Depending on the deployed video encoding technique, however, using a higher sending rate, while tolerating a certain amount of lost packets, may lead to a higher QoE.

On the other hand, however, streaming services that are using an unreliable transport protocol such as UDP need to deploy mechanisms mitigating the impact of packet losses and out-of-order delivery. Although most decoders are able to operate in the presence of packets losses, reconstructing transmitted video frames from incomplete information results in more or less visible artefacts. Moreover, since modern video compression methods deploy inter-frame coding that leverages temporal redundancy in the content, decoding a frame typically involves accessing previous, or even subsequent frames [185]. Thus, the loss of a packet does not only result in one corrupted frame but propagates to a number of other frames. In order to mitigate this impact, applications have to deploy mechanisms, such as retransmissions, forward error correction [64], error concealment techniques [30, 208, 212], error-resilient encoding [191], or a combination of those [226]. Typically, UDP is combined with RTP/RTCP [173], which facilitate monitoring the QoS and reacting to changing network conditions. In addition, with a lossy transmission the Group of Pictures (GOP) size should not exceed a certain threshold in order to limit the error propagation, which negatively influence the compression rate.

Whenever the receiver is willing to tolerate a delay on the order of seconds or more, as in the case of live streaming or VoD, the effects of retransmission delays and varying sending rate can be mitigated by introducing a playback buffer at the receiver side. In this case, TCP becomes a valid choice [109]. In addition, using TCP also offers several benefits. Services that leverage a reliable transport protocol do not have to react to packet losses, since the lost packets are retransmitted. In addition, TCP provides built-in congestion avoidance and congestion control mechanisms, that are necessary to

maintain network stability, as well as to ensure basic fairness among competing flows. It thus helps reducing the complexity of the streaming application. Furthermore, reliable communication enables the usage of efficient video compression technologies that are particularly sensitive to packet losses (the loss of an I-frame may result in several seconds of corrupted playback).

However, this gain does not come for free. The retransmissions make the throughput dynamics more complex, and thus require more sophisticated adaptation algorithms. In addition, the operation of the congestion avoidance and congestion control mechanisms, further increases the complexity. Many video quality adaptation approaches rely on explicit or implicit throughput estimations. Consequently, their performance is highly correlated with the accuracy of the estimations. This is particularly true for low-delay streaming, since, due to its delay constraints, it has a tight limitation on the maximum level of the playback buffer. Therefore, performing efficient adaptation on top of reliable protocols such as TCP is highly challenging.

RealVideo, which is one of the first popular streaming solutions introduced in 1997, leveraged UDP in order to benefit from the better utilization of network resources and reduced throughput and delay fluctuations, but suffered from problems such as lost or delivered out of order packets. One recent example is WebRTC[5], a multimedia communication framework for web browsers.

Other early streaming applications, such as VivoActive 1.0, were using TCP, which helped to keep the design simple. The impact of throughput fluctuations was mitigated by building up a playback buffer of 5-10 seconds prior to starting the streaming session [42]. In the last years, streaming solutions that make use of HTTP, which uses TCP as transport protocol, have become increasingly popular and are meanwhile dominating the Internet traffic.

There also exist approaches that can be seen as a compromise between UDP and TCP, e.g. performing TCP-friendly end-to-end congestion control but without the delay-constrained reliability [60, 94, 164]. Several such approaches have been standardized but have not seen wide deployment.

2.1.4 Adaptive Streaming

Any OTT streaming technology has an inherent challenge. The Internet is a so-called best-effort network; it was not designed to provide any QoS guarantees. Considerable effort has been put into developing networking architectures addressing this shortcoming [10, 21, 62, 224]. So far, however, none of them has seen wide deployment. One part of the problem lies in the complexity of implementing QoS models, forcing network operators to resort to costly "brute-force" solutions based on resource overprovisioning.

Another way to overcome the problem is to adapt the media characteristics of the streamed content to the dynamically varying network conditions. This approach is also more in line with the traditional Internet design principle, requiring that the complexity belongs at the edges, while the core should be kept simple and stateless [19]. This idea becomes manifest in the concept of adaptive streaming.

[5] https://webrtc.org

In fact, a dynamic adaptation of the operating parameters of an application to the available resources is a general problem which rises in different contexts. Whenever the scarcity of the resources does not allow to satisfy the maximum requirements of each single application, two basic issues need to be addressed. First, how to allocate the available resources to the individual applications, and second, how to meet the operational objectives of each application despite the undersupply.

To keep the system design simple, these two issues are often solved separately, leading to suboptimal performance. For example, an operating system might allocate an equal fraction of the Central Processing Unit (CPU) time to each application, even though the demands of the applications can be quite different. Another example is allocating the same fraction of the network capacity to a background file transfer and a video stream. While the user might not even notice that her or his backup completed earlier than expected, they would most probably be quite unhappy to watch their favorite movie in a low quality.

A more advanced approach might take into account the extent to which an application is capable of adapting its operation to the available resources, e.g., in form of a utility function [143]. A generic middleware adaptation framework which jointly solves these two problems by using control theory is presented in [124]. It consists of two main components, a task control model performing the resource allocation, and a fuzzy control model mapping the allocated resources to the configuration parameters of an application. The developed framework is successfully applied to an application which, at the client side, tracks objects in a video stream transmitted from a server. In this example, the application-layer adaptation is performed by trading off the rendered video quality against an accurate and stable tracking.

Adaptive delivery of multimedia content has been in the focus of the research community for quite some time [32, 96, 104, 110, 145, 163, 205, 210]. The proposed approaches and frameworks are quite diverse: controlling the encoding rate of a live source [104, 203], server-based or network-based filtering [94], transcoding [6], selecting a subset of available media layers [127, 149, 191], selecting from a set of independent media representations [132], or by a combination of several of these approaches [226]. Other examples include adapting interactive streaming services to the varying end-to-end delay [72].

One of the early approaches is SureStream [132], introduced in 1998. With SureStream, multiple representations of the original content are stored in a single file. While monitoring the network conditions, the client instructs the server to switch to a particular representation. The approach benefits from reduced complexity due to the client-side implementation of the adaptation logic. It is also worth noting that it is not tied to a particular file format or video coding. Also, SureStream was one of the first systems built on the Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) standards for Internet multimedia [42].

One of the currently most successful streaming technologies, HAS, includes adaptation as its core design feature. Among the first works proposing HAS-based approaches are [24, 132]. However, it took almost a decade until HAS has seen wide deployment and commercial success. HAS flavors such as Apple HTTP Live Streaming (HLS)[6] or Microsoft Smooth Streaming [223] has been used by various streaming services, and

[6] https://developer.apple.com/streaming/

have been supported by most operating systems and browser environments. In 2011, an open standard Moving Picture Expert Group (MPEG)-Dynamic Adaptive Streaming over HTTP (DASH) [50, 180] was created to facilitate the interoperability.

In a HAS system, the video content is encoded in several representations that may differ w.r.t. various characteristics such as the resolution, frame rate, compression technology, compression rate, video format, etc. They are typically configured by the service provider during the planning phase [175]. Typically, the representations differ w.r.t. their mean media bit rates, that is, the average number of bits representing one second of video.

Each representation is split into segments, typically containing several seconds of video data. Each segment starts with a random access point of the stream such that segments from different representations can be concatenated to obtain a valid video file. Consequently, switching the representation is feasible on each segment boundary. Note that this approach requires a suitable container format that enables fast transitions between representations, and has a low overhead [165].

HAS-based solutions are typically client-driven. In the beginning of the streaming session, the client downloads an Extensible Markup Language (XML) file called the Media Presentation Description (MPD) or the manifest file. It contains the stream meta data, including links to the individual segments, or an instruction on how to construct the links from the individual components such as, e.g., the base Uniform Resource Locator (URL), the file naming scheme, the representation index, and the segment index.

The client then issues a series of HTTP GET or GET RANGE requests to download the segments, typically in chronological order, selecting a representation for each of them from the set of available representations. More recently, the HTTP/2 standard has introduced the Server Push feature that allows the server to respond with a preconfigured sequence of segments to a single request. This feature has the potential to reduce the protocol overhead and increase the responsiveness, which can be used to improve the efficiency of low-delay streaming [198, 214]. After a segment is downloaded, it is stored in the playback buffer until the playback of the previous segment has been completed.

If the download is not completed in time, the playback is halted. We term this event a buffer underrun. According to multiple studies, the frequency and the duration of the buffer underruns have the strongest impact on the QoE. Consequently, in order to avoid underruns, a client strives to keep the playback buffer at a level that is high enough to mitigate the impact of throughput drops and link outages. Especially in wireless networks, where the link quality is affected by interference and fading effects, but also in busy networks with a lot of cross-traffic, strong throughput fluctuations are quite common.

Since the individual representations typically have different media bit rates, the client is able to control the buffer level to a certain extent by dynamically selecting an appropriate representation for each downloaded segment. Here, one need to distinguish between VoD and live streaming. With VoD, the complete video content is available for download the whole time throughout the streaming session. With live streaming, the content is being recorded, encoded, and published to the servers while being streamed. Consequently, a VoD client may theoretically prefetch the complete rest of the content into

its playback buffer while streaming, if the media bit rate of the selected representation is sufficiently small as compared to the available network throughput. In contrast, with live streaming the possibility to prefetch content is severely limited. Particularly with low-delay live streaming, the buffer level cannot exceed a few seconds.

We remark, however, that even with VoD, where the complete content is available for download, the buffer level is typically bounded from above. This bound is determined by several factors. It may be determined by the available storage capacity, especially for embedded devices that may have tight memory resources. It may also be limited by the service providers in an effort to prevent the client from wasting network resources by downloading content that won't be presented because, e.g., the user quits the streaming session prematurely or because a throughput increase allows the client to switch to a higher quality, discarding the already downloaded low-quality content.

As already mentioned, if a segment cannot be downloaded prior to its playback deadline, the playback is interrupted. The subsequent procedure again depends on the type of the streaming session. With VoD, a client waits until the segment is downloaded, then resumes the playback at the position where it has been interrupted. In addition, the client may even further delay the playback to prefetch more content into the playback buffer in order to minimize the risk of another playback interruption in quick succession. In contrast, a low-delay live streaming client has to satisfy latency requirements, defined by the user or the service provider. Consequently, when a segment is delayed, the client may have to skip its playback and proceed with one of the subsequent segments.

In this context it is worth noting that short segment duration helps improving the client's responsiveness to throughput changes. At the same time, however, small segments increase the overhead due to the higher number of HTTP requests as well as reduce the video compression efficiency due to the decreased GOP size. Typical segment durations lie between 2 and 15 seconds.

Since HTTP offers no means to cancel an ongoing request, the only way to avoid a buffer underrun in the case of a sudden throughput drop or to avoid wasting bandwidth downloading a segment whose playback will be skipped is to use a different TCP connection and to shutdown the old one. Since opening a new TCP connection is associated with communication overhead, the client may maintain multiple TCP connections, using them in a Round Robin manner in order to keep their internal state such as congestion window size and Round-Trip Time (RTT) estimation up-to-date. In this way, the client is able to immediately start using another TCP connection, while closing and replacing the old one.

At the core of a HAS client is an adaptation algorithm that selects the representation for each of the downloaded video segment. The adaptation is not part of the MPEG-DASH standard and thus it has been subject to intensive research efforts over the last years. To meet its adaptation decisions, an adaptation algorithm is using the information it can acquire from its environment. Most often this information includes the dynamics of the past throughput, with a level of details depending on the underlying platform, and the dynamics of the level of the playback buffer. More sophisticated approaches leverage cross-layer information from lower layers of the protocol stack or other network entities, such as a local network controller. They may also leverage context information, such as the location [166], mobility pattern, or available sensor data.

As we already have mentioned, the segment size may vary across the segments of the same representation if Variable Bit Rate (VBR) encoding has been used to generate the media data. The information about the sizes of the individual segments may or may not be available to the client. It may be conveyed by the MPD file but not all possible formats contain this information. It may also be fetched by the client by using, e.g., HTTP HEAD requests, prior to requesting the actual segment. Finally, it may be signaled to the client using in-band mechanisms such as the DASH Events which have been included into the second edition of the MPEG-DASH standard [51]. If no information about the segment sizes is available, the client may approximate them by the average segment sizes that it can compute from the average media bit rates of the individual representations, contained in the MPD.

Using precise knowledge about the individual segment sizes is particularly beneficial in the case of low-delay streaming, where this information can help the client to precisely estimate if a certain segment can be downloaded prior to its playback deadline or not. In the case of VoD, the typical buffer sizes of several tens of seconds and the typical segment durations of few seconds make the benefit of knowing the precise segment size less pronounced. In this case, the average segment size computed over the number of segments that correspond to the target buffer level should be fairly closely approximated by the average segment size of the representation.

The adaptation algorithm used by a streaming client has a strong impact on the QoE. If, e.g., the adaptation decisions follow the short-term changes of the network throughput, the resulting average video quality may be quite high but the overall QoE will decrease due to a high rate of quality fluctuations. On the other hand, a conservative approach might reduce the risk of playback interruptions due to video segments missing their playback deadline, but at the same time decrease the overall video quality. Thus, a good adaptation strategy needs to maintain a balance between several trade offs that we will discuss in more details in Section 2.2.

2.2 Quality of Experience

When developing a system or evaluating its performance, it is crucial to define the goals it has to achieve. For a video streaming service, identifying these goals and expressing them in a way that facilitates objective measurement has turned out to be a highly challenging task.

At the dawn of the OTT streaming era, QoS metrics of the lower networking layers, such as the packet loss rate or delay jitter, were used to assess the streaming performance. Later, it has been recognized that an evaluation of a multimedia delivery service must inevitably take into account human's perception and cognitive processing, involved in consuming the video content [217]. These phenomena, however, are influenced by a large amount of hardly measurable factors.

The notion of the QoE was introduced in an effort to assess these phenomena and help making them accessible to an objective evaluation process. The International Telecommunication Union (ITU) defines QoE as "the overall acceptability of an application or service, as perceived subjectively by the end-user", which might be influenced by "user expectations" and "context" [90]. In the following, we describe the main factors influ-

encing QoE, along with the available evaluation methodologies, with a closer look on QoE for OTT streaming.

2.2.1 QoE Influence Factors

The number of factors influencing the QoE is immense, and many of them have a high level of subjectivity, which results in extremely difficult modeling. Based on [20, 162], we group them into the following categories: System Influence Factors (SIF's), Context Influence Factors (CIF's), and Human Influence Factors (HIF's).

System Influence Factors

SIF's are factors related to the properties and characteristics that determine the technically produced quality of an application or service. They include properties of the content, such as the amount of detail or motion, or the amount of depth in a 3D video. Further, they include device-related factors, such as the display characteristics [108], support for 3D [188], or computational power. They also include *media-related and network-related factors*, that constitute the focus of the present work.

It is mostly unavoidable that video content recorded in a high quality has to be compressed in order to be transmitted over a network since there is typically a substantial gap between the network throughput and the media bit rate of the recorded video. To a certain degree, the compression may be performed without a noticeable quality degradation. However, compressing High-Definition (HD), 4K and 8K Ultra-High-Definition (UHD), or even VR streams such that they can be transmitted over the Internet inevitably leads to a visible distortion, including, e.g., blocking effects, blurring, edginess, motion jerkiness, etc. After being compressed, the content has to be encoded into a certain format that can be processed by the target playback device. The nature and parameters of the used compression method, the encoding format, resolution, sampling rate, and frame rate are media-related SIF's.

Network-related SIF's include the end-to-end characteristics of the network path from the source to the streaming client. They include throughput, delay, packet loss rate, as well as their temporal variations. These factors are frequently termed QoS, although the official definition of this term has a broader scope [89].

Context Influence Factors

CIF's include situational properties of the user's environment [103]. They include the physical context (indoor/outdoor, quiet/noisy, lighting conditions, but also the level of activity, such as sitting/walking/jogging), the temporal context (time of day/week/-month/year), the social context (is the person alone or not, does the experience involve inter-personal actions), the economic context (service cost), and the task context (level of distraction). For example, a user might be not interested in a long movie during working hours, or in slow music during a physical exercise. The same video content might leave a totally different quality impression when watched on a mobile phone while riding on the bus than when watched on a TV screen at home. Context-aware multimedia services have recently become an active research topic, empowered by the increasing intelligence

and the ubiquity of interconnected devices that are able to recognize situational context, learn user's preferences, and consolidate the information to provide services tailored to user's needs under the given circumstances [231].

Human Influence Factors

HIF's include the demographic and socio-economic background of a person, her or his physical and mental constitution, or the emotional state. It was shown that, e.g., user's expectations, prior knowledge and skills, and technology affinity may significantly affect QoE [169, 182, 217]. Due to their subjectivity and relation to internal states and processes, however, the influence of HIF's on QoE is difficult to model and is still poorly understood [162].

Controlling QoE Influence Factors

If a streaming services operates over a managed network, many network-related factors can be controlled by the service provider, e.g., by investing into the equipment and the infrastructure, and by optimizing the network configuration for the requirements of the particular service. Thus, in this case both network-related and media-related factors can be jointly optimized to deliver a high QoE. An example for such a service is Internet Protocol Television (IPTV).

With OTT streaming, however, the situation is different. The Internet "does its best" to deliver the packets in a fast and reliable way but does not provide any guarantees. Consquently, both network-related and media-related SIF's can and typically do vary during the course of a streaming session. In order to provide a high or even the best possible QoE, a streaming technology has to dynamically adapt the characteristics of the streamed media to the varying network conditions. This adaptation may be defined as adjusting media-related and/or network-related SIF's that can be controlled to those that cannot.

Finally we would like to remark that while taking into account the CIF's and the HIF's has the potential to increase the QoE of streaming services, their complex and subjective nature, and the lack of an established model combining them into a quantifiable metric, are hindering them from entering state-of-the-art technologies.

2.2.2 QoE Evaluation Methodology

Once the influencing factors presented in Section 2.2.1 are identified, it is necessary to determine their impact on the QoE, as well as their relative importance. The methods for doing that can be grouped into three categories: subjective tests, objective models, and data-driven models [31].

Subjective tests

With subjective tests, QoE of a set of videos is evaluated by human viewers in a laboratory environment [91]. Asking a person directly is a natural way to determine the quality of his or her experience. Still, subjective tests have certain limitations. First,

they are costly due to the required time, equipment, and human effort. Moreover, they are conducted in a laboratory environment, with a limited variability of tested video characteristics, test conditions, and viewer demography. Finally, subjective tests cannot be used for a real-time QoE evaluation required, e.g., in order to dynamically tune the QoE of a service.

Objective models

To avoid these drawbacks, many studies have been focusing on objective models to allow computing QoE from measurable influencing factors without human intervention [33, 107, 131]. Ideally, they should reflect the way video signals are processed by the Human Visual System (HVS). Objective models should correlate with subjective test results, and may be validated using the latter as the ground truth. Some objective quality models rely on subjective tests to train model parameters.

Objective models can be categorized into Full Reference (FR), Reduced Reference (RR), and No Reference (NR) models, depending on the amount of information they require about the original content.

One commonly used, although imperfect, method is to quantify the difference between the original and the distorted video, potentially weighing the errors according to spatial and temporal features of the video. Many studies use the Sum of Squared Differences (SSD), or its equivalents, such as the Mean Squared Error (MSE) or the Peak Signal-to-Noise Ratio (PSNR) [150].

Another approach that takes into account the fact that the HVS is highly trained to extract structural information from images, and is thus very sensitive to structural distortion, is the Structural Similarity (SSIM) index [213]. Yet another approach is the Video Quality Metric (VQM) [157] that has been accepted by the ITU-T as a recommended objective video quality metric.

In order to develop QoE prediction models that do not depend on the original videos, network statistics (such as the packet loss) and spatiotemporal features extracted or estimated from the distorted video, may be leveraged. A performance evaluation study in [93] compares 9 FR, RR, NR models that take into account network statistics extracted from the received video bitstream.

Unfortunately, all mentioned approaches are designed to measure the quality of a video subject to impairments resulting from packet losses, and other artefacts, rather than losslessly transmitted adaptive video, as in the case of HAS.

Data-driven models

Recently, data-driven models have emerged as a promising way to circumvent the drawbacks of the other two methods. Data-driven models leverage large-scale data collections created from the feedback provided by the streaming clients. The large volume of the collected data allows to replace sophisticated models required to draw valid and reliable conclusions from small data sets by simpler models without affecting the performance. Based on the collected data, individual influence factors can be mapped to metrics such as the user engagement, which reflects the viewing time, the number of watched videos and the probability of return [12, 43, 111].

2.2.3 QoE for HTTP-Based Adaptive Streaming

QoE for HAS is currently a very active and fast developing research area [11, 162, 175, 181]. The main two design aspects of HAS that influence the degrees of freedom for maximizing QoE are (i) the choice of TCP as transport protocol, and (ii) the client-driven data flow control. The choice of a lossless transport protocol eliminates impairments due to packet losses. Instead, packet losses are translated into delayed packets and throughput fluctuations. Locating the application-layer stream control at the client determines the information available as input to the decision process.

The main factors influencing QoE that can be controlled by a HAS client are: the number and duration of playback interruptions, the adaptation trajectory (determining the video representations selected for the individual video segments and the frequency of representation transitions), the initial delay, and, in the case of live streaming, the live latency. In the following, we describe each of these factors in more details.

Playback interruptions

When a streaming client's playback buffer has been drained, and the next video segment does not arrive before its playback deadline, the playback must be halted – a playback interruption occur. This is often referred to as a buffer underrun. In the case of VoD, a buffer underrun is typically followed by a rebuffering period, where the client waits until enough video data is accumulated in the buffer to resume playback. The conditions that need to be fulfilled before the playback is resumed depend on the client's rebuffering strategy. In the case of live streaming, rebuffering increases the live delay since a segment is played out at a later time than its playback deadline. In the case of a soft bound on the live latency, as determined by the service provider or by the user, the client might chose this option. In the case of a strict bound, the client has to skip the playback of the late segment, cancel its download, and continue with downloading one of the subsequent segments, in order to stick to the configured latency bounds.

Adaptation trajectory

The selected video representations, or the adaptation trajectory, is obviously a factor that dramatically influences the overall QoE, by influencing the video quality (video distortion) of the individual video segments. A classical approach to characterize the video quality is the PSNR, which is typically a strictly concave function of the media bit rate. Video quality is, however, not the only important property of an adaptation trajectory. Recent studies have shown that also quality transitions have a significant impact on the QoE of an adaptive streaming session. It is worth noting, that due to partially quite significant media bit rate fluctuations within a single representation, resulting from VBR encoding used by many service providers, quality fluctuations cannot always be completely avoided, even in the presence of a stable network throughput.

Initial delay

At the start of a streaming session, the user's playback buffer is empty, so that the user has to wait until enough video data is downloaded in order to start the playback. This

phase is sometimes termed prebuffering or initial buffering. In contrast to rebuffering, during prebuffering, a streaming client typically does not have information about the network conditions. Especially when a user frequently starts a new streaming session, e.g., by switching TV channels, or when she or he repeatedly watches short videos, even a moderate start-up delay of 2 seconds might severely degrade the QoE and even make the user decide not to watch the video at all [111].

Live latency

In the case of live streaming, an additional factor that plays an important role is the latency. While currently, OTT live streaming services might exhibit a latency of several tens of seconds, many services would strongly benefit from reducing this value. In particular, low-delay services such as surveillance or passive participation in video conferences, requires lower delays on the order of a few seconds.

Maximizing QoE

Note that the individual factors described above cannot be considered separately. For example, in order to minimize the number and duration of playback interruptions and the number of quality transitions, one may always select the lowest video quality, which, obviously, is suboptimal w.r.t. the overall QoE if the available network capacity actually allows a higher media bit rate. On the other hand, maximizing the video quality by always selecting the highest representation will too often result in an unacceptably high number of playback interruptions.

Some of the described factors, such as rebuffering, initial delay, and quality transitions, have not been part of traditional QoE metrics for broadcast-based video transmission, but have a tremendous impact on QoE for adaptive OTT streaming. Still, the relative importance of the individual factors is not completely understood. Nevertheless, the number of studies dedicated to this topic has been dramatically increasing with the growing popularity of video streaming services, so that a lot of valuable insights are available to help designing QoE-optimized streaming approaches.

Many studies suggest that the number and duration of rebuffering periods have the most severe impact on QoE, especially with live streaming [43]. In particular, users are willing to accept a higher initial delay and higher video distortion, if it helps minimizing rebuffering periods [76, 160, 175, 179].

Other studies confirm the negative impact of video quality transitions resulting from dynamically changing the representation [122, 219, 221, 234]. In particular, some studies come to the conclusion that a lower overall video quality might be tolerated if it helps reducing the amount of representation changes [155].

Minimizing the initial delay is another important goal of a streaming client. The large-scale study in [111] reveals that a delay beyond 2 seconds may cause the viewers to abandon the video.

As already mentioned in Section 2.2.2, the user engagement is another important metric, which is especially of interest for content providers because it directly relates to advertising-based revenue schemes [12, 43, 111].

2.3 Small Cell Wireless Networks

Since the time when the Internet first entered our lives, we have been observing a steady increase in the traffic demand, stimulated by the advance of new services, miniaturization of electronic devices, and the expanding coverage and increasing speeds of access networks. An essential enabler of this success is the wireless communication technology, which provides connectivity on the go, covering areas lacking the infrastructure required for fixed line access.

It is, however, well understood that the current trend of wireless technology (e.g., Long-Term Evolution (LTE) [174]) cannot cope with the exponential traffic increase expected in the following years [37], unless the density of the deployed infrastructure is increased correspondingly [7]. In fact, throughout the history of wireless networks, throughput gains resulting from the increased network density exceeded the gains from individual other factors by an order of magnitude [26].

It is widely agreed that the next generation of wireless networks will involve a combination of multiuser Multiple Input Multiple Output (MIMO) technology, cell densification, and nested tiers of smaller and smaller cells, operating at higher and higher carrier frequencies [26]. Consequently, we have recently observed a surge of research on a dense deployment of base station antennas, in the form of massive MIMO schemes [78, 85, 139] and in the form of densely deployed small cell networks [26, 77].

Reducing the wireless cell size by using scaled-down macrocell Base Stations (BS's) requires a substantial amount of planning, integration, management, and maintenance, involving significant Capital Expenditures (CAPEX's) and Operational Expenditures (OPEX's). In contrast, the network design concept of small cell networks is based on the idea of a very dense deployment of low-cost, low-power BS's that are substantially smaller and cheaper than the macrocell equipment.

To achieve an appropriate backhaul capacity, small cell BS's may reuse the already existing wireless or wireline access points. In addition, caching and preloading of popular (video) content can significantly reduce the load on the backhaul [63, 177], due to the highly uneven popularity distribution of video content [65, 177], potentially complemented by recommendation engines optimizing the hit rate [112]. Moreover, small cells can greatly improve indoor signal quality as compared to macrocells, which has the potential to substantially improving the overall efficiency due to the fact that a large fraction of voice and data traffic originates indoors [156].

The design and performance analysis of efficient resource allocation mechanisms in small cell networks is challenging because of several factors. User terminals are typically covered by multiple BS's with different link properties. Optimal resource allocation requires Channel State Information (CSI), which grows with the amount of transmitters and receivers. Due to the dense deployment of BS's and the high frequency reuse factor, the capacity of a small cell network is severely impacted by intercell interference.

One the one hand, one potential benefit of the small cell technology is its self-organization capability. On the other hand, a centralized network entity may enable a certain degree of cooperation among the BS's in order to further increase the efficiency [141]. While, consistently with the current wireless standards, the individual cells may deploy intra-cell orthogonal access, such as Frequency Division Multiple Ac-

cess (FDMA) or Time Division Multiple Access (TDMA), the centralized controller may allocate the downstream traffic to the cells, and control the BS associations.

Chapter 3

Related Work

In this chapter, we review the existing literature on those aspects of adaptive streaming that are related to our work. We begin with adaptation strategies for Video on Demand (VoD) streaming clients, which is the most actively pursued research direction. We proceed by reviewing the literature on low-delay live streaming, followed by prediction-based adaptation. We then focus on the studies addressing cross-layer optimization for streaming, especially in wireless networks. Afterwards, we outline works studying optimal adaptation approaches. Finally, we study the literature on Transmission Control Protocol (TCP) throughput prediction.

3.1 Video on Demand

In the last years, there has been a significant number of studies on adaptation algorithms for HTTP-Based Adaptive Streaming (HAS)-based VoD. Various approaches have been proposed that are based on the control theory [220, 229, 230, 232], Markov decision processes [18, 97], machine learning [38], dynamic programming [128], data-driven techniques [12, 69, 135, 166], and various other heuristics selecting the video quality based on the client's playback buffer level and/or average throughput [98, 129, 133, 134, 146, 197].

Unfortunately, due to the lack of a standard evaluation methodology and performance metrics for HAS systems, comparing the performance results of the individual studies is hardly possible. Consequently, several studies perform comparative evaluations of the adaptation behavior of multiple (three or more) clients [3, 82, 148, 199].

Akhshabi et al. [3] experimentally evaluate three HAS clients, focusing on the following three aspects: reaction to persistent or short-term throughput changes, the ability of two players to properly operate on a shared network path, and if a player is able to sustain a short playback delay and thus perform well with live content. The authors

identify significant inefficiencies in each of the studied players.

Timmerer et al. [199] compare the performance of 10 adaptive streaming clients in a controlled environment, and using real-world measurements. The evaluation reveals that all the tested players exhibit deficiencies and that there is no clear winner. However, it also reveals that two relatively simple approaches perform reasonably well in the tested scenarios.

Liu et al. [133] propose an adaptation algorithm that selects the representation based on the throughput measured during the download of the last segment, and a minimum buffer level threshold, below which no quality increase is performed. The algorithm performs conservative upward transition and more aggressive downward transition. A simulative evaluation is performed using artificial background traffic, without a baseline approach.

Zhou et al. [229] propose an adaptation algorithm that leverages a throughput estimation and a Proportional-Derivative (PD) controller in order to keep the buffer level within a target interval. The details of the throughput estimation method are not specified. A simulative performance evaluation is conducted with constant throughput, short-term throughput spikes, and long-term throughput spikes.

Evensen et al. [55, 56] and Kaspar et al. [106] study HAS via multiple network interfaces, for VoD, live streaming with a soft latency bound, and low-delay live streaming with a strict latency bound. They propose an algorithm that selects for the next segment the highest representation that can be downloaded before the playback deadline, based on the average aggregated throughput from the past two seconds. A deployed request scheduler then assigns fractions of the video segment to the available network interfaces for the download. The algorithm includes no mechanisms to limit the number of quality transitions. In order to reduce the overhead due to the high number of sub-segment requests, a new request is issued before the previous has been completed. The performance is evaluated in an emulated test bed, and in real-world experiments. It reveals that sub-segment requests with dynamically adjusted sizes are crucial to fully utilize the aggregated capacity and achieve a high Quality of Experience (QoE), especially in the cases with highly unequal link conditions.

Müller et al. [148] propose an adaptation approach that selects the representation based on the average throughput of the whole streaming session, the average throughput during the download of the preceding segment, and a factor depending on the current buffer level. The exact formula is not provided. The evaluation is performed in an emulated environment fed by three application-layer traces recorded on a mobile 3G link while driving on a freeway, w.r.t. the total duration of playback interruptions, the Mean Media Bit Rate (MMBR), and the number of quality transitions. Three algorithms are used for comparison. The evaluation reveals that the proposed algorithm performs exhibits a similar MMBR as Microsoft SmoothStreaming (MSS), the best-performing baseline approach, accompanied by a higher number of quality transitions. Other approaches show inferior quality due to a low MMBR or a high number of quality interruptions.

Liu et al. [134] describes an approach to downloading segments using multiple parallel TCP connections over a single network interface, when multiple clients compete for the available network capacity.

Several studies specifically look into client design aspects affecting performance and fairness issues when multiple streaming clients compete for bottleneck capacity.

Zhu et al. [232] propose an adaptation scheme which is based on a Proportional-Integral (PI) controller with the goal to combat video quality oscillations they observe when multiple video clients share a common bottleneck. The algorithm uses buffer level as input and does not explicitly take into account the network throughput. The algorithm is evaluated against a baseline approach in a wired test bed with adjustable network capacity but without cross-traffic. In the evaluation, one, two, and 36 clients compete for capacity on a shared link. It is demonstrated that the proposed approach is able to reduce the amount of quality transitions by up to 60% as compared to the baseline approach, without degrading the average quality.

The adaptation algorithm FESTIVE, proposed by Jiang et al. [98], has the goal of heuristically optimizing three performance metrics: maximizing the utilization of the available network capacity, minimizing the number and magnitude of quality transitions, and minimizing application-layer fairness between clients sharing a capacity bottleneck. FESTIVE uses the harmonic mean over a fixed number of segment downloads to estimate the available throughput. Based on this estimation, it selects a target video quality and initiates a convergence procedure, whose speed depends on the distance between the target and the current media bit rate, and the amount of quality transitions the client has already performed in the recent past. These two factors are considered jointly in order to resolve the trade off between the utilization and the amount of quality fluctuations. In addition, FESTIVE randomizes the inter-request delays in order to avoid the dependency of the adaptation trajectories on the initial state. The authors evaluate FESTIVE against several commercial players, revealing its superior performance along all three considered performance metrics. We use FESTIVE as a baseline approach for the performance evaluation presented in Chapter 6.

The experiments performed by Huang et al. [82] reveal that some of the video quality selection algorithms deployed in commercial services rapidly decrease the streamed video bit rate below their fair share when confronted with an arrival of TCP background flow. The authors observe that the problem is partially due to the TCP congestion control algorithm which decreases the size of the congestion window after the idle period caused by inter-request delays of approximately 4 seconds. Another factor is an inaccurate estimation of the achievable throughput. Finally, an overly conservative behavior of the deployed adaptation algorithms further degrades the performance.

An early work proposing a middleware framework enabling a coordination of (i) the application-layer adaptation of the individual applications, and (ii) the system-wide allocation of the shared resources is proposed by Li and Nahrstedt [124]. The authors are using a task control model for a fair and efficient resource allocation, and a fuzzy control model for mapping the allocated share to a parametrization of a particular application.

Huang and Nahrstedt [84] propose to adapt an interactive streaming service using a genetic algorithm that searches for local non-dominated Pareto optima in the QoE space including the synchronization quality, the latency, and the audio and video quality.

3.2 Low-Delay Live Streaming

As outlined in the previous section, there is a large number of recent studies focusing on adaptation algorithms for HAS that do not address the low-delay requirement. They typically consider playback buffer sizes of 10 to 30 seconds or more. In contrast, the number of studies that specifically target live or low-delay HAS is significantly smaller.

Lohmar et al. [138] compare the delay and communication overhead in Hypertext Transfer Protocol (HTTP)-based live streaming with the one in Real-Time Transport Protocol (RTP)-based systems. They observe that with HTTP the transport delay is by one segment duration larger and that the communication overhead becomes substantial for sub-second segment durations (approx. 31 kbps for one second segments).

Evensen et al. [57] study how the interaction between segment download scheduling on the application layer and the TCP congestion control dynamics can be used to improve the performance when using non-persistent HTTP connections. Such techniques, however, have limited applicability to low-delay streaming, due to the tight timing constraints. They propose two algorithms targeting a buffer size of 10 seconds, one of which is using context information in form of a location-based throughput map.

Thang et al. [196] evaluate several adaptation algorithms using buffer sizes from 6 to 20 seconds. The authors observe that the used set of representations can significantly affect the performance of individual methods, resulting in by up to a factor of 2.8 higher average video quality. They also observe that none of the selected five methods performs best in all cases but that the ranking depends on the throughput variability. However, the study was performed using only one throughput trace.

Wei and Swaminathan [214] demonstrate that the server push feature introduced in HTTP/2 can be used to support low-delay streaming by allowing a reduction in the segment duration, and thus the lower bound on the achievable delay, without suffering from the super-linear increase in the number of requests observed with HTTP/1.1. However, the study does not analyze the protocol overhead caused by response headers and the reduction in video compression rate due to the decreased Group of Pictures (GOP) size.

Le et al. [118] propose an adaptation algorithm for live streaming that compares potential adaptation trajectories for the next few segments and heuristically maximizes the video quality represented by the Just Noticeable Difference (JND) metric. The algorithm is evaluated using a maximum buffer size of 10 seconds and compared against two baseline approaches. The evaluation shows that the algorithm exhibits lower average video quality than the maximum of the two baseline methods. However, it is able to reduce the average step size of a quality transition by 32% w.r.t. the minimum of the two baseline methods. The performed evaluation used only one throughput trace.

Kupka et al. [113] compare various strategies that a HTTP-based live streaming client has w.r.t. tuning into the stream, inter-request delays, and reacting to segments that miss their playback deadline. The study is performed using a simple adaptation algorithm that bases its decision on the throughput achieved during the download of the last segment.

He et al. [72] propose a control-based cross-layer approach to adapt an interactive streaming service to the end-to-end delay.

3.3 Prediction-Based Adaptation

While many adaptive streaming clients leverage throughput averages computed in different ways over one or multiple time periods in the past, several studies explicitly create models of the throughput process or compute throughput predictions.

Mok et al. [146] consider path probing techniques to obtain throughput estimations that, however, typically requires support from the network infrastructure, server instrumentation, and/or modifying lower protocol layers. The proposed adaptation algorithm is not evaluated.

Similar in spirit to our work is the study by Liu and Lee [136], which uses predictions to match a target number of skipped segments and a minimum delay between quality transitions to control the transitions frequency. The algorithm is evaluated in a mobile network and compared against two baseline approaches. The evaluation reveals that the baseline approaches, using default configurations, require up to an order of magnitude more quality transitions in order to achieve a similar media bit rate and a similar number of skipped segments as the proposed algorithm. Unfortunately, only one network environment is used for the evaluation, and the characteristics of the observed throughput dynamics are not disclosed in the study. Also, the maximum buffer size used in the evaluation is not specified.

Yin et al. [220] study the effect of prediction errors on the performance of two adaptation approaches, rate based and Model Predictive Control (MPC), compared against a buffer based approach using synthetic throughput traces. The authors conclude that when the prediction error exceeds 25%, prediction-based approaches might exhibit worse performance than the buffer based algorithm. The used MPC algorithm is not described in the publication.

The study by Tian and Liu [197] proposes a prediction-based adaptation algorithm, where the media bit rate is selected to equal the predicted throughput times a dynamically varying adjustment factor. The proposed approach is, however, not designed to support QoE-based performance targets and has a set of configuration parameters that does not allow for a straightforward tuning of the algorithm for particular network environments.

Li et al. [128] use dynamic programming to solve a Network Utility Maximization (NUM) problem for a finite time horizon, for which a bandwidth estimation is computed based on an Exponentially Weighted Moving Average (EWMA) of the recent segment downloads. The proposed adaptation algorithm is evaluated using buffer size limits between 30 and 50 seconds.

A heuristic adaptation algorithm is proposed by Le et al. [117] that tries to satisfy constraints on future buffer levels over a finite time horizon using throughput predictions obtained using a modified EWMA model where the weights are dynamically adjusted based on the most recent relative prediction error. The algorithm is evaluated against two baseline approaches using a single throughput trace and a buffer size of 20 seconds. The evaluation reveals that the average video quality offered by the proposed algorithm is by 3% (17%) higher, the average quality transition magnitude is by 27% (9%) lower, and the number of quality transitions by 80% higher (46% lower) as compared to the two baseline approaches.

3.4 Cross-Layer Approaches

Many studies targeting adaptive streaming adopt the perspective of a streaming client, while modeling the network environment as a black box. A number of studies, however, specifically focus on video transmission over wireless networks, leveraging cross-layer techniques that jointly perform video quality selection and network resource allocation, for different types of wireless networks.

In most such studies, video quality selection is performed in a centralized way [1, 28, 35, 49, 54, 67, 80, 116, 126, 170, 193]. In contrast, we assume a client-driven approach, where every streaming client performs adaptation individually and asynchronously w.r.t. the other clients, which is inline with the HAS streaming model [50, 183].

Moreover, while these studies focus on a setting with a single base station, we consider a small cell network [26] with a dense base station deployment and a bandwidth reuse factor of 1. This setting is considered one of the candidate solutions to cope with the recently observed dramatic increase of wireless and mobile traffic.

Several works have developed joint transmission scheduling and video quality adaptation schemes following a NUM approach, where the network utility function captures some notion of fair maximization of the users' QoE.

Joseph and De Veciana [102] apply this approach to a network described by a convex feasible rate region. In wireless, this essentially corresponds to a single-cell scenario. Bethanabhotla et al. [16] solve the NUM problem using the Lyapunov drift-plus-penalty method applied to the same multi-cell wireless network model considered in our paper, where the network utility is a concave function of the long-term average video qualities. The approach couples the transmission scheduling slot duration to the video segment size, requiring large start-up delays to combat buffer underruns. In contrast, we use a control-theoretic approach to control users playback buffers, complemented by heuristics aiming at further increasing QoE. An early work considering NUM in a wireless network, using the classical congestion pricing framework, is reported by He et al. [71].

Essaili et al. [54] propose a cross-layer approach to jointly control the resource allocation in a wireless network and the video quality adaptation of the clients in a centralized network controller. The goal is to maximize the sum of the utilities, where the individual utility functions are proportional to the Peak Signal-to-Noise Ratio (PSNR) of the streamed content. The approach can be deployed with legacy streaming clients by overwriting the HTTP requests at a network proxy. In contrast to our work, the authors focus on a single cell, propose a centralized approach, and do not include means to mitigate video quality fluctuations. The conducted performance evaluation reveals that the cross-layer optimization improves the median Mean Opinion Score (MOS) by approximately 0.5.

In their work performed in the context of Wireless Mesh Networks (WMN's), Hua and Xie [81] propose to merge the individual unicast video streams on a segment-by-segment base in order to avoid redundant transmissions. Sundaram and Hua [186] extend this work by a fast handover schemes to enable seamless streaming to fast moving clients.

Analogously to cross-layer approaches, streaming clients may use context information to improve their adaptation behavior. Context may include information such as the location, the mobility pattern, or sensor data. One approach that uses location-throughput

maps to perform adaptation decisions is presented by Riiser et al. [166].

Finally, Klaue et al. [107] and Kang et al. [105] propose frameworks that leverage cross-layer information to evaluate the performance of video streaming services.

3.5 Optimal Adaptation

An approach to calculating optimal adaptation trajectories using a Markov decision process is presented by Jarnikov and Özçelebi [97]. With this approach, an optimal strategy is calculated for a given distribution function of segment download times. The objective function is a linear function giving constant penalty to playback interruptions and changes of video quality, and a reward proportional to the selected video quality. The authors perform a numerical evaluation of the approach using fixed, uniform and normal distributions of the available bandwidth. A potential limitation of this approach is that the temporal correlation of segment download times is not considered.

Zou et al. [235] assume perfect information about future throughput to compute optimal adaptation trajectories that can be used to benchmark existing algorithms and evaluate the potential for performance increase. In contrast to our work, the proposed approach does not minimize the video quality fluctuations.

3.6 TCP Throughput Prediction

Since having accurate throughput predictions is beneficial for a number of applications running over TCP, there are dedicated studies that address this subject, see He et al. [70] for an overview. Since, however, the main application of TCP was for a long time bulk data transfer, many of those studies predict throughput averaged over much longer time intervals than required for low-delay streaming [70, 152]. He et al. [70] observed that time series prediction methods perform quite well on the time scale of 50 seconds (Root Mean Square Relative Error (RMSRE) is less than 0.4 for about 90% of studied traces), but the accuracy strongly varies across studied network paths. Evaluations are often limited to wired networks which, however, are not subject to channel state dynamics and effects caused by the data link layer to the extent seen with wireless technologies [70, 144, 152]. Some of the developed approaches use information available only at the sender, or they require cross-layer information, or additional path measurements that need to be supported by the other end-point [144, 152]. In addition, some analytical models target specific TCP flavors making their performance uncertain given recently developed variants [152].

A number of studies investigate the prediction of the available bandwidth in wireless networks [176]. This information might be considered to improve the accuracy of the end-to-end TCP throughput prediction, especially in cases where the access link constitutes the bottleneck. However, it requires a cross-layer interface to the lower protocol layers, restricting the set of targeted platforms. We remark that while Linux supports cross-layer communication with the data link layer using the radiotap headers[1], conveying

[1] http://www.radiotap.org

them on other platforms, including Internet browser environments, is challenging to impossible.

Chapter 4

Notation

In this section, we will introduce the basic notation used throughout the paper. A summary is provided in Table 4.1. Some of the approaches introduced in the subsequent chapters require minor extensions of the basic notation, these extensions will be described in the corresponding chapters.

The duration of the video content contained in one segment is assumed to be constant and is denoted by $\tau \in \mathbb{R}_+$. We use index $i \in \{0, 1, \ldots, n-1\}$, $n \in \mathbb{N}_+$, to indicate a particular segment in a stream.

We denote the finite set of available video representations by $\mathcal{R}$, indexed by $j \in \{0, 1, \ldots, m-1\}$, $m = |\mathcal{R}|$. We assume that $\mathcal{R}$ only contains representations feasible for the considered user. (A representation might be infeasible if its playback requires features not supported by the user device or if its properties are excluded by configuration. Unsupported features might include, e.g., 3D, particular video codecs, high resolutions, etc. Properties excluded by configuration might include, e.g., low video quality, explicit content, etc.)

For a segment i from representation j, we denote by $s_{ij} \in \mathbb{N}_+$ its size in bits, and by $r_{ij} = s_{ij}/\tau$ its Mean Media Bit Rate (MMBR). Note that both may vary across segments of the same representation if Variable Bit Rate (VBR) encoding has been used to generate the media data. We denote by $r_j = \frac{1}{n}\sum_{i=0}^{n-1} r_{ij}$ the MMBR of representation j. We write $r_{\min} = \min_{j=0}^{m-1} r_j$ and $r_{\max} = \max_{j=0}^{m-1} r_j$ for the lowest and highest MMBR among the representations in $\mathcal{R}$. Whenever the representation is clear from the context or otherwise unambiguously specified, we might omit the index j. Thus, s_i might, e.g., denote the size of a downloaded segment i, while $r_i = s_i/\tau$ will then denote its MMBR. We write $r_i^{\uparrow}$ and $r_i^{\downarrow}$ to denote the MMBR of the next higher and the next lower representation than the one that was used to download segment i. Finally, we denote by $r(t)$ the MMBR of the segment being downloaded by the user at time t.

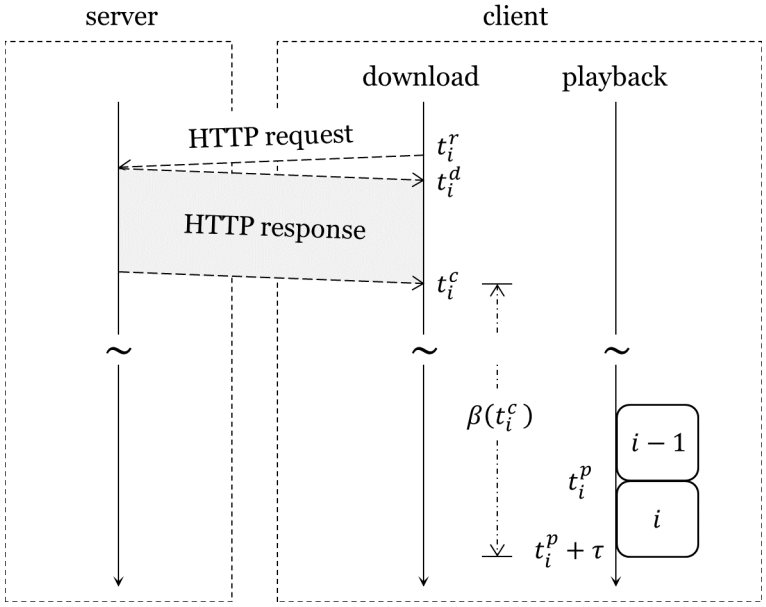

Figure 4.1: Illustration of the basic time-related notation

We use the following non-negative real-valued variables to denote continuous time in seconds. An illustration is provided in Figure 4.1. t_i^r denotes the time when the request to download segment i is sent by the user. t_i^d (t_i^c) denotes the time when the first (last) bit of segment i is received by the user. t_i^p denotes the time when the playback of segment i is started.

We denote by $\beta(t)$ the level of the user's playback buffer at time t, expressed in seconds of stored video content. More precisely, it is defined as the time until the playback deadline of the first segment whose download is not yet completed:

$$\beta(t) = \max\{t_i^p \mid t_i^c \leq t\} + \tau - t\,. \tag{4.1}$$

Note that the maximum transport latency for segment i is given by $\beta\left(t_i^r\right)$. We denote by $\beta_{\min}\left(t_1, t_2\right) = \min_{t\in[t_1,t_2]} \beta(t)$ the minimum buffer level reached during a time interval $[t_1, t_2]$.

We remark that in the context of video streaming the buffer level is typically measured in seconds of stored content, not in the volume of bytes representing this content. The reason is that from the application layer perspective, it is the duration of the stored content that influences the performance by indicating how strongly the performance may be impacted by a sudden throughput decrease. In addition, for adaptive streaming, the stored data volume is not proportional to the corresponding content duration, due to the potentially different encoding rates of the stored video segments. We do not consider scenarios where the size of the playback buffer is constrained by the available storage space since, due to the relatively small considered buffer levels of at most several tens of seconds, and due to the large amounts of storage available on most device platforms, those case are rare. Moreover, such constraints can be easily incorporated into the presented approaches.

We denote the total relative duration of interrupted playback until time t by $\Sigma(t) \in [0,1]$. $\Sigma(t)$ is computed as the total absolute duration of interrupted playback until time t, divided by t. As described in Section 2.1.4, in a Video on Demand (VoD) streaming session, and in a live streaming session with a soft latency bound (that is, the latency is allowed to increase), this corresponds to the total relative time spent in rebuffering. In a live streaming session with a hard latency bound, where delayed segments have to be skipped, this corresponds to the total number of segments skipped before the time t, divided by the total number of segments, whose playback deadline is less than or equal to t.

When segment i is downloaded and played in representation j different from the representation of the previous successfully downloaded segment, a quality transition occurs. The fraction of segments that were successfully downloaded in a different quality than their predecessors until time t is denoted by $\Omega(t) \in [0,1]$.

We denote by $\rho\,(t_1, t_2)$ the mean application layer throughput in the time interval $[t_1, t_2]$. Note that the traffic generated by a streaming client might contain inter-request periods. In the case of VoD they arise when the playback buffer level has reached its maximum value and the client has to wait prior to starting the download of the next segment. In the case of live streaming, the client might have to wait until the next segment becomes available at the server. When computing average application layer throughput, we exclude the inter-request periods in order to obtain an estimate of the throughput that was actually achieved during the data transmission. In particular, we first define the segment throughput of a downloaded segment i by $\rho_i = s_i/(t_i^c - t_i^r)$. For incomplete downloads, t_i^c must be replaced by the time when the download was interrupted, while s_i must be replaced by the number of actually downloaded bytes. Note that defined this way, segment throughput accounts for the round-trip time. We then define the application layer throughput as the mean segment throughput in the time interval $[t_1,\, t_2]$:

$$\rho\,(t_1, t_2) = \frac{\sum_{i=0}^{n-1} \rho_i \cdot |[t_i^r, t_i^c] \cap [t_1, t_2]|}{\sum_{i=0}^{n-1} |[t_i^r, t_i^c] \cap [t_1, t_2]|}\,. \tag{4.2}$$

Here, $|[a,b]| = b - a$ denotes the length of an interval $[a,b]$. For time intervals, for which the denominator equals 0, $\rho\,(t_1, t_2)$ is not defined. Note that effectively, the sum needs only be computed over $\{l_1, \ldots, l_2\}$, where l_1 corresponds to the last segment requested before t_1, l_2 is the first segment whose download was completed after t_2. For time intervals, for which the denominator equals 0, $\rho\,(t_1, t_2)$ is not defined. To simplify the presentation, we will occasionally use $\rho(t)$ to denote instantaneous throughput, which roughly corresponds to $\rho\,(t_1, t_2)$, with $t \in [t_1, t_2]$, $t_2 - t_1$ small.

$\tau \in \mathbb{R}_+$	Video segment duration in seconds
$n \in \mathbb{N}_+$	Number of segments in the stream
$i \in \{0, 1, \ldots, n-1\}$	Segment index
$\mathcal{R}$	Set of available video representations
$m = \lvert\mathcal{R}\rvert \in \mathbb{N}_+$	Number of available representations
$j \in \{0, 1, \ldots, m-1\}$	Representation index
$s_{ij} \in \mathbb{N}_+$	Size in bits of segment i from representation j
$r_{ij} = s_{ij}/\tau$	MMBR of segment i from representation j
$s_i \in \mathbb{N}_+$	Size in bits of a downloaded segment i
$r_i = s_i/\tau$	MMBR of a downloaded segment i
$r_j = \frac{1}{n}\sum_{i=0}^{n-1} r_{ij}$	MMBR of representation j
$r_{\min} = \min_{j=0}^{m-1} r_j$	Representation with the lowest available MMBR
$r_{\max} = \max_{j=0}^{m-1} r_j$	Representation with the highest available MMBR
$r_i^{\uparrow}$ and $r_i^{\downarrow}$	MMBR of the next higher (lower) representation than the one used to download segment i
$r(t)$	MMBR of the segment being downloaded at time t
$t_i^r \in \mathbb{R}_+$	Time when the request for segment i is sent
$t_i^d \in \mathbb{R}_+$ and $t_i^c \in \mathbb{R}_+$	Time when the first (last) bit of segment i is received
$t_i^p \in \mathbb{R}_+$	Playback deadline of segment i
$\beta(t)$	Playback buffer level in sec. at time t, defined in (4.1)
$\beta_{\min}(t_1, t_2)$	Minimum buffer level in the time interval $[t_1, t_2]$
$\Sigma(t) \in [0, 1]$	Relative duration of interrupted playback until t
$\Omega(t) \in [0, 1]$	Relative number of representation transitions until t
$\rho_i = s_i/(t_i^c - t_i^r)$	Segment throughput of segment i
$\rho(t_1, t_2)$	Mean throughput in $[t_1, t_2]$, defined in (4.2)
$\rho(t)$	Instantaneous throughput at time t

Table 4.1: Basic notation

Chapter 5

Joint Transmission Scheduling and Quality Selection in Dense Wireless Networks

To cope with the enormous traffic increase that has been observed over the last years and that will most probably continue into the future [37], it is necessary to increase the density of the deployed wireless infrastructure. Hence, in this chapter, we present our design of an efficient system that supports a large number of unicast video streaming sessions in a dense wireless access network, such as a small cell network. To achieve our goals, we use a cross-layer approach – we jointly consider the two problems of wireless transmission scheduling and video quality adaptation, using techniques inspired by the robustness and simplicity of Proportional-Integral-Derivative (PID) controllers. We show that the control-theoretic approach allows to efficiently utilize the available wireless resources, providing a high Quality of Experience (QoE) to a large number of users.

5.1 Introduction

The traffic from wireless and mobile devices will exceed the traffic from wired devices by 2018. By that time, it will account for 61 percent of the total Internet traffic. And by

far the largest part of it will be video content [37]. It is well understood that the current trend of cellular technology (e.g., Long-Term Evolution (LTE) [174]) cannot cope with the traffic increase caused by the various new video services, unless the density of the deployed wireless infrastructure is increased correspondingly. In fact, throughout the history of wireless networks, throughput gains resulting from increased network density exceeded the gains from individual other factors by an order of magnitude [26].

This motivates the recent flurry of research on massive and dense deployment of Base Station (BS) antennas, either in the form of "massive Multiple Input Multiple Output (MIMO)" solutions (hundreds of antennas at each cell site) [168] or in the form of very dense small cell networks (multiple nested tiers of smaller and smaller cells, possibly operating at higher and higher carrier frequencies) [26]. If supplied with sufficient storage capacity that would allow to cache frequently demanded content, these technologies can also help reducing the load on the backhaul (i.e., the wired network connecting the access network to the Internet), which have recently become a bottleneck in cellular networks [63].

This development confronts video streaming technologies with the challenge to deal with highly dynamically varying network conditions that are typical for wireless links. Even a static user in an indoor residential or office Wireless Local Area Network (WLAN) is typically exposed to interference, fading effects, and cross-traffic. The link quality fluctuations further increase in the case of mobile users.

Consequently, in our work we focus on designing an efficient mechanism to support a large number of parallel streaming sessions in a wireless access network, such as a small cell network. In order to maximize the performance, we use a cross-layer approach – we jointly consider the two problems of wireless transmission scheduling and video adaptation.

To reach our goals, we apply the technology of Proportional-Integral-Derivative (PID) control; inspired by the analytical tractability of the approach, complemented by its powerful features. A PID controller is typically used to stabilize a dynamic system around a given target state. Its strength lies in the ability to do so in the presence of model uncertainties (that is, the system parameters are not completely known and might be time-varying) and disturbances (unknown, potentially random, inputs to the system). In our case, we use it to stabilize the clients' playback buffers around configured target values, in the presence of dynamically changing network conditions due to user arrivals and departures, mobility, and fading effects.

We evaluate the developed approach by means of simulations in different deployment scenarios, such as long-term users with low user churn, short-term users with high user churn, and a mix of short-term and long-term users. We use QoE-related performance metrics such as the total rebuffering time, the average video quality, the video quality fluctuations, the start-up delay, and the media bit rate fairness.

5.2 System Model and Notation

In the following, we extend the basic description of a HTTP-Based Adaptive Streaming (HAS) system from Section 2.1.4, and the basic notation from Chapter 4 (summarized in Table 4.1), to include a multi-user setting and a sufficiently detailed wireless network

model, required by the cross-layer approach presented in this chapter. Wherever necessary, symbols defined in Chapter 4 are used with an additional subscript u referring to the user. A summary of the extensions is provided in Table 5.1, which also includes the notation introduced in the subsequent sections.

5.2.1 Streaming Model

We consider a set of users, where each user wants to watch a video file from a set of provided files. Recall that in accordance with the HAS model [183], each video file is segmented in chunks of τ seconds duration, and each segment is encoded in several representations, each representation providing a different video quality and thus a different MMBR. Further, each segment starts with a random access point of the stream, thus allowing a video client to concatenate segments from different representations during the playback. A video client sequentially issues HTTP GET or GET RANGE requests to download the individual segments. The meta information about the available segments and representations is downloaded by the client prior to starting the streaming session, e.g., in form of an Extensible Markup Language (XML) file, called the Manifest or the Media Presentation Description (MPD). This technology has been standardized in the open standard MPEG-DASH [50].

After a segment is downloaded, it is stored in the playback buffer until the playback of the previous segment has been completed. If a segment is not downloaded until its playback deadline, a buffer underrun occurs causing a playback interruption, typically followed by a rebuffering period, during which the playback is halted. At the start of the streaming session, the video client may delay the start of the playback in order to prebuffer a certain amount of video (start-up delay).

The goal of a video client is to maintain a high QoE by adapting the selected segment representations to the available network throughput. Among the main factors influencing the QoE are (1) the total rebuffering duration, (2) the average video quality, (3) the number of quality transitions, and (4) the duration of the start-up delay. The exact nature of QoE for adaptive streaming is an ongoing research problem, see also Section 2.2.

5.2.2 Distributed Cross-Layer Design

We assume that the users are connected to a wireless network, consisting of a set of BS's linked with a central network controller. While the individual streaming clients issue requests for the video segments asynchronously, each at its own pace, the network controller performs transmission scheduling for all the BS's involved. Note the different time scales of these two types of processes. While the segments are requested every few seconds, the transmission scheduling decisions must be met at a much faster time scale, tens or hundreds of milliseconds, as described in more details further below.

In order to jointly optimize the two processes of quality selection and transmission scheduling, an information exchange is required between the individual streaming clients and the network controller. We assume that there is a control plane channel that allows the clients to pass to the network controller (via the BS's) their throughput demands at the beginning of each video segment download. We do not assume the existence of a

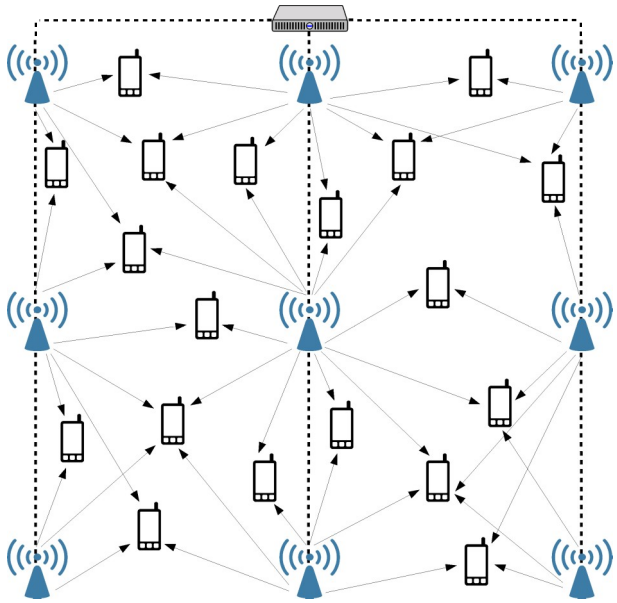

Figure 5.1: Small cell network model. A user node can be served by multiple serving nodes. Transmission scheduling is coordinated by a centralized network controller.

reverse control channel; the streaming clients are estimating the allocated network capacity by measuring their throughput. If such a channel exists, however, it may enhance the performance by supplying the users with a more exact information on their capacity share. We remark that there are ongoing standardization efforts towards enabling coordinated operation of HTTP-based streaming clients that may provide the required control channels [52].

The approach presented in this chapter is a cross-layer approach that requires a cooperating among the individual layers of the protocol stack. In particular, the video quality selection process is taking place at the application layer of the client's protocol stack, while the transmission scheduling process resides at the PHY/MAC layers of the BS's and the network controller. The interaction of the proposed approach with the transport layer is discussed in Section 5.3.

The BS's either store cached video files, or have a (wired or wireless) connection to video server(s), which we assume not to be the system bottleneck. In general, some user nodes might also participate in the distribution of video data; therefore, we generally refer to BS's and user nodes serving as streaming sources as serving nodes. See Figure 5.1 for an illustration.

5.2.3 Wireless Network Model

We formally define the network as a bipartite graph $G_t = (U_t, S, E_t)$, where t is the time index, U_t denotes the set of users, S denotes the set of serving nodes, and E_t contains edges for all pairs $(s, u) \in S \times U_t$ for which there exists a potential transmission link

between s and u at time t. We denote by

$$\mathcal{N}_t(u) = \{s \in S : (s,u) \in E_t \wedge \rho_{su}(t) \geq \rho_{\min}\} \tag{5.1}$$

the neighborhood of user u at time t. Similarly,

$$\mathcal{N}_t(s) = \{u \in U_t : (s,u) \in E_t \wedge \rho_{su}(t) \geq \rho_{\min}\} \ . \tag{5.2}$$

Here, $\rho_{su}(t)$ denotes the link rate of the wireless link between serving node s and user u, while $\rho_{\min}$ is a technology dependent minimum link rate. In the following, we omit the index t to simplify the notation.

Although the approach presented in this paper works with different kinds of wireless network models, we will focus on the wireless network model used in [16], as described in the following. The wireless channel for each link (s,u) is modeled as a frequency and time selective underspread fading channel [202]. Using Orthogonal Frequency-Division Multiplexing (OFDM), the channel can be converted into a set of parallel narrowband sub-channels in the frequency domain (subcarriers), each of which is time-selective with a certain fading channel coherence time. The small-scale Rayleigh fading channel coefficients can be considered as constant over time-frequency "tiles" spanning blocks of adjacent subcarriers in the frequency domain and blocks of OFDM symbols in the time domain.

We assume that the transmission scheduling decisions performed at the central network controller are met according to the underlying PHY and MAC air interface specifications. For example, in an LTE setting, users are scheduled over resource blocks which are tiles of 7 OFDM symbols x 12 subcarriers, spanning (for most common channel models) a single fading state in the time-frequency domain.

Nevertheless, it is unreasonable to assume that the rate can be adapted on each single resource block. As a matter of fact, rate adaptation is performed according to some long-term statistics that capture the large-scale effects of propagation, such as distance dependent path loss and interference power. It is well-known that with a combination of rate adaptation and hybrid ARQ, a link rate given as the average with respect to the small-scale fading of the instantaneous rate function, for given large scale pathloss coefficients and interference power, can be achieved. This averaging effect with respect to the small scale fading is even more true when massive MIMO transmission is used, thanks to the fact that, due to the large dimensional channel vectors, the rate performance tends to become deterministic [85, 168]. Therefore, in our treatment we shall use the link rate function given by eq. (5.3).

We assume that the serving nodes transmit at a constant power, and that the small cell network makes use of universal frequency reuse, that is, the whole system bandwidth is used by all serving nodes. We further assume that every user u, when decoding a transmission from a particular serving node $s \in \mathcal{N}(u)$, treats the inter-cell interference as noise. Under these system assumptions, the maximum achievable rate during a scheduling time slot t_k for link $(s,u) \in E$ is given by

$$\rho_{su}(t_k) = W \cdot \mathbb{E}\left[\log\left(1 + \frac{P_s g_{su}(t_k)\,|f_{su}|^2}{1 + \sum_{s' \in \mathcal{N}(u)\setminus\{s\}} P_{s'} g_{s'u}(t_k)\,|f_{s'u}|^2}\right)\right] , \tag{5.3}$$

where P_s is the transmit power of the serving node s, f_{su} is the small-scale fading gain from serving node s to user u, $g_{su}(t_k)$ is the slow fading gain (path loss) from serving node s to user u, and W is the system bandwidth.

Consistently with the current wireless standards, we consider the case of intra-cell orthogonal access. This means that each serving node s serves its neighboring users $u \in \mathcal{N}(s)$ using orthogonal Frequency Division Multiple Access (FDMA)/Time Division Multiple Access (TDMA). We denote by $\alpha_{su}(t_k)$ the fraction of time/spectrum serving node s uses to serve user u in scheduling time slot t_k. It must hold $\sum_{u \in \mathcal{N}(s)} \alpha_{su}(t_k) \leq 1$ for all $s \in S$. The throughput of user u in time slot t_k is then given by

$$\rho_u(t_k) = \sum_{s \in \mathcal{N}(u)} \alpha_{su}(t_k) \rho_{su}(t_k) \,. \tag{5.4}$$

The underlying assumption, which makes this rate region achievable, is that the serving node s is aware of the slowly varying path loss coefficients $g_{su}(t_k)$ for all $u \in \mathcal{N}(s)$, such that rate adaptation is possible. This is consistent with the rate adaptation schemes currently implemented in LTE and IEEE 802.11 [147, 151, 174].

5.3 Interaction with Transport Protocols

In this chapter, we are specifically focusing on optimizing the operation of the wireless segment of the access network. In general, effects introduced by an end-to-end transport protocol with congestion avoidance and congestion control functionality, such as Transmission Control Protocol (TCP), might result in a discrepance between the allocated link rate and the effective throughput, especially if the bottleneck is outside of the access network, reducing the performance of approaches such as the one presented here. In our work, we assume that user nodes can be served at the rate allocated by the wireless network without a "starvation" caused by core network bottlenecks. This assumption is supported by the current trend towards increasing the locality of video content, e.g., by pushing Content Delivery Network (CDN) nodes closer to the access networks [121], or by temporarily or permanently storing the requested video in or close to the access network [63, 177, 190].

It is also worth noting that we propose an approach aiming at jointly controlling the resource allocation performed by the MAC/PHY layers of the wireless access network, *and* the application-layer demand. Thus, in the absence of core network bottlenecks, an additional end-to-end rate control is not required. Therefore, it is sufficient to deploy a transport protocol with reduced or even without congestion avoidance and congestion control functionality [23, 200].

Finally, we would like to remark that the presented scheme deploys heuristics to deal with (i) the asynchrony of decisions performed by the network controller and individual users, and (ii) their different timelines. These heuristics introduce a certain robustness w.r.t. the exact throughput achieved in a particular scheduling time slot.

$u \in U$	A user and the set of users
$s \in S$	A serving node and the set of serving nodes
(s, u)	The transmission link from serving node s to user u
$E \subseteq S \times U$	Set of potential transmission links
$\mathcal{N}(u) \subseteq S$	Set of neighbors of user u, defined in (5.1)
$\mathcal{N}(s) \subseteq U$	Set of neighbors of serving node s, defined in (5.2)
t_k	k-th scheduling time slot
$\rho_{su}(t_k)$	Link rate of (s, u) in time slot t_k, def. in (5.3)
$\rho_{\min}$	Technology dependent minimum link rate
$\alpha_{su}(t_k)$	Fraction of time/spectrum allocated to (s, u) in t_k
$\rho_u(t_k)$	Throughput of user u in time slot t_k, def. in (5.4)
$K_{p,u}$, $K_{d,u}$, $K_{i,u}$	PID controller parameters of user u, defined in (5.7)
$g_{\max}$	Upper bound on the output of the saturated controller, defined in (5.10)
$g_{i,\max}$	Upper bound on the error integral, defined in (5.13)
$\omega_u(t)$	PID controller output for user u, defined in (5.8)
$\tilde{\omega}_u(t_1, t_2)$	PID controller output during $[t_1, t_2]$ for a sampled system, defined in (5.15)

Table 5.1: Notation extensions for JINGER

5.4 JINGER — Joint Scheduling and Quality Selection Scheme

In this section we present a partially distributed mechanism consisting of two components: video quality selection, performed at the application layer of each streaming client independently and asynchronously, and wireless transmission scheduling, performed at the MAC/PHY layers of the central network controller at equidistant time intervals.

5.4.1 General Idea

In the setting we consider, the buffer level dynamics can be approximated in continuous time by the following system of differential equations:

$$\dot{\beta}_u(t) = \frac{\rho_u(t)}{r_u(t)} - 1, \quad \forall u \in U\,. \tag{5.5}$$

Here, $r_u(t)$ is the MMBR of the video segment downloaded by user u at time t, while $\rho_u(t)$ is the instantaneous throughput (for the moment, we approximate the data transfer by a continuous process). Note that $\rho_u(t)/r_u(t)$ is the rate at which the buffer is filled by the arriving video data, and -1 is the rate at which the buffer is emptied by the playback.

The general idea behind the approach presented in the following is to apply PID control to the playback buffer levels of the individual clients by controlling their filling rates $\rho_u(t)/r_u(t)$. Note that a standalone streaming client is typically only able to control the media bit rate $r_u(t)$. In our approach, we assume that we are able to control both the media bit rate and the throughput $\rho_u(t)$. The challenge lies in the requirement to control them in an asynchronous and distributed way, and in discrete time. We will describe the challenges in more details at the end of this section. In the following, we describe how we apply a PID controller to control the system described by (5.5).

Let us consider a general continuous-time dynamic system described by the following equations

$$\begin{cases} \dot{x}(t) = f_1\left(x(t)\right) + z(t) \\ y(t) = f_2\left(x(t)\right) \end{cases}, \tag{5.6}$$

where $x(t)$ is the system state at time t, $z(t)$ is the input to the system, $y(t)$ is the system's output, and $f_1(\cdot)$ and $f_2(\cdot)$ are some functions. Assume that we want to stabilize system's output $y(t)$ around a certain target value y^*. From the control theory we know that for many systems this can be achieved in an efficient way by setting their input using a PID controller, that is, by setting

$$z(t) = K_p e(t) + K_d \dot{e}(t) + K_i \int_0^t e(\xi) d\xi \,, \tag{5.7}$$

where the error function $e(t) = y(t) - y^*$ is the deviation of system output $y(t)$ from the target value y^*, and K_p, K_d, and K_i are the controller parameters.

In order to apply the PID controller to control the playback buffer levels of the individual clients whose dynamics are governed by (5.5), we proceed as follows. We define the system state as the vector of the playback buffer levels $\beta(t) = \big(\beta_u(t), u \in U\big)$, representing the duration of the stored video content in seconds (defined in (4.1)). We define the system output as identical to the system state.

The specific system in (5.5) that we consider can be mapped to the general system equation (5.6) by defining $f_1\left(\beta(t)\right) \equiv -1$, $f_2\left(\beta(t)\right) = \beta(t)$, and $z(t) = \big(\rho_u(t)/r_u(t),\, u \in U\big)$. In other words, our goal is to stabilize users' buffer levels $\beta(t)$ around a target value β^* by setting the system's input $\rho_u(t)/r_u(t)$ to the output of the PID controller.

In the following analysis, in order to simplify the notation, we set the target buffer level $\beta^* = 0$. This allows us to replace the error function $e(t) = \beta(t) - \beta^*$ in (5.7) by $\beta(t)$. However, all obtained results remain valid for the general case with an arbitrary $\beta^* \in \mathbb{R}$. In the performance evaluation presented in Section 5.5, for example, β^* is set to 20 seconds.

In order to apply PID control to the buffer level dynamics (5.5), we have to set the

system's input $\rho_u(t)/r_u(t)$ to the controller output

$$\frac{\rho_u(t)}{r_u(t)} = \underbrace{K_{p,u}\beta(t) + K_{d,u}\dot{\beta}(t) + K_{i,u}\int_0^t \beta_u(\xi)d\xi}_{=:\omega_u(t)} \tag{5.8}$$

for each user $u \in U$. Thus, the output of the controller $\omega_u(t)$ sets the rate at which the playback buffer is filled by the arriving video data. Note that $\omega_u(t)$ does not have a unit. Intuitively, it is the duration of the video content in seconds received by the client per second. If we are able to control our system in this way, the buffer level dynamics will obey the following differential equation

$$\dot{\beta}_u(t) = \omega_u(t) - 1,\ \forall u \in U\,, \tag{5.9}$$

where $\omega_u(t)$ is as defined in (5.8).

In the following we present a basic result on the stability of the system (5.9). Proposition 5.1 ensures that it converges to the target buffer level β^* for arbitrary initial values, and is thus globally asymptotically stable. This result guarantees that if we control the arrival rate $\rho_u(t)/r_u(t)$ of the video data using a PID controller as in (5.8), the buffer levels will return to the target level after a disturbance caused, e.g., by a throughput change.

Proposition 5.1. *System* (5.9) *is globally asymptotically stable if and only if* $K_{d,u} \neq 1$, $\frac{K_{p,u}}{1-K_{d,u}} < 0$ *and* $\frac{K_{i,u}}{1-K_{d,u}} < 0$.

Proof. In the following, we omit the user index u.

First, we observe that for $K_d = 1$ system (5.9) degenerates and is only solvable for zero initial value: $\beta(0) = 0$.

Assuming $K_d \neq 1$ and substituting $\gamma(t) = \int_0^t \beta(\xi)d\xi$, we transform (5.9) into an equivalent system of linear differential equations

$$\begin{bmatrix}\dot{\beta}(t)\\ \dot{\gamma}(t)\end{bmatrix} = \begin{bmatrix}a & b\\ 1 & 0\end{bmatrix}\begin{bmatrix}\beta(t)\\ \gamma(t)\end{bmatrix} - \begin{bmatrix}c\\ 0\end{bmatrix}\,,$$

with $a = \frac{K_p}{1-K_d}$, $b = \frac{K_i}{1-K_d}$, and $c = \frac{1}{1-K_d}$.

We proof the claim by explicitely constructing the general solution. We distinguish two cases. Case I: $a^2 + 4b = 0$. In this case, the solution for the initial value problem $\beta(0) = \beta_0$ is given by

$$\beta(t) = \big(\beta_0 + (0.5\beta_0 - 1)at\big)e^{\frac{1}{2}at}\,.$$

It converges to 0 for $t \to \infty$ if and only if $a < 0$. For case II: $a^2 + 4b \neq 0$, we obtain

$$\beta(t) = \frac{d\beta_0 - a\beta_0 + 2c}{2d}e^{0.5(a-d)t} + \frac{\beta x_0 + a\beta_0 - 2c}{2d}e^{0.5(a+d)t}\,,$$

with $d = \sqrt{a^2 + 4b}$. It converges to 0 for $t \to \infty$ if and only if $Re(a + d) < 0$ and $Re(a - d) < 0$, where $Re(\cdot)$ denotes the real part of a complex number. This, however, is equivalent to $a < 0$ and $b < 0$, proving the claim. □

Global asymptotic stability is a very attractive property of a system. In a real-world deployment, however, it is not always possible to set the buffer filling rate $\rho_u(t)/r_u(t)$ on the left-hand side of (5.8) such that the equality holds, due to three issues. In the following, we list these issues, along with references to the sections where we address them.

- The buffer filling rate $\rho_u(t)/r_u(t)$ can be only set to the values from the interval $\left[0, \frac{\rho_{u,\max}}{r_{u,\min}}\right]$. Here, $\rho_{u,\max}$ is the maximum throughput that can be allocated to the user u by the network, and $r_{u,\min}$ is the smallest available media bit rate. The lower bound 0 is attained if the download is paused. The upper bound is attained when the user downloads the representation with the lowest MMBR at the maximum available throughput. This means that controller gain values outside of this region cannot be applied to the system. Such a controller with a limited gain is called saturated. It might become unstable due to the problem called integral windup. We describe this problem in more details and present our solution to it in Section 5.4.2.

- In a real deployment scenario, both $\rho_u(t)$ and $r_u(t)$ cannot be adapted in continuous time, as demanded by (5.8). They can only be adjusted at certain time instants, and after that they are kept constant for a certain period of time. Such a control system is called a sampled system. Moreover, $\rho_u(t)$ (transmission scheduling) and $r_u(t)$ (quality selection) are set at different time instants, since quality selection is performed once per video segment, while transmission scheduling is performed once per scheduling time slot. Consequently, our system under study is a distributed system. We address these two issues in Section 5.4.3.

- Finally, even if an ideal PID control could be applied to stabilize the buffer levels as in (5.8), it would not be sufficient to ensure high QoE. It would allow to avoid buffer underruns but, in addition, we also would like to avoid excessive video quality fluctuations and unfairness among the individual users. From the perspective of the buffer level stability, assigning a user low throughput and low video quality stabilizes his buffer level as good as assigning him high throughput and high video quality, whereas from the perspective of QoE the second situation is clearly preferred. We will address these issues in Sections 5.4.4 and 5.4.5.

Finally, we remark that it may be reasonable to define different target levels β_u^* for different users, in order to account for their individual mobility patterns, link statistics, and QoE expectations. We may imagine a specific per-user adaptation, e.g., at the application layer, that acts on the control parameter β_u^*. In the present work we do not address this outer control, and assume a common target β^* for all users.

5.4.2 Integral Windup

As mentioned in the previous section, in practice, it is not always possible to set control variables $\rho_u(t)$ and $r_u(t)$ such that the equality (5.8) is satisfied, due to a limited link rate on the one hand, and due to a limited set of available media bit rates on the other

hand. A control system where the input variable cannot be set to the controller output whenever the controller output is particularly low or particularly high is called saturated.

Saturated PID controllers have been subject of intensive research in the last decade. Partially, the attention has been motivated by the control of robotic manipulators. One issue with saturated controllers is the so-called integral windup. For large deviations of the system state $\beta(t)$ from its target value, the unsaturated controller would apply a high positive or negative gain to bring $\beta(t)$ back to β^*. Due to the saturation, however, only a smaller gain can be applied. Thus, it takes more time to bring the state back to the target value. During this time, however, the error integral obtains a larger value than it would have had without the saturation. The result is a higher overshoot and oscillations, leading to potential instability.

To formalize the notion of saturation, we use the following notation

$$[x]_{x_{\min}}^{x_{\max}} = \begin{cases} x_{\min} & \text{for } x \leq x_{\min} \\ x_{\max} & \text{for } x \geq x_{\max} \\ x & \text{otherwise} \end{cases} .$$

With this definition, the saturated and thus more realistic (closer to being implementable) version of the system (5.9) can be written as

$$\dot{\beta}_u(t) = [\omega_u(t)]_0^{g_{\max}} - 1, \ \forall u \in U \,, \tag{5.10}$$

with $g_{\max} > 1$, and $\omega_u(t)$ as defined in (5.8).

The following proposition, which leverages ideas and results from [5] and [75], shows that for a small enough $|K_{i,u}|$, the saturated system (5.10) retains its global asymptotic stability property. To obtain the following result, we set $K_{d,u} = 0$. We remark that this is common practice in many applications. The reason behind it is that when the derivative of the system state is estimated from sampled measurements and the measurements are noisy, the derivative action amplifies the noise, introducing additional jitter in the system variables.

Proposition 5.2. *Assume $K_{p,u} < 0$, $K_{i,u} < 0$, and $K_{d,u} = 0$. Then, for every set of initial values $[\beta_{\min}, \beta_{\max}]$ with $\beta_{\min} \leq 0 \leq \beta_{\max}$ there exists a $\tilde{K}_{i,u} > 0$ such that for $|K_{i,u}| < \tilde{K}_{i,u}$ and $\beta(0) \in [\beta_{\min}, \beta_{\max}]$ the solution of the initial value problem (5.10) converges to 0 for $t \to \infty$. This property is sometimes called* semi-global practical stability*.*

Proof. Throughout the proof, we will omit the user index u.

We start by observing that with $K_p < 0$, $K_i < 0$, and $K_d = 0$, the conditions of Proposition 5.1 are fulfilled.

We proceed by substituting $\gamma(t) = K_i \int_0^t \beta(\xi) d\xi$. We obtain the equivalent formulation

$$\begin{cases} \dot{\beta}(t) = \left[K_p \beta(t) + \gamma(t)\right]_0^{g_{\max}} - 1 \\ \dot{\gamma}(t) = K_i \beta(t) \end{cases} . \tag{5.11}$$

Observe that the unique equilibrium of this system is $(0, 1)$. In order to shift the equilibrium to $(0, 0)$, we define $\tilde{\beta}(t) = \beta(t) - \frac{1-\gamma(t)}{K_p}$ and $\tilde{\gamma}(t) = \gamma(t) - 1$. We obtain

$$\begin{cases} \dot{\tilde{\beta}}(t) = \left[K_p\tilde{\beta}(t) + 1\right]_0^{g_{\max}} - 1 + K_i\left(\frac{1}{K_p}\tilde{\beta}(t) - \frac{1}{K_p^2}\tilde{\gamma}(t)\right) \\ \dot{\tilde{\gamma}}(t) = K_i\left(\tilde{\beta}(t) - \frac{1}{K_p}\tilde{\gamma}(t)\right) \end{cases}.$$

Next, we write the integral gain as $K_i = \epsilon\bar{K}_i$ and substitute the time variable $t' = \epsilon t$, where $\epsilon > 0$. We define new variables $\chi(t') = \tilde{\beta}(t'/\epsilon)$ and $\zeta(t') = \tilde{\gamma}(t'/\epsilon)$ to obtain a "fast" version of our system

$$\begin{cases} \epsilon\chi'(t') = \left[K_p\chi(t') + 1\right]_0^{g_{\max}} - 1 + \epsilon\bar{K}_i\left(\frac{1}{K_p}\chi(t') - \frac{1}{K_p^2}\zeta(t')\right) \\ \zeta'(t') = \bar{K}_i\left(\chi(t') - \frac{1}{K_p}\zeta(t')\right) \end{cases}. \tag{5.12}$$

Observe that $(0, 0)$ is the equilibrium point of (5.12). Further, observe that $(\chi(t'), \zeta(t')) \to (0, 0)$ as $t' \to \infty$ implies that $\left(\tilde{\beta}(t), \tilde{\gamma}(t)\right) \to (0, 0)$ as $t \to \infty$, and thus, stability results for (5.12) transfer to (5.10). Further, for $\epsilon \ll 1$, the system (5.12) is in the form of the standard singular perturbation [75].

Now, it is relatively straightforward to validate that the conditions of Theorem 3 in [75] are fulfilled for system (5.12), proving the claim. □

Note that this result only guarantees stability if the integral coefficient $K_{i,u}$ is small enough. However, making $K_{i,u}$ smaller than necessary might have negative impact on the convergence speed. In practice, it is difficult to compute the maximum value $K_{i,u}$ that still ensures stability. Therefore, in the following, we present an alternative approach to solve the problem of integral windup: conditional integration. With this approach, the value of the integral part of the controller is not allowed to exceed certain hard limits.

In particular, we use the equivalent form (5.11) of the saturated system (5.10), and apply a bound $g_{i,\max}$ on the error integral. We obtain

$$\begin{cases} \dot{\beta_u}(t) = \left[K_{p,u}\beta(t) + \gamma_u(t)\right]_0^{g_{\max}} - 1 \\ \dot{\gamma_u}(t) = \begin{cases} \max\left(0, K_{i,u}\beta_u(t)\right) & \text{if } \gamma_u(t) \le 1 - g_{i,\max} \\ \min\left(0, K_{i,u}\beta_u(t)\right) & \text{if } \gamma_u(t) \ge 1 + g_{i,\max} \\ K_{i,u}\beta_u(t) & \text{otherwise} \end{cases} \end{cases}. \tag{5.13}$$

The resulting controller is then given by

$$\frac{\rho_u(t)}{r_u(t)} = \left[K_{p,u}\beta(t) + \gamma_u(t)\right]_0^{g_{\max}}, \tag{5.14}$$

with $\beta_u(t)$, $\gamma_u(t)$ as defined by (5.13). The advantage of this formulation is that it accounts for the saturated gain, and is thus practically implementable, and that it limits the value of the error integral, reducing the impact of integral windup. An analytic study of the performance of this anti-windup strategy is, however, notably complex, forcing us to resort to a simulative evaluation, presented in Section 5.5.

Further potential anti-windup strategies include limiting the integral action of the controller from growing by keeping it constant whenever the controller enters saturation, adding anti-windup compensating terms to the integral action, etc. (see, e.g., [194]).

5.4.3 Sampled Distributed System

So far, we studied a system, where control decisions, that is, transmission scheduling and video adaptation, are made in continuous time. In practice, however, both control decisions take place at discrete time instances, while in between, the values of the control variables are fixed. In this section, we reformulate our approach to adapt it to this requirement.

The challenge here stems from the fact that while we may assume that transmission scheduling takes place regurlarly, at equidistant time intervals, quality selection can only take place when a user starts downloading a new video segment, which happens for each user independently and, in general, on a different time scale than transmission scheduling.

Moreover, time intervals between individual segment downloads may be subject to considerable variation over time. Whenever the buffer level of a user is in equilibrium (that is, it stays around the target level for a certain period of time), the average duration of a segment download equals the duration of the segment itself. However, when the buffer level is increasing or decreasing, the duration of a segment download might be subject to significant fluctuations. In addition, segment sizes might substantially deviate from representation averages, due to VBR encoding used by modern compression technologies, causing further variations of download times.

Let us for the moment assume that both control decisions, scheduling and adaptation take place simultaneously at some given time instants t_ℓ, $\ell \in \mathbb{N}_+$. In order for the sampled system to have the same state as the continuous system at the given time instants t_ℓ, we need to set the control variables as follows:

$$\frac{\rho_u(t)}{r_u(t)} = \underbrace{\frac{\beta_u(t_{\ell+1}) - \beta_u(t_\ell)}{t_{\ell+1} - t_\ell}}_{=:\tilde{\omega}_u(t_\ell, t_{\ell+1})} + 1\,,\ \forall t \in [t_\ell, t_{\ell+1})\,, \tag{5.15}$$

where $\beta_u(t_{\ell+1})$ is computed by solving the initial value problem defined by (5.13).

Proposition 5.3. *Assume that the conditions of Proposition 5.1 are fulfilled. Then, sampled control* (5.15) *and continuous-time saturated control with anti-windup* (5.14) *lead to identical system states at time instants t_ℓ, $\ell \in \mathbb{N}_+$.*

Proof. The claim is proven by substituting (5.15) into (5.5), integrating the right hand side, and using $\beta(t_{\ell+1})$ that solves the initial value problem (5.13). □

In a real deployment, however, $\rho_u(t)$ and $r_u(t)$ cannot be set simultaneously. Instead, we are dealing with a distributed system, where transmission scheduling and quality selection are performed independently from each other and at different time instants. Consequently, in the following two sections, we present heuristics for controlling transmission scheduling and quality selection in a distributed way.

5.4.4 Quality Selection

In the following, we present several heuristics that complement the mechanisms presented in the previous sections, so that we obtain a distributed, practically implementable ap-

proach. While this section covers the quality selection part, the subsequent Section 5.4.5 covers transmission scheduling.

Recall that with HTTP-based adaptive streaming, each video segment is available for the download in several representations. Each representation offers a different video quality and thus has a different MMBR. In the approach presented in this chapter, the client selects the video quality based on the MMBR of the corresponding representation.

The idea we leverage to organize the operation of the proposed controller in a distributed way is the following. The network on the one hand and each individual user on the other hand shall try to maintain the equality (5.15) every time they adapt their respective decision variable. The network does so at the beginning of each scheduling time slot, while each user does so when it is about to request a new video segment.

We denote by t_{iu}^r the time, when the user u is about to request the next segment i. We transform (5.15) to obtain a first version of the quality selection rule

$$\tilde{r}_{iu} = \frac{\rho_u\left(t_{iu}^r\right)}{1 + \tilde{\omega}_u\left(t_{iu}^r, t_{i+1,u}^r\right)}, \tag{5.16}$$

where $t_{i+1,u}^r$ is the time when the download of segment i would be completed if the buffer dynamics would obey (5.13), allowing to request the subsequent segment $i+1$. $t_{i+1,u}^r$ can be computed by solving

$$\beta_u\left(t_{i+1,u}^r\right) = \beta\left(t_{iu}^r\right) + \tau - \left(t_{i+1,u}^r - t_{iu}^r\right),$$

with $\beta_u(t)$ being the solution to the initial value problem defined by (5.13). Using the quality selection rule (5.16), the buffer levels at the time instants, when the users issue new segment requests, would equal those of the system defined in (5.13), given that the network throughput remains constant. Thus, the client selects the media bit rate based on (i) the network throughput, (ii) the current buffer level, (iii) the target buffer level at the end of the segment download, as determined by the output of the controller.

In order to avoid excessive quality fluctuations and provide a smooth adaptation trajectory, we extend (5.16) by using exponential moving average for the selected video quality. In addition, the user shall approximate the network throughput by a simple moving average over the past T seconds. We obtain a second version of the quality selection rule

$$\hat{r}_{iu} = (1-\alpha)\hat{r}_{i-1,u} + \alpha\frac{\rho_u\left(t_{iu}^r - T, t_{iu}^r\right)}{1 + \tilde{\omega}_u\left(t_{iu}^r, t_{i+1,u}^r\right)}, \tag{5.17}$$

where $\hat{r}_{i-1,u}$ is the previously selected video quality, and $\alpha \in (0,1)$ and $T \in \mathbb{R}_+$ are configuration parameters.

Further, since only a finite set of media bit rates is available, we round $\hat{r}_{iu}$ down to the next available value: $r_{iu} = \max\{r \in \mathcal{R}_u \mid r \leq \hat{r}_{iu}\}$. If $\{r \in \mathcal{R}_u \mid r \leq \hat{r}_{iu}\}$ is empty, the representation with the lowest available MMBR is selected. Thus, r_{iu} represents the final quality decision of user u for the segment i.

In order for the buffer level dynamics to resemble the dynamics of the system (5.13), the user would have to select the video quality $\tilde{r}_{iu}$. Instead, due to the reasons described above, she or he selects the video quality r_{iu}. In order to account for the difference

between these two values, for the case when $r_{iu} \leq \tilde{r}_{iu}$, we introduce a delay after the download of the current segment, computed as follows. If the media bit rate $\tilde{r}_{iu}$ were selected and assuming user's throughput $\rho_u\left(t^r_{iu}\right)$ would remain constant until the download is completed, the buffer level when requesting the next segment $i+1$ would be

$$\beta\left(t^r_{i+1,u}\right) = \beta\left(t^r_{iu}\right) - \tau\left(1 - \frac{\tilde{r}_{iu}}{\rho_u\left(t^r_{iu}\right)}\right). \tag{5.18}$$

If the actually selected media bit rate is smaller, however, the buffer level after the download will be larger. Consequently, we delay the subsequent request until the buffer level drops below $\beta(t^r_{i+1,u})$, as defined by (5.18). It is worth noting that since

$$\beta\left(t^r_{iu}\right) - \beta\left(t^r_{i+1,u}\right) = \tau\left(1 - \frac{\tilde{r}_{iu}}{\rho_u\left(t^r_{iu}\right)}\right) \leq \tau\,,$$

the buffer level will never drop by more than one segment duration at a time as a result of a deliberately delayed request.

In order to complete the description of the quality selection approach, we describe in the following the handling of buffer underruns and the start-up procedure.

As described in Section 5.2, during a particularly long period of low throughput a buffer underrun may occur, causing a playback interruption. In order to avoid multiple interruptions within a very short period of time, the client starts to play out a segment only after it has been fully downloaded. On the other hand, we limit the duration of the rebuffering period to the download time of one segment, that is, once we have at least one segment in the buffer, the playback is immediately restarted.

Finally, at the begin of the streaming session when no throughput information is available, the client downloads the first segment in lowest quality in order to minimize the start-up delay.

5.4.5 Transmission Scheduling

The goal of the network is, on the one hand, to provide the capacities required by the users, that is, to maintain (5.15) at the beginning of each scheduling time slot. On the other hand, it shall allocate the remaining capacity, if available, in a fair manner in order to ensure a high resource utilization and to eventually enable users to switch to a higher video quality. In order to achieve these goals, we let the network controller solve a series of linear optimization problems.

In the following, we present the scheduling process for the scheduling time slot t_k. We assume that the most recent segment requested by each user prior to the beginning of the scheduling time slot t_k has the index i. In general, each user independently requests segments at his own pace, and thus the segment indexes will be different. However, while this assumption does not restrict the generality of the results, it greatly improves the readability.

We denote by $\rho^r_{iu} = r_{iu}\left(1 + \tilde{\omega}_u\left(t_{iu}, t_{i+1,u}\right)\right)$ the throughput demand of user u. Here, r_{iu} is the quality selected by the user by applying the quality selection procedure described in the previous section.

First, the network controller maximizes the minimum fraction of each user's throughput relative to his demand, similar to the well-known maximum concurrent flow problem. At the same time, the controller tries to improve fairness by encouraging users streaming at a low quality to switch to a higher quality, if there are sufficient network resources. This is achieved by artificially raising the demand ρ^r_{iu} of the 10% of the users with the lowest demands to the 10th percentile across all users. We obtain the following optimization problem:

$$\max \min_{u \in U} \quad \frac{\rho_u(t_k)}{\max\left(\rho^r_{iu},\, \rho_{10}(t_k)\right)} \tag{\textbf{TS1}}$$

$$\text{s.t.} \quad \sum_{u \in \mathcal{N}(s)} \alpha_{su} \leq 1\,, \quad \forall s \in S \tag{C1}$$

$$\alpha_{uh} \geq 0\,, \quad \forall s \in S,\, \forall u \in U\,, \tag{C2}$$

where $\rho_{10}(t_k)$ is the 10-percentile of $(\rho_{iu}, u \in U)$. Recall that

$$\rho_u(t_k) = \sum_{s \in \mathcal{N}(u)} \alpha_{su} \rho_{su}(t_k)\,.$$

We denote the optimum value of (**TS1**) by ϑ^*.

In the second step, the network controller fixes the minimum relative allocated capacity to its optimum value ϑ^*, and maximizes the minimum allocated capacity.

$$\max \min_{u \in U} \quad \rho_u(t_k) \tag{\textbf{TS2}}$$

$$\text{s.t.} \quad \vartheta^* \leq \frac{\rho_u(t_k)}{\max\left(\rho^r_{iu},\, \rho_{10}(t_k)\right)},\ \forall u \in U \tag{C3}$$

$$\text{(C1), (C2)}\,.$$

We denote the optimum value of (**TS2**) by $\rho^*_{\min}$.

Finally, it fixes the minimum allocated relative capacity ϑ^* and the minimum allocated absolute capacity to $\rho^*_{\min}$ and maximizes the total network throughput. In order to avoid high-amplitude throughput spikes for the individual users, it limits the capacity allocated to a user to either twice the median demand across all users or twice the minimum allocated capacity $\rho^*_{\min}$, whichever is larger.

$$\max \quad \sum_{u \in U} \rho_u(t_k) \tag{\textbf{TS3}}$$

$$\text{s.t.} \quad \rho_u(t_k) \geq \rho^*_{\min},\ \forall u \in U \tag{C4}$$

$$\rho_u(t_k) \leq 2 \max\left(\rho^*_{\min},\, \rho_{50}(t_k)\right),\ \forall u \in U \tag{C5}$$

$$\text{(C1), (C2), (C3)}\,,$$

where $\rho_{50}(t_k)$ is the median demand across all users.

Since the maximum number of users per serving node is typically limited by a technology dependent value, the number of optimization variables and constraints is $\mathcal{O}\left(\max\left(|S|, |U|\right)\right)$, which can be handled very efficiently by modern linear program solvers even for large networks. Also note that most solvers allow to iteratively modify and reoptimize a model, which reduces the complexity of subsequent optimizations such as we have here.

5.5 Evaluation

In the following, we present our evaluation setting and results. All results were obtained by means of simulations. The simulation code was written in C++, we used Gurobi [68] to solve optimization problems, and we used odeint [2] to solve differential equations.

Section 5.5.1 describes the performance metrics. Section 5.5.2 describes the general setting, such as network and video parameters. Section 5.5.3 elaborates on the goal and design of the individual experiments. Finally, Section 5.5.4 presents evaluation results.

5.5.1 Performance Metrics

We use the following metrics to assess the performance of the proposed system.

Stability

A well-known issue with closed-loop control systems is their potential to become unstable, leading to high-amplitude fluctuations of the system state. Although not necessarily harmful per se, instability can have a dramatic impact on other performance metrics. We evaluate stability by means of buffer level statistics, such as the maximum buffer level overshoot and the minimum buffer level of a user during a simulation run.

Rebuffering Duration

When a client's video buffer has been drained so that the next video segment does not arrive before its playback deadline, the playback must be halted. This is often refered to as a buffer underrun. A buffer underrun is followed by a rebuffering period, where the client waits until enough video data is accumulated in the buffer to resume playback. The conditions that need to be fulfilled before the playback is resumed depend on client's rebuffering strategy. In our design, we resume playback after at least one segment is completely downloaded. In our evaluation we look at the cumulative rebuffering time a user experienced during a simulation run.

Prebuffering Duration (Start-Up Delay)

At the start of a streaming session, client's video buffer is empty, so it has to wait until enough video data is downloaded to start playback. In contrast to rebuffering, a client at this state typically does not have information about the network conditions. Especially when a user frequently starts a new streaming session, e.g., by switching TV channels, or when she or he repeatedly watches short videos, even a moderate start-up delay might severely degrade QoE and even make the user decide not to watch the video at all.

Mean Video Quality

The mean video quality is obviously a factor that dramatically influences the overall QoE, although studies have shown that it cannot be considered as a standalone QoE measure for adaptive video streaming. In our evaluation, we identify the mean video quality with the mean representation index selected during the course of a streaming

session. Assuming exponentially increasing distances between MMBR's of the individual representations, the representation index is roughly proportional to the video distortion in terms of the Peak Signal-to-Noise Ratio (PSNR) [184].

Number of Quality Transitions

Recent studies have revealed the impact of temporal quality fluctuations on the overall QoE. As a measure of quality fluctuations we use the fraction of segments played out in a different quality than the preceding segment, denoted by $\Omega \in [0, 1]$. That is, $\Omega = 0.01$ means that one segment out of 100 was played in a different quality than its predecessor. For a segment duration of 2 seconds, e.g., this would mean one quality adaptation in 200 seconds.

Note that quality transitions cannot not be completely avoided. They are necessary to compensate throughput fluctuations but also MMBR variations across segments of the same representation, stemming from the VBR encoding used by most modern video compression technologies.

MMBR Fairness

Although a selfish user might not care about that, from the perspective of a service provider and/or system designer, a fare distribution of MMBR's among clients sharing a common bandwidth resource is essential for the overall system performance. In our evaluation, we computed MMBR fairness as follows. Taking the set of MMBR's of all users in one simulation run, we defined the unfairness index as the interquartile range, that is, the distance between the 0.25 and the 0.75 quantiles. The name unfairness index accounts for the fact that higher values correspond to lower fairness or higher unfairness.

5.5.2 Evaluation Setting

In this section we describe the evaluation setting, such as video, network and controller parameters.

5.5.2.1 Video-Related Settings

For evaluation, we used a mix of 6 videos, contributed by the University of Klagenfurt [120]. For each video, we selected 6 representations, ranging from approximately 500 kbps to 4.5 Mbps, with roughly exponentially increasing distances between MMBR's. The segment duration was 2 seconds. The target buffer level of the video clients was set to 20 s. In the performance evaluation study in Section 7.4, this value is shown to be sufficient to provide good performance in wireless networks.

5.5.2.2 Network-Related Settings

The simulated network spans an area of 50 x 50 meters, covered by 25 serving nodes, distributed on a uniform grid. The duration of a scheduling time slot is set to 10 ms.

The path loss coefficients $g_{su}(t)$ between serving node s and user u are based on the WINNER II channel model [114]:

$$g_{su}(t) = 10^{-0.1\mathrm{PL}(d_{su}(t))}\,,$$

where $d_{su}(t)$ is the distance from serving node s to user u at time t, and where

$$PL(d) = A\log d + B + C\log 0.25 f_0 + \chi_{\mathrm{dB}}\,. \tag{5.19}$$

In (5.19), d is expressed in meters, the carrier frequency f_0 in GHz, and χ_{dB} denotes a shadowing log-normal variable with variance σ^2_{dB}. The parameters A, B, C and σ^2_{dB} are scenario-dependent constants. Among the several models specified in WINNER II we chose the A1 model [114], representing an indoor small cell scenario. In this case, $3 \leq d \leq 100$, and the model parameters are given by $A = 18.7$, $B = 46.8$, $C = 20$, $\sigma^2_{\mathrm{dB}} = 9$ in Line-of-Sight (LOS) condition, or $A = 36.8$, $B = 43.8$, $C = 20$, $\sigma^2_{\mathrm{dB}} = 16$ in Non-Line-of-Sight (NLOS) condition. For distances less than 3 m, we extended the model by setting $\mathrm{PL}(d) = \mathrm{PL}(3)$. Each link is in LOS or NLOS independently and at random, with a distance-dependent probability $p_l(d)$ and $1 - p_l(d)$, respectively, where

$$p_l(d) = \begin{cases} 1 & \text{if } d \leq 3 \text{ m} \\ 1 - 0.9(1 - (1.24 - 0.6\log d)^3)^{1/3} & \text{otherwise} \end{cases}\,.$$

In the following experiments, d is updated in every scheduling time slot, while the random components, χ_{dB} and p_l are updated every 5 seconds (except when d falls below 3 m, which forces the link to switch into the LOS mode immediately, to maintain a consistent setting).

Finally, links with a link rate below 2 Mbps in a particular scheduling time slot are not used for transmission during this time slot. We call the remaining links *active* links.

To visualize the resulting network conditions, Figure 5.2 shows some connectivity statistics. The left column shows statistics for users uniformly distributed over the simulated area, while the right column illustrates the case of clustered users, where users' distances to the center of the simulated area are exponentially distributed with $\lambda = 0.2\ln 4$. The top subfigures show the histogram of the number of serving nodes that can serve a client at a particular location. The subfigures in the second row show the histogram of link rates over all active links. The subfigures in the third row show the histogram of the sum of link rates for a client. The bottom subfigures show the average number of users served by a serving node, for different total numbers of users in the network, plus 10th and 90th percentiles.

5.5.2.3 Controller Parameters

All experiments are performed with varying controller parameters, and varying numbers of users. In all experiments, K_d is set to 0, due to the reasons described in Section 5.4.2. In the following, whenever appropriate, results are only provided for selected parameter values, in order to improve the readability.

For better illustration, we would like to give an intuition for the scale of the parameters. For $K_p = -0.05$, if the buffer level is below the target value by 20 seconds (as,

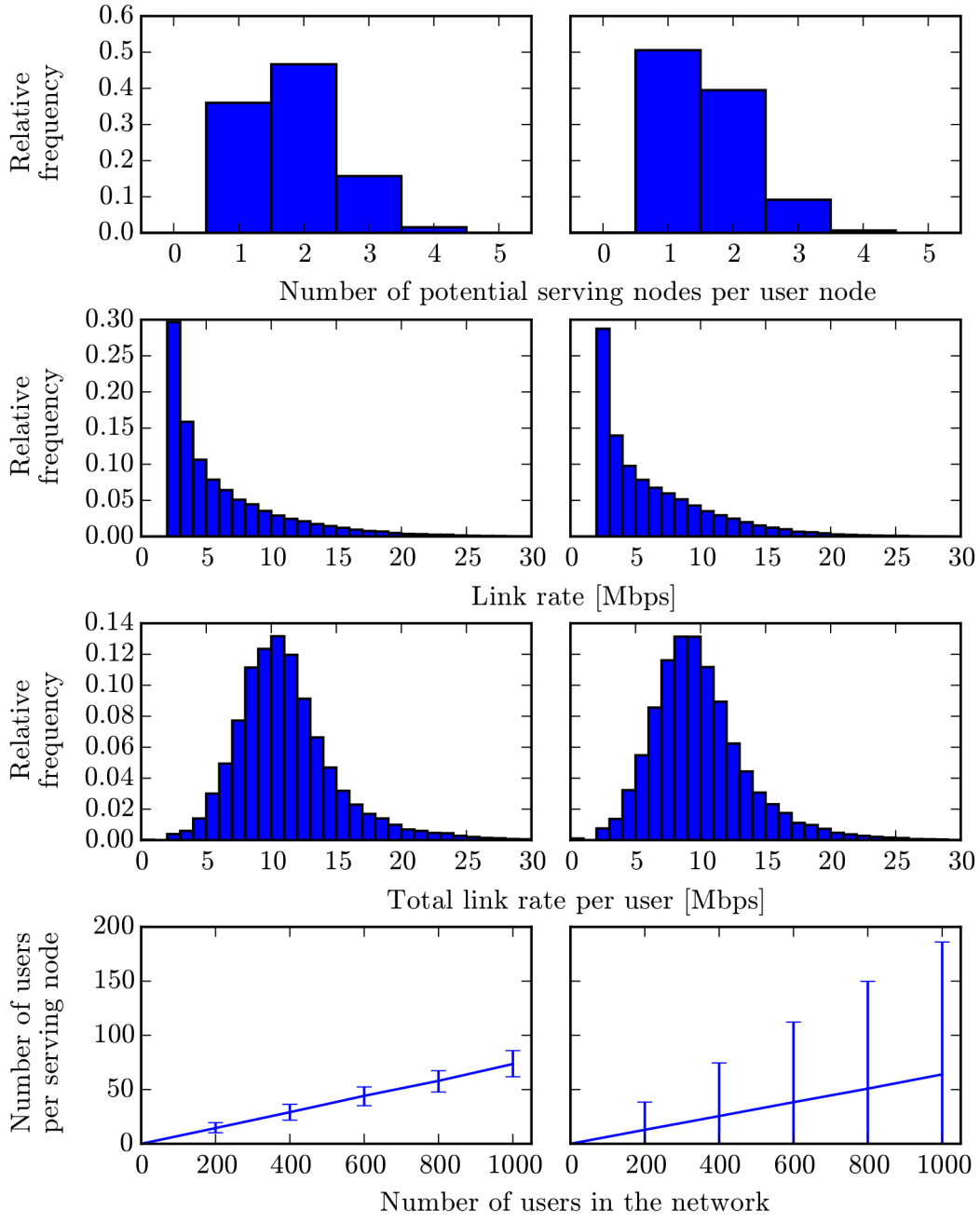

Figure 5.2: Connectivity statistics for uniformely distributed users (left) and clustered users (right). From top to bottom: number of serving nodes per client; active link rates; sum of link rates per client; average number of users per serving node for different total numbers of users in the network (with 10th and the 90th percentiles). See Section 5.5.2.2 for details.

e.g., at the beginning of a streaming session), and the integral gain is at its equilibrium value (which is 1 in our case), then the client would try to download the video at twice the playback speed. That is, in one second the client would try to download two seconds of video, which would get him closer to the target value by 1 second. For a deviation of 10 seconds, the download rate would be 150% of the playback rate, and so on.

For each controller configuration, 30 runs were performed.

5.5.3 Experimental Design

We evaluate the proposed system in three types of experiments, each of them focusing on certain deployment scenarios, such as long-term users with no user churn, short-term users with high user churn, and a mix of short-term and long-term users. The performance is then compared with the performance of the baseline approach, described below.

Each of the experiments is executed with two different user distributions, called uniform and clustered in the following. With the former, arriving users are dropped at a random location, uniformly distributed over the simulated area. With the latter, arriving users are clustered around the center of the simulated area, with exponentially distributed distance ($\lambda = 0.2 \ln 4$).

Upon joining the network, each user starts to watch a randomly selected video from a random point within the video. If she or he arrives at the end of the video, she or he continues to watch from the beginning.

Experiment 1

The first experiment is intended to analyze system's behavior under constant load (fixed number of users) and without user churn (all users are long-term users). In particular, all users arrive during an initial arrival phase at the begin of the simulation and remain in the network until its end. Thus, after the arrival phase, the number of users in the network remains constant.

The initial arrival phase starts at $t = 0$. The users arrive at a rate of 10 users per second until a predefined number of users is reached. Then, the simulation continues with a constant number of users for another 400 seconds.

Experiment 1*

In order to compare the performance with a baseline approach, we rerun experiment 1 with the following transmission scheduling and quality selection. In each scheduling time slot, every user is receiving data form exactly one serving node, namely the one with the highest Signal-to-Interference-plus-Noise Ratio (SINR). (This is not always the closest serving node, due to the random component in the pathloss.) Further, the client selects the video representation with the highest media bit rate that is still below the average throughput from the last 5 seconds.

Experiment 2

In this setting, the goal is to analyze the system's performance under continuous user churn. That is, users continuously join and leave the network. As in experiment 1, there is an initial arrival phase, during which users arrive at a rate of 10 users per second until a certain number of users is reached. After that, 'old' users leave the network, while new users join it, at a rate of 2 users per second.

Experiment 3

This experiment is intended to study system's behavior under constantly increasing load, in a deployment scenario with both short-term and long-term users. Here, new users continuously join the network at a constant rate of 2 users per second, and remain active until the end of the simulation run.

5.5.4 Evaluation Results

In this section, we present the evaluation results for the four types of experiments described in the previous section. But beforehand we would like to illustrate the system dynamics based on one example run. Figure 5.3 shows the dynamics of one user during a run of experiment 3, with $K_p = -0.05$, $K_i = -0.00001$, $K_d = 0$, $g_{i,\max} = 0.1$. The experiment runs for 500 seconds, that is, in the end there are 1000 users in the network. The plot shows the first user in the network, who starts to stream at second 0. The top subfigure shows the accumulated link rate from all neighboring serving nodes. The second subfigure shows the network throughput allocated to the user. Note that in the scheduling time slots that correspond to inter-request delays, no resources are allocated to the user and thus his throughput drops to 0. The third subfigure shows the selected video representation, the fourth subfigure shows the buffer level. Finally, the bottom subfigure shows the total time spent in rebuffering.

5.5.4.1 Stability

One of the issues that needs to be taken care of when designing a closed-loop controller is system's stability. In order to study stability, we analyze the buffer level statistics of each user during each of the simulation runs. In particular, we look at the maximum and minimum buffer levels, where the maximum and minimum operators are first applied to the traces of individual users, then to the resulting per user values, and finally to the whole set of runs for a specific configuration. In the following, we present the results for experiment 1, with a uniform distribution of users across the simulated area. The results for the other settings are consistent with the presented findings and are therefore omitted here. Finally, in order for the results not to be biased by the system behavior during the initial arrival phase, we remove the initial arrival phase and the subsequent 100 seconds from each trace.

The first and the second rows in Figure 5.4 show the maximum buffer level overshoot (that is, the maximum difference between user's buffer level and the target buffer level). As expected, if the integral gain coefficient is large, as compared to the proportional gain coefficient, the system tends to become unstable. At the same time, however, we observe

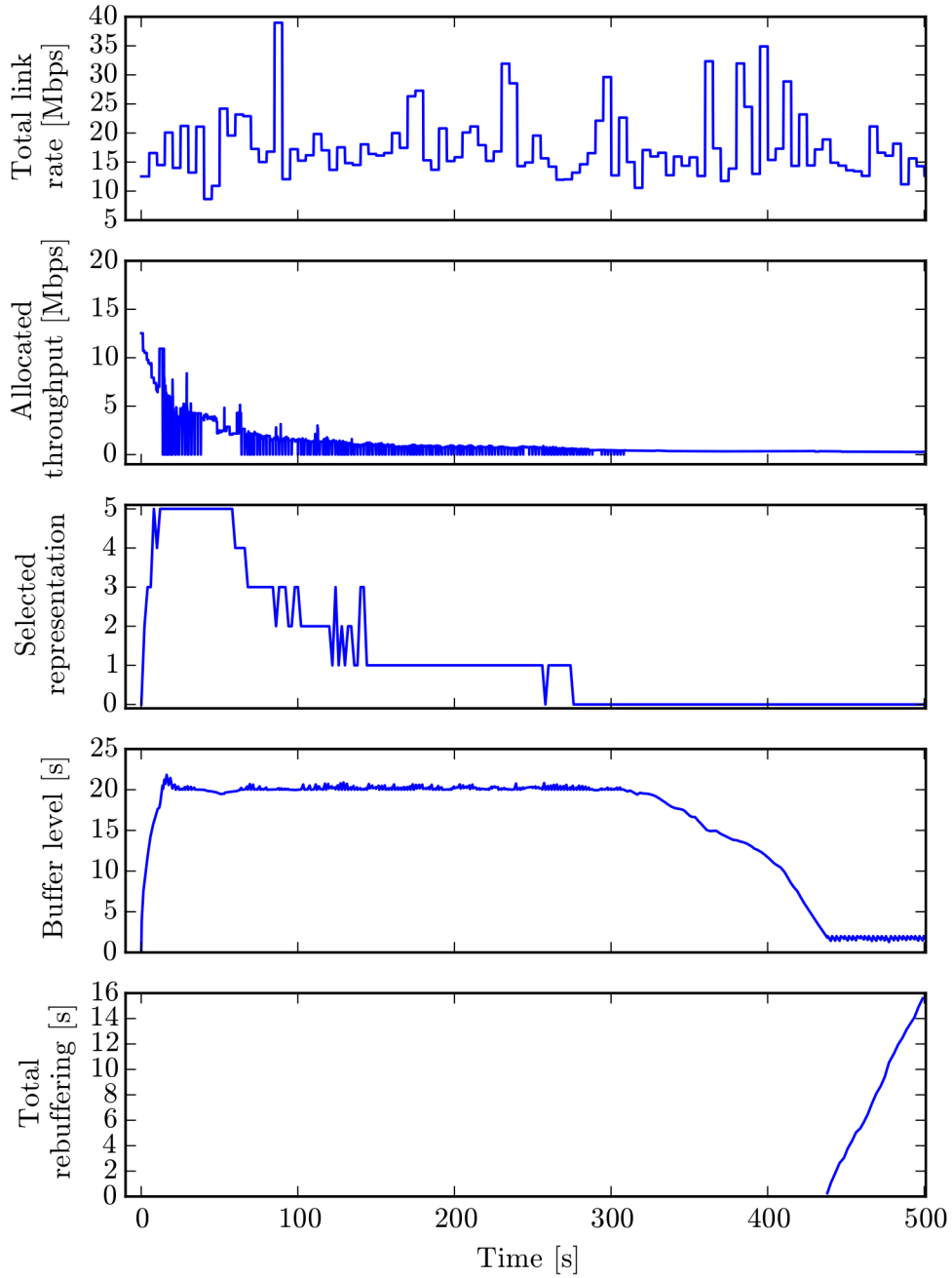

Figure 5.3: Dynamics of one user during an example run of experiment 3. See Section 5.5.4 for details.

that the conditional integration anti-windup technique successfully combats this effect, if $g_{i,\max}$ is sufficiently small. Also the minimum buffer level, depicted in the third and fourth rows in Figure 5.4, confirms the efficiency of the conditional integration technique in avoiding system instability. We also studied the mean buffer level. We observed that

it is within few seconds of the target buffer level, even for unstable configurations, and omit it here.

In the following, we only report results for $K_p = -0.05$, $K_i = -0.00001$, and $g_{i,\max} = 0.1$, which we confirmed to be a stable configuration, and omit the results for the other configurations.

5.5.4.2 Rebuffering

One of the main factors influencing QoE for video streaming is the amount of time a client spends rebuffering video data while the playback is halted. This happens when the playback buffer has been drained and the next segment does not arrive before its playback deadline.

Figure 5.5 shows mean and maximum total rebuffering per user, where the mean and the maximum are first taken over all users of a simulation run, and then over all runs performed for a setting. As in the previous section, we remove the initial arrival phase and the subsequent 100 seconds from each trace, for experiments 1, 1*, and 2 (experiment 3 does not contain an initial arrival phase).

As expected, we observe that the number of users the network can accomodate without rebuffering is higher with a uniform distribution of users. Moreover, we observe that the baseline approach results in significantly higher rebuffering values.

5.5.4.3 Prebuffering (Start-Up Delay)

Another critical factor influencing QoE is the prebuffering duration, or the start-up delay. Especially in a mobile context, when users tend to watch shorter videos, long start-up delays can be very annoying.

Figure 5.6 shows the mean and maximum prebuffering delays for experiments 2 and 3, for the uniform and the clustered user distributions. For experiment 2, only users who arrived after the initial phase are considered. For experiment 3, to facilitate comparison, the x-axis shows the number of users based on the user arrival rate of 2 users per second, instead of time.

With $K_p = -0.05$ and a target buffer level of 20 seconds, new users try to download the first segment at twice the media bit rate, that is, within one second. When there are few users in the system, the network can satisfy the corresponding throughput requests and even allocate some additional capacity to the individual users. When there are too many users in the system, the network cannot allocate the requested capacity for every user. When the load is moderate, many user receive exactly the requested capacity, resulting in one second prebuffering delay, as can be seen in Figure 5.6, left column.

5.5.4.4 Mean Video Quality

Figure 5.7 (top subfigure) illustrates the mean video quality, represented by the mean representation index, across all users. It displays the results for all four experiments, for uniform (left column) and clustered (right column) user distributions.

As in the previous sections, the arrival phase was removed for experiments 1, 1*, and 2. In addition, the first 30 seconds of each user's trace were removed from experiments

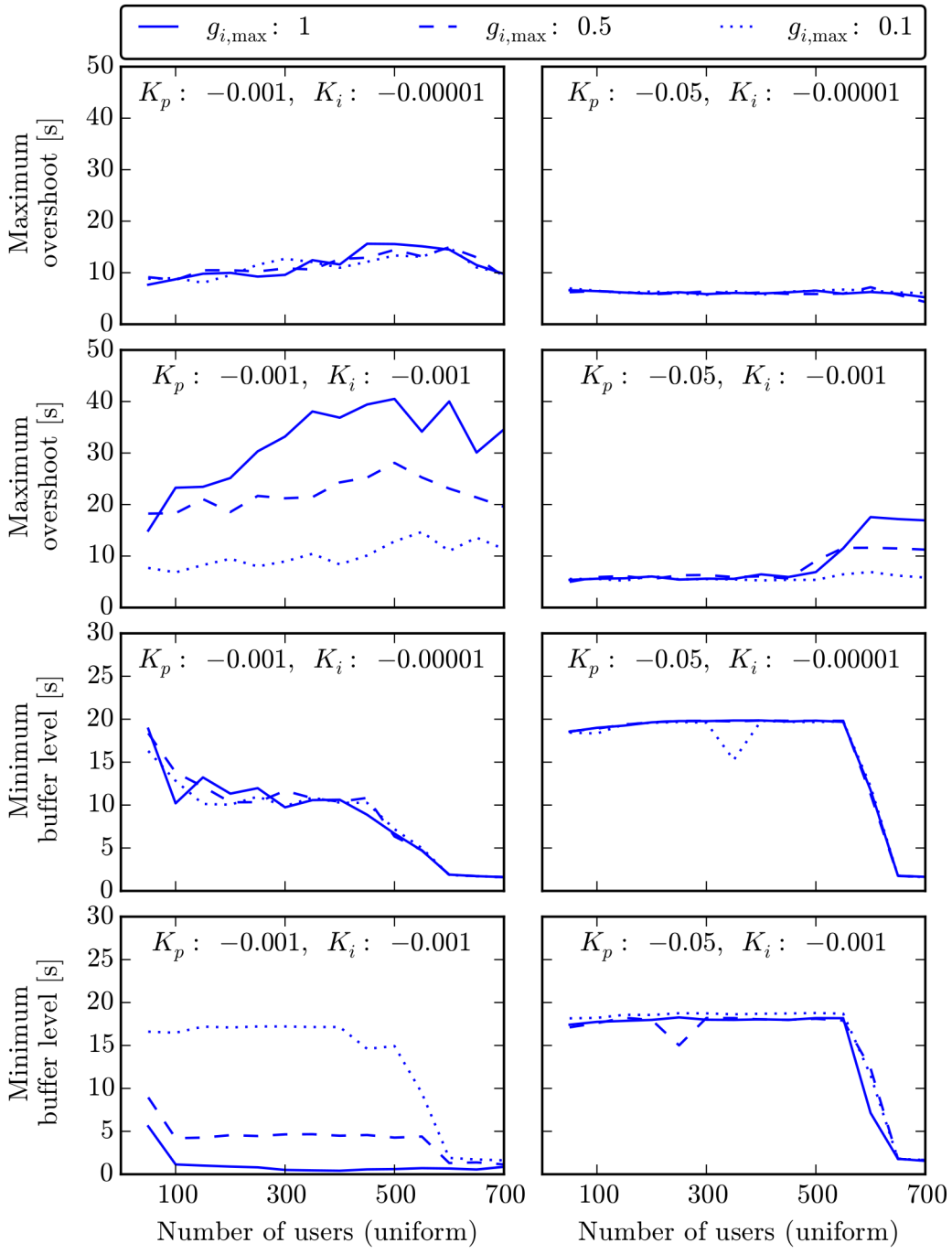

Figure 5.4: Stability analysis based on the buffer level statistics from experiment 1, see Section 5.5.4.1 for details. Top subfigures: maximum buffer level overshoot (difference between the buffer level and the target buffer level). Bottom subfigures: minimum buffer level.

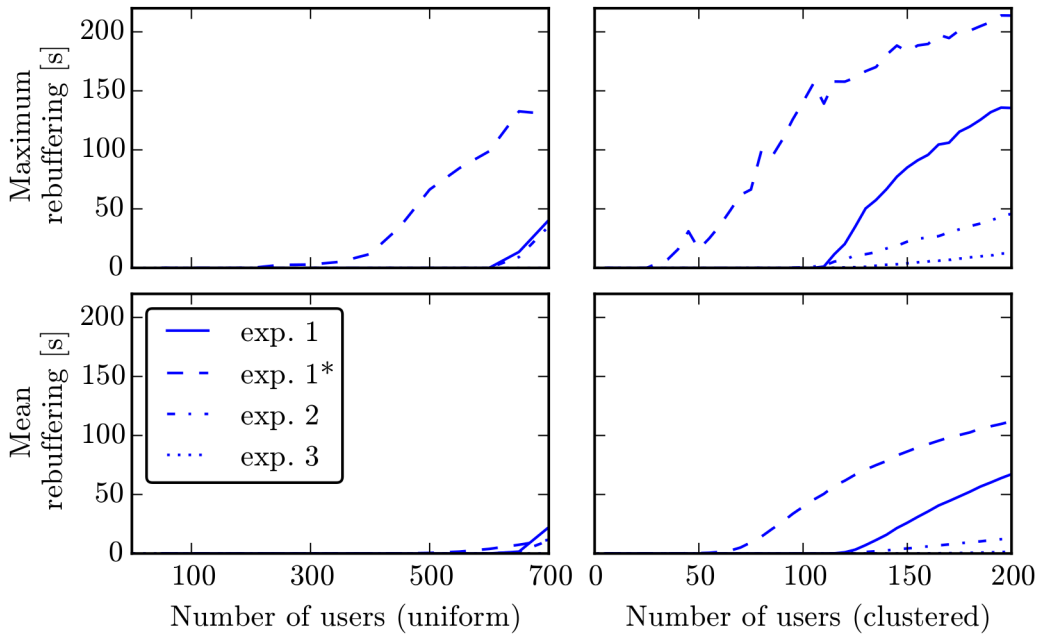

Figure 5.5: Mean and maximum total rebuffering per user. See Section 5.5.4.2 for details.

2 and 3, since a user always starts to stream at lowest available quality. Because of the latter, the results for experiments 2 and 3 do not include settings with less than 60 users, since in those settings no user remains in the network longer than 30 seconds.

We observe that experiment 1 and 1* exhibit comparable average video qualities for both user distributions. With clustered user distribution, the controller driven approach offers slightly better values, especially for small numbers of users.

5.5.4.5 Quality Transitions

Several studies have shown that severe quality fluctuations can dramatically degrade the QoE even if the average quality is high. Especially in cases when the network conditions may change very fast, such as in wireless networks, video clients have to implement adaptation strategies that restrain from immediately adapting the video quality to the throughput variations but that only react to long-term changes, where "long-term" is relative to the size of the playback buffer.

The middle subfigure in Figure 5.7 shows the number of quality transitions for all four experiments. As described in Section 5.5.1, we express the number of quality transitions as the fraction of segments that were played in a different quality than their predecessor, denoted by $\Omega \in [0, 1]$. With this definition, a value of 0.01 means that an adaptation takes place every 100 segment. With a segment duration of 2 seconds, this corresponds to one adaptation in 200 seconds. Note that since the set of available representations is discrete, the amount of quality transitions depends on the exact resource share of individual users. If the resource share is close to an available media bit rate, fluctuations are less likely than if it is in the middle between two representations, explaining the

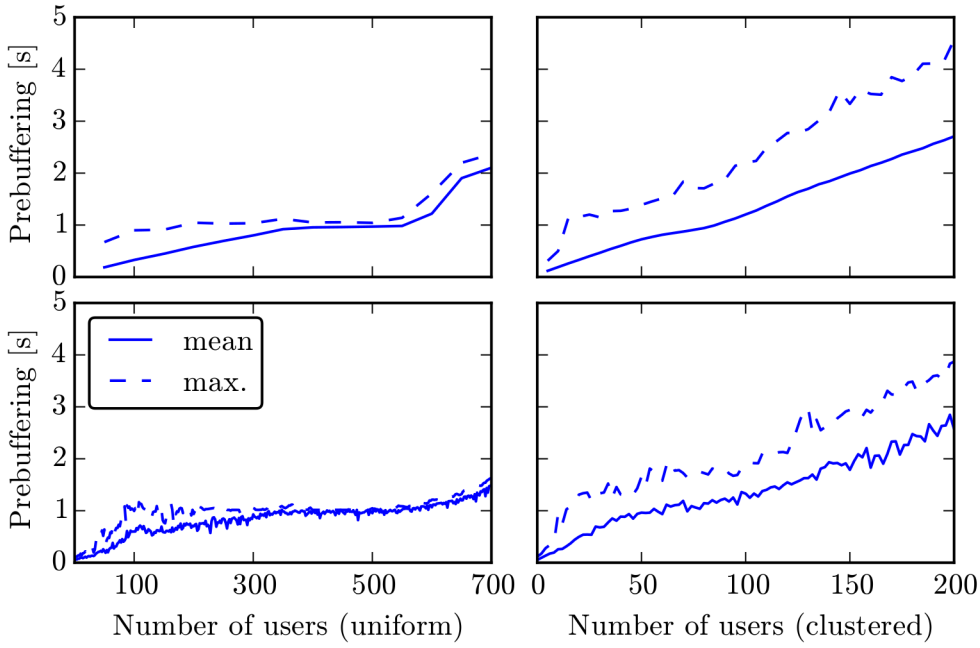

Figure 5.6: Mean and maximum prebuffering duration (start-up delay) for experiments 2 (top) and 3 (bottom). See Section 5.5.4.3 for details.

uneven shape of the curve.

We observe that the amount of adaptations is up to twice as high with the baseline approach as with the controller driven approach.

5.5.4.6 Media Bit Rate Fairness

Finally, the bottom subfigure in Figure 5.7 illustrates the (un)fairness by showing the interquartile range over all users in a network of their respective MMBR's. The higher the value, the lower the fairness. We observe that, except for the case of clustered users with less than 60 users in the network, the controller driven approach offers significantly better fairness than the baseline approach.

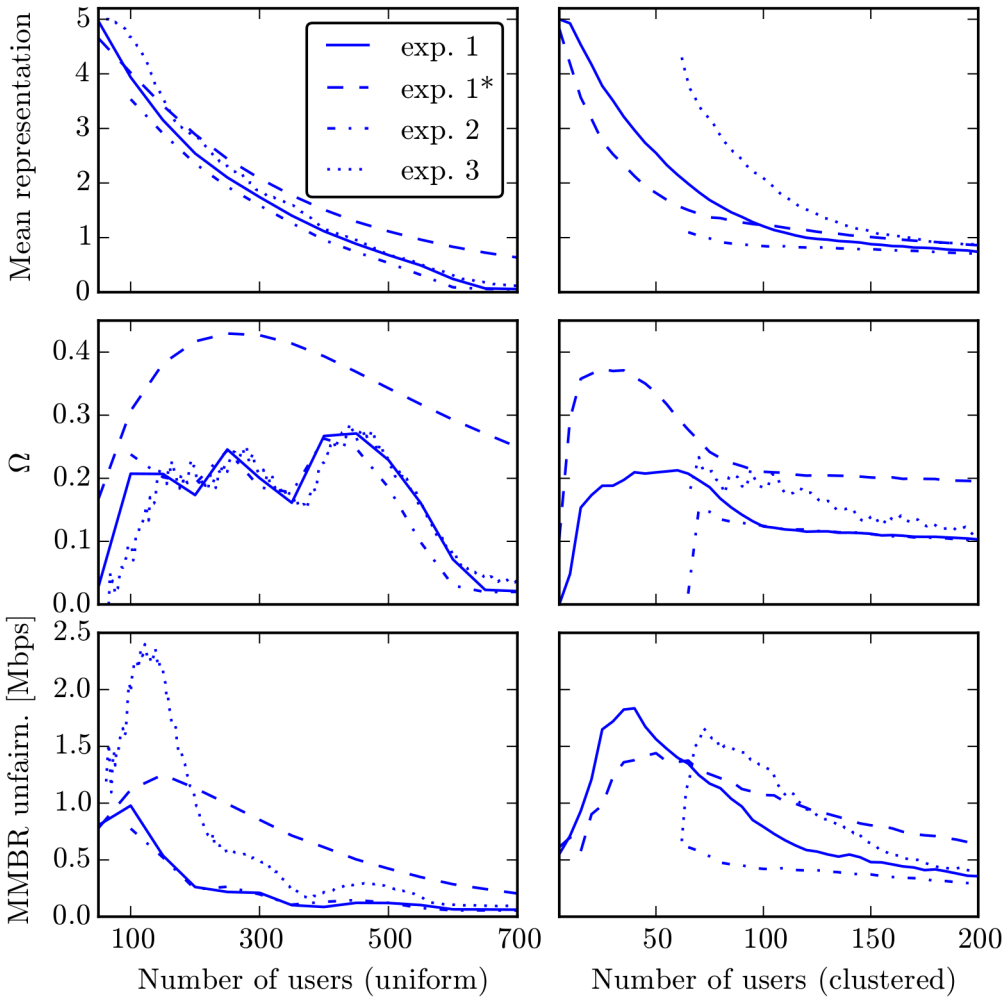

Figure 5.7: From top to bottom: mean video quality (representation index); number of quality transitions Ω; mean interquartile range of MMBR's (unfairness indicator). See Sections 5.5.4.4, 5.5.4.5, and 5.5.4.6 for details.

Chapter 6

Prediction-Based Low-Delay Live Streaming

Although the majority of the video content currently being streamed over the Internet is VoD, the amount of live streaming is growing rapidly [187]. In the case of live streaming, the task of dynamically adapting the media characteristics to varying network conditions in order to ensure a high QoE becomes particularly challenging due to the latency constraints. The challenge further increases if a client uses a wireless access network, where the throughput is subject to considerable fluctuations. Consequently, live streams often exhibit latencies of up to 20 to 30 seconds.

In our second contribution, we introduce an adaptation algorithm for HTTP-based live streaming called LOLYPOP (**Lo**w-**L**atenc**y** **P**redicti**o**n-Based Ada**p**tation) that is designed to operate with a transport latency of just a few seconds. To reach this goal, LOLYPOP leverages TCP throughput predictions on multiple time scales, from 1 to 10 seconds, along with an estimate of the relative prediction error distribution. In addition to satisfying the latency constraint, the algorithm heuristically maximizes the QoE by maximizing the average video quality as a function of the number of skipped segments and quality transitions. In order to select an efficient prediction method, we studied the performance of several time series prediction methods in IEEE 802.11 wireless access networks.

We evaluated LOLYPOP under a large set of experimental conditions, limiting the transport latency to 3 seconds, against a state-of-the-art adaptation algorithm from the literature, called FESTIVE. We observed that the average video quality is by up to a

factor of 3 higher than with FESTIVE. We also observed that LOLYPOP is able to reach a broader region in the QoE space, and thus it is better adjustable to the user profile or service provider requirements.

6.1 Introduction

In a live streaming system, the video content is recorded and published while being streamed, in contrast to being prerecorded and stored at the server as in the case of VoD. The difference between the time when the content is recorded and the time when it is rendered on the user's device is often termed live latency. In order to provide the "live" experience, the live latency is typically constrained by an upper bound. This severely limits the capability of the client to prefetch content to alleviate transport latency variations caused by varying network conditions, thus making the design of the system more challenging.

While current live streaming services can exhibit a latency of several tens of seconds, *low-delay* streaming refers to live streaming with a particularly low upper bound on the latency: a few seconds or less. Such a requirement is desirable for scenarios such as transmissions of sports events. Moreover, a low latency is absolutely necessary in the case of video conferencing and online gaming, where active participants have latency requirements on the order of hundred milliseconds [92], which is infeasible for HAS, while permanently or temporarily passive participants may be served with a delay of a few seconds.

HAS, however, was primarily developed to replace the progressive download of VoD content and therefore its usage for low-delay streaming has received little attention in the research community. Typical buffer sizes used in studies for evaluation and deployment of HAS-based clients are on the order of tens of seconds. The capability of the HAS approach to efficiently stream low-delay content, especially in wireless networks, is still an open question.

Consequently, in our second contribution we demonstrate that efficient HAS-based low-delay live streaming is possible by leveraging short-term TCP throughput predictions over multiple time scales, from 1 to 10 seconds, along with estimations of the relative prediction error distribution. We design a novel prediction-based algorithm called **Lo**w-**L**atenc**y** **P**redict**i**o**n**-Based Ada**p**tation (LOLYPOP) that supports quality-based adaptation with a transport latency on the order of a few seconds. The approach introduced in LOLYPOP jointly considers four QoE components: the live latency, the number of playback interruptions, the number of quality transitions, and the average video quality. Its goal is to maximize the average video quality as a function of the operating point defined by the other three components. The operating point is controlled by three input parameters: the target live latency, an upper bound on the number of quality transitions, and a parameter controlling the number of playback interruptions. Thus, LOLYPOP provides configurable QoE that can be adjusted to the nature of the video, the user context and preferences, or the service provider's business model.

At the core of LOLYPOP is an estimation of download success probabilities for individual segments. To obtain these estimations, LOLYPOP leverages predictions of throughput distributions, computed from a time series prediction and an error estima-

tion. We evaluate several time series prediction methods using TCP throughput traces collected in IEEE 802.11 WLANs, including public hotspots (indoor and outdoor), campus hotspots, and access points in residential environments. We observe, somewhat surprisingly, that taking the average over the previous T seconds as a prediction for the next T seconds provides the best prediction accuracy among the considered methods for all considered time scales. That is, taking into account the trend does not help to reduce the prediction error.

We implement a prototype of the algorithm and evaluate it against FESTIVE [98], a well-known adaptation algorithm from the literature. We limit the transport latency to 3 seconds using a segment duration of 2 seconds. We observe that LOLYPOP is able to reach a broad range of operating points and thus can be flexibly adapted to the user profile or service provider requirements. Furthermore, we observe that at the individual operating points, LOLYPOP provides an average video quality which is by up to a factor of 3 higher than the quality achieved by the baseline approach.

6.2 System Model and Notation

In the following, we extend the basic description of a HAS system from Section 2.1.4, and the basic notation from Chapter 4 (summarized in Table 4.1), to the case of a low-delay live streaming client. A summary of the extensions is provided in Table 6.1, which also includes notation introduced in the subsequent sections.

In a live streaming system, the video content is recorded and published while being streamed, in contrast to being prerecorded and stored at the server as in the case of VoD. The difference between the time when the content is recorded and the time when it is rendered on the user's device is often termed the live latency. In order to provide the "live" experience, it is typically constrained by an upper bound. This severely limits the capability of the client to prefetch content to alleviate transport latency variations caused by varying network conditions, thus making the design of the system more challenging. The live latency consists of several components: sever-side processing (cutting, encoding, etc.), publishing (making available for download, distributing among CDN nodes, etc.), transport latency (downloading the content), and client-side processing (demultiplexing, decoding, rendering).

Recall that in an HAS system, the video content is encoded in several representations varying w.r.t. their media characteristics such as the spatial resolution, frame rate, compression rate, etc. They are configured by the service provider during the planning phase [175]. Each representation is split into segments, typically containing several seconds of video data, such that switching the representation is feasible on each segment boundary. The client issues Hypertext Transfer Protocol (HTTP) requests to download the segments in chronological order, selecting the representation for each of them. After the segment is downloaded, it is stored in the playback buffer until its playback deadline is reached. With live streaming, a segment becomes available for download during the course of the streaming session. If the download is not completed before the playback deadline, the playback is skipped. Since different representations typically have different media bit rates, the client is able to satisfy the latency constraint by dynamically selecting an appropriate representation for each segment. Note that the segment duration

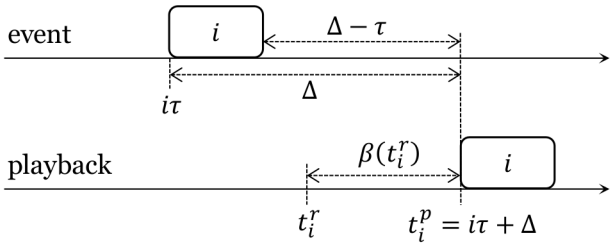

Figure 6.1: Illustration of the time-related notation for low-delay live streaming

affects the client's responsiveness to throughput changes and thus facilitates achieving low latencies. At the same time, however, small segments increase the overhead due to the higher number of HTTP requests as well as reduce the video compression efficiency due to the decreased Group of Pictures (GOP) size. Typical segment durations lie between 2 and 15 seconds.

Since HTTP offers no means to cancel an ongoing request, the only way to prevent wasting bandwidth by downloading the remaining bytes of a segment whose playback has to be skipped is to shutdown the TCP connection. Since opening a new TCP connection is associated with communication overhead, we assume that the client maintains multiple TCP connections, using them in a Round Robin manner in order to keep their internal state such as congestion window size and Round-Trip Time (RTT) estimation up-to-date. Whenever a TCP connection is closed, other connections are used, while the closed connection is replaced by a new one.

One of the main goals of the adaptation logic is to maximize the QoE. In the following, we define QoE as the quadruplet (live latency, number of skipped segments, number of quality transitions, average video quality). We use the term video quality to refer to the video distortion, which is typically a concave function of the video bit rate [184]. As stated previously, it is necessary to consider these factors jointly since optimizing any one parameter individually leads to poor QoE.

Our approach is to heuristically maximize the average video quality as a function of the triplet: live latency, number of skipped segments and number of quality transitions, which we define as an operating point. Since the duration of the streaming session is not known in advance (the user might quit the session prematurely), the number of skipped segments and the number of quality transitions are expressed in relative terms: fraction of skipped segments and fraction of segments that result in a transition. The operating point may be defined by the user, the operating system, the client software, or the content provider. It might depend on various factors, such as the nature of the video, the user context, the provider's business model, etc.

We assume that segment i contains video material covering the time period $[i\tau, (i+1)\tau]$ and becomes available for download at time $(i+1)\tau$. Consequently, the time t_i^r when the request to download segment i is sent by the user arises from the maximum of two values: the time when the client finished downloading segment $i-1$, and the time when segment i becomes available at the server. We denote the target live latency by Δ. Consequently, the playback deadline of segment i is $t_i^p = i\tau + \Delta$. The value of Δ bounds

Δ	Target live latency
$\Sigma^* \in [0,1]$	LOLYPOP input parameter controlling the number of skipped segments Σ
$\Omega^* \in [0,1]$	LOLYPOP input parameter controlling the number of quality transitions Ω
i_0	Selected tune-in segment, defined in (6.1)
P_{ij}^p	Download success probability for segment i from repr. j
k_p	Number of past throughput values used for prediction
$\hat{\rho}(t_1, t_2)$	Predicted throughput for the time interval $[t_1, t_2]$
$\epsilon(t_1, t_2)$	Relative prediction error for $[t_1, t_2]$ (unsigned), defined in (6.2)
$\tilde{\epsilon}(t_1, t_2)$	Relative prediction error for $[t_1, t_2]$ (signed), defined in (6.3)
$\hat{\rho}_i$	Predicted segment throughput for segment i
$T_{\max}$	Maximum prediction horizon in seconds

Table 6.1: Notation extensions for LOLYPOP

the maximum transport latency, which is given by $\Delta - \tau$ if other latency components are neglected. Note that the maximum transport latency for individual segments can be smaller since it depends on the playback buffer level at the time of the request. The maximum transport latency for segment i is thus given by $0 < \beta\left(t_i^r\right) \leq \Delta - \tau$, where $\beta(t)$ is the playback buffer level representing the duration of the stored video content in seconds (defined in (4.1)). See Figure 6.1 for an illustration.

6.3 LOLYPOP — Adaptation Algorithm for Low-Delay Live Streaming

In this section, we present our design of LOLYPOP, a novel prediction-based adaptation algorithm for low-delay live streaming over HTTP.

6.3.1 Algorithm Description

As described in Section 6.2, the goal of LOLYPOP is to maximize the average video quality as a function of the operating point, defined by the triplet (Δ, Σ, Ω). The input parameters controlling the reached operating point are: (i) the target latency Δ, (ii) $\Sigma^* \in [0,1]$, which controls the fraction of skipped segments, and (iii) $\Omega^* \in [0,1]$, which is an upper bound on the (relative) number of quality transitions. The output of the algorithm is the representation for the next segment to be downloaded. The approach leverages throughput predictions and prediction error estimations to compute the probability P_{ij}^p that the download of segment i in quality j will be completed before its playback deadline.

Computation of P^p_{ij} is described in detail in Section 6.5.5. For now, we assume that P^p_{ij} is given.

Let us consider the decision about downloading the segment i. First, LOLYPOP identifies the highest representation j' such that the probability for missing the playback deadline is bounded by Σ^*, i.e. $1 - P^p_{ij'} \leq \Sigma^*$. If no representation satisfies this condition or the download success probabilities cannot be computed (e.g., because the streaming session has just started or after a period of zero throughput), j' is set to 0.

In the second step, LOLYPOP computes $\Omega(t)$, the fraction of segments that were played in a different quality than their predecessor. If $\Omega(t) > \Omega^*$, and $j' > j^{\leftarrow}$, where $j^{\leftarrow}$ is the representation of the last successfully downloaded segment $i^{\leftarrow} < i$, representation $j^{\leftarrow}$ is selected in order to prevent $\Omega(t)$ from further exceeding the upper bound Ω^*. The pseudocode for the described algorithm is presented in Algorithm 1.

Algorithm 1: LOLYPOP

Input: τ, $\mathcal{R}$ ▷ *Video parameters*
Input: t^r_i, t^p_i, Σ^*, Ω^*, $T_{\max}$ ▷ *Invocation time, playback deadline, config. parameters*
Input: $(P^p_{ij}, j \in \{0, \ldots, |\mathcal{R}| - 1\})$ ▷ *Estimated download success probabilities, or -1*
Input: $j^{\leftarrow} \in \{0, \ldots, |\mathcal{R}| - 1\}$ ▷ *Repr. of the last successf. downloaded segment*
Output: j^* ▷ *Selected representation*
Require: $(i+1)\tau \leq t^r_i < t^p_i \leq t^r_i + T_{\max}$ ▷ *Segment i available, playback deadline not passed and within prediction horizon*

1 **if** $P^p_{ij} = -1,\ \forall j \in \{0, \ldots, |\mathcal{R}| - 1\}$ **then** ▷ *No estimation available*
2 $\quad j^* = 0$ ▷ *Select lowest representation*
3 **else** ▷ *An estimation of download success probabilities is available*
4 $\quad j' = \max\left(\{0\} \cup \left\{j \in \{0, \ldots, |\mathcal{R}| - 1\} \mid 1 - P^p_{ij} \leq \Sigma^*\right\}\right)$ ▷ *Max. representation satisfying Σ^*, or 0*
5 $\quad$ **if** $\Omega\left(t^r_i\right) \leq \Omega^*$ **then** ▷ *Transition to a higher representation is possible*
6 $\quad\quad j^* = j'$ ▷ *Select the computed representation*
7 $\quad$ **else** ▷ *Transition to a higher representation is not possible*
8 $\quad\quad j^* = \min\left(j', j^{\leftarrow}\right)$ ▷ *Select the computed representation, if below the previous*

The intuition for letting LOLYPOP switch to a lower representation, even if the upper bound on the quality transitions is exceeded, is that preventing a quality decrease can significantly increase the number of skipped segments. According to a large-scale study of user engagement (time before the user quits a streaming session), it is always better to drop video quality than to let the streaming stall [43].

6.3.2 Tuning into the Stream

Starting a new streaming session at time t_0, the client decides which segment i_0 to download first and in which representation. After the download, the clients delays the playback until the playback deadline $t^p_{i_0} = i_0\tau + \Delta$, in order to fully benefit from the configured target latency Δ. Starting with the newest available segment maximizes the download success probability (there is more time until the playback deadline), but increases the initial delay (the client has to wait longer after the download). In contrast, taking an older segment whose playback deadline is sufficiently far into the future minimizes the initial delay given that the download can be completed in time.

LOLYPOP adopts the latter approach, selecting the oldest segment whose playback deadline is at least τ seconds into the future:

$$i_0 = \min \{i \geq 0 \mid (i+1)\tau \leq t_0 \leq t_i^p - \tau\} \ , \tag{6.1}$$

and downloading it in the lowest quality to minimize the risk of skipping the first segment and thus increasing the initial delay. The expectation is that the available network resources support the download of a segment in its lowest quality in less time than the segment duration, while, due to the small segment duration, the low quality of the first segment should have negligible impact on the QoE of the whole session. If the first segment is downloaded in time, the start-up delay $t_{i_0}^p - t_{i_0}$ lies in the interval $[\tau, \Delta - \tau]$, which can be seen by using the definition of i_0, and that of the playback deadline $t_{i_0}^p = i_0\tau + \Delta$.

LOLYPOP applies the same decision process when the client skips a segment and has to select a segment to proceed with.

6.4 TCP Throughput Traces

As previously stated, LOLYPOP leverages estimations of probabilities P_{ij}^p that segment i can be downloaded in representation j before its playback deadline. In order to develop an efficient estimator, it requires a data set, that is, a collection of TCP throughput traces from IEEE 802.11 WLANs, to evaluate the accuracy and error distributions of different time-series prediction methods. In addition, such a data set is required to evaluate the proposed adaptation algorithm under different network conditions. Due to the targeted transport latency of a few seconds, the required data set must contain throughput averages computed over relatively small time intervals: 1 second or less. To the best of our knowledge, there exists no such publicly available data set. Therefore, we collected a representative set of TCP throughput traces in IEEE 802.11 networks in different environments at various locations throughout Berlin, Germany and Irbid, Jordan. Our selected locations include public hotspots (indoor and outdoor), campus hotspots, and access points in residential environments. The traces were collected using laptops running Ubuntu 13.04 and Ubuntu 14.04 operating systems with default Media Access Control (MAC) and TCP configurations.

We collected 127 traces of continuous downstream TCP flows, lasting between 600 and 3600 seconds each. In order to focus on the more challenging scenarios, we removed traces with a Coefficient of Variation (CV) of less than 0.1 resulting in 92 traces with a total length of 45 hours. As a sender, we used either a server located at the TU Berlin campus (running Ubuntu 12.04) or an Amazon EC2 micro instance located in Ireland (running Ubuntu 14.04). All traces are available upon request.

Each of the collected traces contains several types of information. First, it contains the first 96 bytes of each incoming TCP packet belonging to the monitored TCP flow. Second, it contains the first 512 bytes of each received IEEE 802.11 frame independent of its destination address, including radiotap headers[1] that contain internal MAC information, such as the retransmission flag, the Modulation and Coding Scheme (MCS),

[1] http://www.radiotap.org

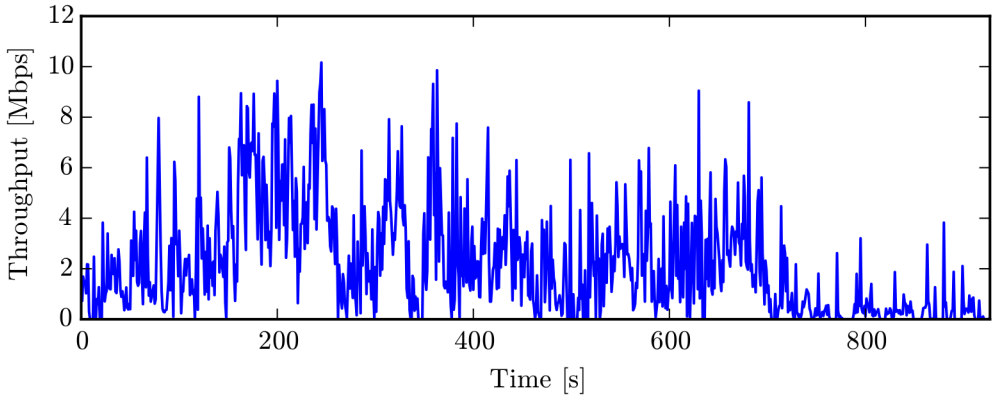

Figure 6.2: A complete example trace with a CV of 0.879, recorded at a busy outdoor hotspot of a major German telecommunications operator.

and the Signal Strength Indicator (SSI). Except for the radiotap headers, the captured frames are encrypted. Finally, the traces contain periodically recorded values of internal TCP variables, obtained using the tcp_info data structure via the socket interface. From the traces, we computed time series containing various statistics from overlapping time intervals of 1 s to 10 s duration shifted with a step size of 1 s. In addition to throughput statistics, we computed statistics of cross-layer information such as for TCP: delay jitter statistics and the statistics of outstanding bytes, and, for MAC: the number of own frames received, the number of other frames received, MCS and SSI statistics, and the statistics of retransmissions. Figure 6.2 shows the throughput, averaged over one second intervals, of one complete trace with a CV of 0.879 recorded at a busy outdoor hotspot of a major German telecommunications operator.

Figure 6.3 provides an illustration of throughput variability and temporal correlation that are among the main factors affecting predictability. The figure shows boxplots for selected sampling intervals of the mean throughput, Coefficient of Variation (CV), auto-correlation at lag 1, and auto-correlation after differencing at lag 1. The CV is defined as the standard deviation divided by the mean. Auto-correlation at lag 1 shows how probable it is that a large throughput value is followed by another large value (auto-correlation close to 1) or a small value (auto-correlation close to -1). A value close to 0 indicates no temporal correlation between subsequent values. Auto-correlation after differencing quantifies the correlation of throughput changes. The computed statistics confirm that our traces cover a broad range of network conditions. 50% of the traces have a mean throughput between 4 and 11 Mbps, while for 90% of traces the mean lies between approximately 1 and 13 Mbps. The range of throughput fluctuations, represented by the CV, varies approximately from 0.1 to 1.3. An interesting observation is that 75% of traces show auto-correlation values of over 0.6 at a sampling interval of 2 seconds, while 95% still have auto-correlation values over 0.3, indicating significant temporal correlation between subsequent measurements. At the same time, the time series of throughput changes exhibits a strong negative auto-correlation, indicating that a throughput increase is likely to be followed by a throughput decrease.

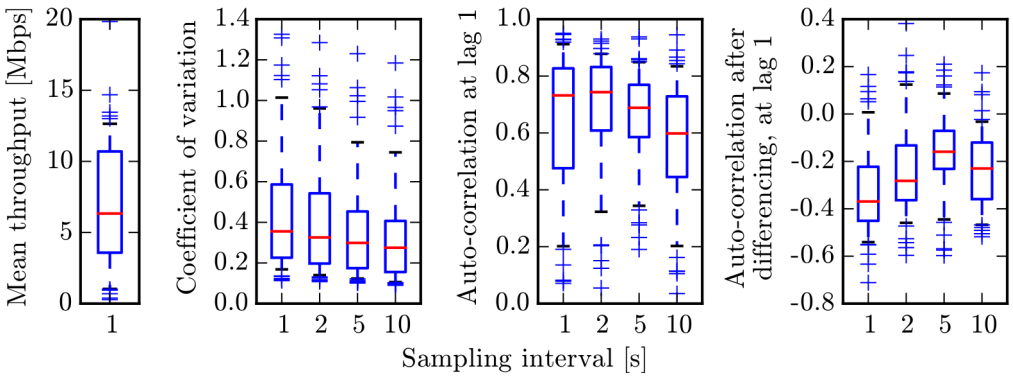

Figure 6.3: Trace statistics: mean throughput (equal for all sampling intervals), Coefficient of Variation (CV), auto-correlation at lag 1, and auto-correlation, after differencing, at lag 1. Horizontal line: median, box: quartiles, whiskers: 0.5 and 0.95 quantiles, flier points: outliers.

6.5 Short-Term TCP Throughput Prediction

In this section, we present our approach to estimating download success probabilities required by LOLYPOP. It is based on predicting TCP throughput and estimating the relative prediction error distribution.

6.5.1 Methodology

Our goal is to estimate the probabilities P_{ij}^p that the download of s_{ij} bytes, requested at time t_i^r, will be completed by the time t_i^p. We achieve this by using a time series prediction complemented by estimating the relative prediction error distribution. A time series prediction method uses several past values to compute one or several future values. Thus, from $\rho\left(t-(k+1)T, t-kT\right)$, $k \in \{0, \ldots, k_p-1\}$, it computes predictions $\hat{\rho}\left(t+kT, t+(k+1)T\right)$, $k \in \{0, \ldots, k_f-1\}$, where T is the averaging interval.

As described in Section 6.2, the upper bound on the download duration $t_i^p - t_i^r$ for segment i takes values from the range $(0, \Delta^p - \tau]$, depending on the completion time of the preceding download. We, therefore, have the following two options to compute predictions. We can fix T and whenever $t_i^p - t_i^r > T$, compute a prediction for multiple steps into the future or, we compute predictions using multiple values for T and then use the smallest one such that $T \geq t_i^p - t_i^r$. In the course of the study, we observed that the latter approach performs significantly better. Consequently, we focus on predictions on multiple time scales, always for one step into the future. We focus on time scales from 1 to 10 seconds because of their relevance to low-delay streaming.

Given a prediction $\hat{\rho}\left(t_1, t_2\right)$ and the corresponding measured throughput $\rho\left(t_1, t_2\right)$, we compute the relative prediction error as

$$\epsilon\left(t_1, t_2\right) = \frac{\left|\max\left(\hat{\rho}\left(t_1, t_2\right), \rho_{\min}\right) - \max\left(\rho\left(t_1, t_2\right), \rho_{\min}\right)\right|}{\max\left(\rho\left(t_1, t_2\right), \rho_{\min}\right)}. \tag{6.2}$$

Here, the maximum operator prevents a distortion of results whenever $\rho \approx 0$ or $\hat{\rho} \leq 0$. In

the following, we set $\rho_{\min} = 10$ kbps. We separately evaluate the overestimation and the underestimation errors, due to their different error ranges ($(0, \infty)$ and $(0, 1]$ respectively) and due to their different impacts on the adaptation. An overestimation increases the risk of skipping a segment, which has the strongest impact on QoE. An underestimation decreases the risk of interruptions but reduces the video quality.

6.5.2 Prediction Methods

We evaluated a number of time series prediction methods, including Simple Moving Average (SMA), linear extrapolation, Cubic Smoothing Splines (CSS), several flavors of exponential and double exponential smoothing, Autoregressive Integrated Moving Average (ARIMA), and several machine learning based methods [27]. Due to their simplicity and superior performance, our throughput prediction results presented in the subsequent sections will focus on three simple methods: SMA, linear extrapolation, and double exponential smoothing (Holt-Winters). In this section, we provide a brief description for a subset of the used methods.

We abbreviate the methods by ⟨type⟩:⟨k_p⟩:⟨parameters⟩, where ⟨type⟩ is the name of the method, k_p is the number of past throughput values used as input, and ⟨parameters⟩ include further optional configuration parameters. For example, SMA:k_p:ar denotes SMA with arithmetic mean, and HW:k_p:mse denotes Holt-Winters with Mean Squared Error (MSE) used for parameter tuning.

Simple Moving Average

SMA is one of the simplest prediction methods. The predicted value is the average over a number of past measurements. The configuration parameters are the number of past measurements and the type of the used mean value: arithmetic, geometric, or harmonic. In the following, we abbreviate this method with SMA:⟨k_p⟩:⟨mean type⟩, where $k_p \geq 1$ is the number of past measurements, and 'mean type' is one of $\{\text{ar}, \text{gm}, \text{hm}\}$. For example, SMA:2:ar means that the predicted value is the arithmetic mean from two past measurements. In particular, we denote the naïve approach of using the most recent measurement as the predicted value with SMA:1:ar.

Simple Exponential Smoothing

Similar to SMA, Simple Exponential Smoothing (SES) computes the predicted value by averaging over past measurements. However, it assumes that the most recent measurements have a higher significance for the prediction, and assigns older measurements exponentially decreasing weights. For given past measurements $x_0, \ldots, x_{k_p-1}$, the predicted value is computed as $\hat{x}_{k_p} = a_{k_p-1}$, where a_k is recursively computed as

$$a_k = \alpha x_k + (1 - \alpha) a_{k-1} \,,$$

and $a_0 = x_0$. Besides the number of past measurements k_p, it has a configuration parameter $\alpha \in [0, 1]$. We tune α for each prediction by minimizing the MSE within past

measurements, given by

$$\frac{1}{k_p - 1} \sum_{k=1}^{k_p-1} (x_k - a_{k-1})^2 .$$

We abbreviate SES with SES:$\langle k_p \rangle$:mse, where $k_p \geq 2$ is the number of past measurements, and "mse" indicates the approach used to tune α.

Linear Extrapolation

Linear extrapolation is another straightforward prediction method that differs from SMA in that it takes into account the linear trend from the past measurements. More specifically, linear extrapolation fits a linear curve into the set of given past measurements, minimizing the MSE, and computes the prediction from extrapolating the curve to the prediction horizon. It thus requires at least two past measurements to compute a prediction. We abbreviate linear extrapolation with LinExt:$\langle k_p \rangle$, where $k_p \geq 2$ is the number of past measurements.

Double Exponential Smoothing (Holt-Winters)

Similar to linear extrapolation, Double Exponential Smoothing (DES) tries to account for the trend in the data. In the following, we use a variant of the method, usually referred to as Holt-Winters DES. With Holt-Winters, for the given past measurements $x_0, \ldots, x_{k_p-1}$, the prediction is computed as $\hat{x}_{k_p} = a_{k_p-1} + b_{k_p-1}$, where a_k, b_k are computed by the following recursive procedure.

$$\begin{aligned} a_k &= \alpha x_k + (1 - \alpha)(a_{k-1} + b_{k-1}) , \text{ for } k > 1 \\ b_k &= \beta(a_k - a_{k-1}) + (1 - \beta) b_{k-1} , \text{ for } k > 1 , \end{aligned}$$

with $a_1 = x_1$, and $b_1 = x_1 - x_0$.

The Holt-Winters method has two configuration parameters α and β that strongly influence the prediction quality and thus have to be carefully tuned. In our work, we tune them for each prediction by minimizing the MSE within the past measurements, which is given by

$$\frac{1}{k_p - 2} \sum_{k=2}^{k_p-1} (x_k - (a_{k-1} + b_{k-1}))^2 .$$

Thus, this method requires at least three past values to compute a prediction. As abbreviation, we use HW:$\langle k_p \rangle$:mse, where $k_p \geq 3$ is the number of the last values, and *mse* indicates the approach used to tune α and β.

Cubic Smoothing Splines

The CSS model provides both smooth historical trend and a linear prediction function. The method uses a likelihood approach to estimate the smoothing parameter. It is based on finding piecewise cubic polynomials that are joined at the equally spaced time series points [87].

Locally Weighted Scatterplot Smoothing

Locally Weighted Scatterplot Smoothing (LOESS) is a non-parametric regression method that tries to build a model describing the deterministic part of the variation in the data by incrementally fitting localized subsets of the data using simple regression models.

Autoregressive Model

Autoregressive model predicts the variable of interest by using a linear combination of past measurements, plus white noise. For past measurements $x_0, \ldots, x_{k_p-1}$, the prediction of an autoregressive model can be written as:

$$\hat{x}_{k_p} = c + \sum_{k=0}^{k_p-1} \alpha_k x_k + \omega_{k_p-1} \,,$$

where c is a constant and ω_{k_p-1} denotes white noise. The model parameters are selected by optimizing the Akaike Information Criterion (AIC).

Autoregressive Integrated Moving Average

ARIMA is a combination of autoregressive and moving average models, with the ability to use a differenced or integrated representation of the time series. In our study, we were using a statistical method proposed in [86] that uses a combination of unit root test, minimization of the AIC and Maximum Likelihood Estimation (MLE) to reach an optimized ARIMA model.

6.5.3 Evaluation of the Prediction Accuracy

We evaluate the prediction methods in two steps. First, we compare the relative prediction errors over the joint data set from all of the traces to identify the method that performs best over a broad range of network environments. In the second step, we evaluate how the prediction accuracy varies over individual traces.

To compare the prediction accuracy over the complete data set, we computed the 0.2-quantiles, 0.5-quantiles, and 0.9-quantiles of the prediction error. For example, a 0.2-quantile of 0.3 means that in 20% of all collected data points, the relative prediction error is below 0.3. Another example: a 0.9-quantile of 0.8 means that less than 10% of data points have an error over 0.8. The results for the sampling interval of 5 seconds are shown in Figure 6.4. We observe that SMA:1:ar has the best performance except for the 0.9-quantile of the underestimation error, where SMA:10:ar is the best performing method. For 50% of the data points, SMA:1:ar results in an overestimation error that does not exceed 10%, while only less than 10% of data points have an overestimation error of 80% and larger. This is somewhat surprising since SMA:1:ar is the most naïve method that uses only the most recent measurement as prediction. It also has a much lower computational complexity than methods such as Holt-Winters due to the optimizations involved in tuning the configuration parameters of the latter for every new prediction. It seems that taking into account the trend in the past measurements does not improve the prediction quality. This is consistent with the observation that in many traces

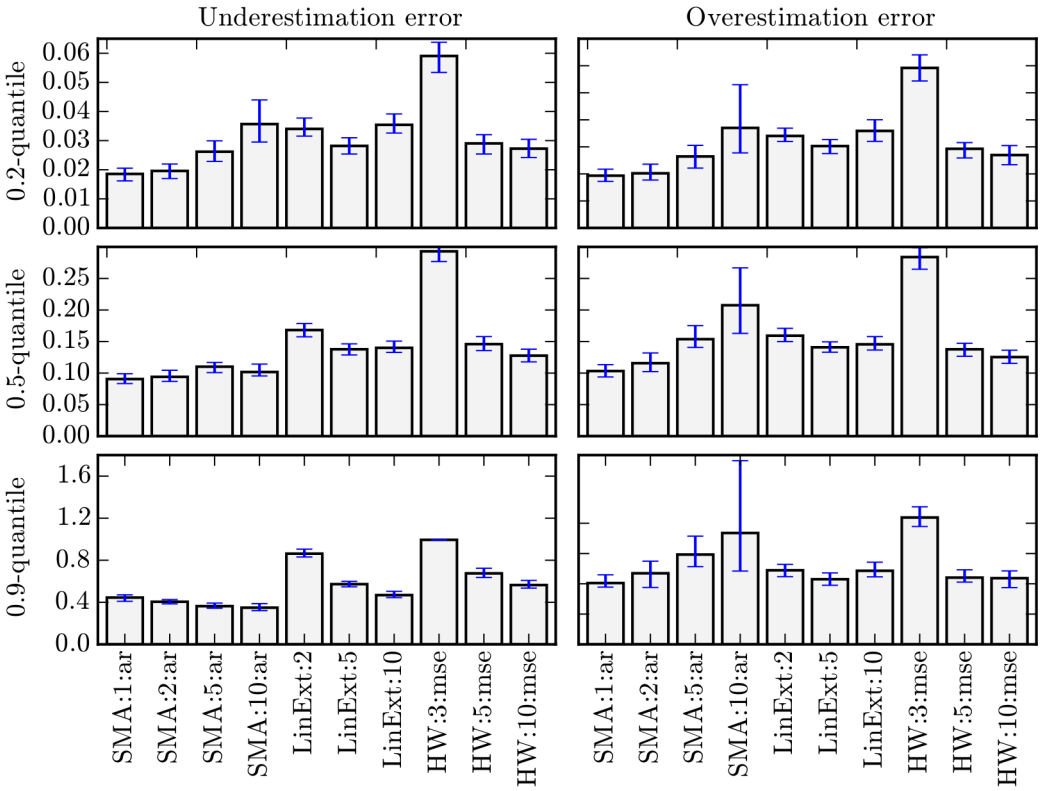

Figure 6.4: Relative prediction error quantiles for the complete data set, on the time scale of 5 seconds. The error bars show confidence intervals for the confidence level of 0.95 (a varying subsampling factor has been used for decorrelation [119]). See Section 6.5.3 for details.

the differences in subsequent throughput measurements show a negative correlation, as depicted in Figure 6.3. Also, methods using a small number of history points implicitly detect level shifts and do not propagate outliers. These two issues were reported to be known challenges in TCP throughput prediction [70].

In the second step, we evaluate how the prediction accuracy varies over the individual traces. For each trace and method, we compute the fractions of predictions with a relative error less than 0.2, 0.5, and 1.0. The Empirical Cumulative Distribution Functions (ECDF's) of these fractions over individual traces for the sampling interval of 5 seconds, is shown in Figure 6.5. The first/second/third column shows for each trace the fraction of measurements with a relative error below 0.2/0.3/0.5 respectively. For example, the point $(0.8, 0.3)$ on the solid line in the middle column, bottom row, represents a trace, where 80% of the overestimations have a relative error of 0.5 or less. Points below (y-value less than 0.3) correspond to traces that have worse performance, while points above (y-value over 0.3) correspond to traces with better performance. The fact that the graph passes through point $(0.8, 0.3)$ means that in 70% of traces (sampled at 5 s), less than 20% of the overestimations have a relative error of 0.5 or more.

From Figure 6.5, we observe that the prediction strongly varies across traces. There

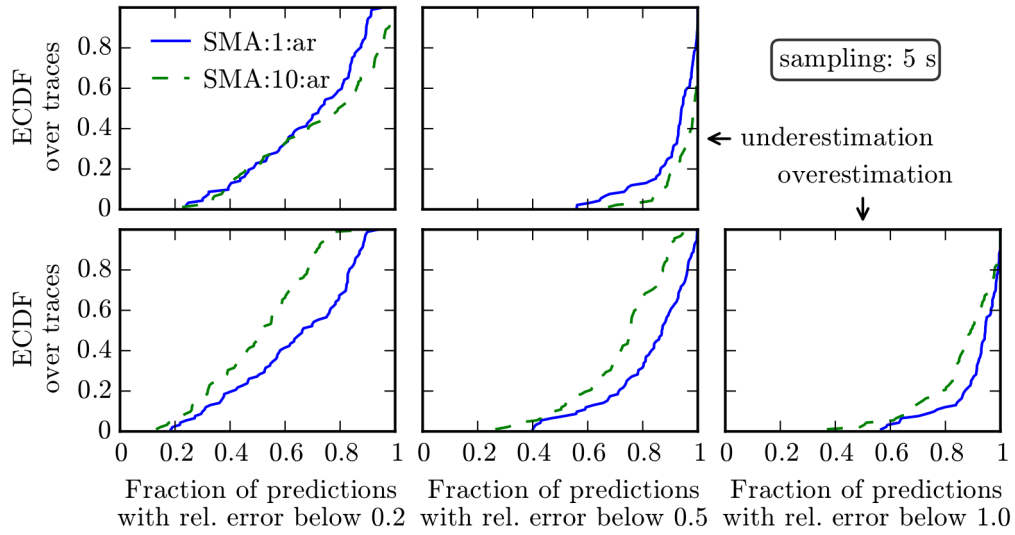

Figure 6.5: Performance of SMA:1:ar and SMA:10:ar for individual traces. See Section 6.5.3 for details.

are traces, where 90% of overestimations have an error less than 20%, while 100% of predictions have an error less than 50%. A video client might account for a relative error of this magnitude by using a fixed safety margin, that is, by always selecting a media bit rate which is 20% smaller than the predicted throughput. There are, however, traces where almost 60% of the overestimations have a relative error of greater than 50%, while more than 40% of the overestimations still have an error greater than 100%. Setting a high fixed safety margin to account for such "bad" traces would result in significant underutilization of network resources, lower media bit rate and thus lower QoE in the "well-behaving" traces. On the other hand, selecting a low fixed safety margin would increase the total number of skipped segments in the "bad" traces. Consequently, we have to complement a time series prediction with an estimation of the prediction error distribution.

Finally, in all of the studied traces, we observed that the probability for occurrences of underestimations and overestimations are well balanced on all time scales. Both occur in approximately $50\% \pm 5\%$ of cases. At the same time, they exhibit a significant temporal correlation. In particular, the probability that an underestimation is followed by an overestimation and vice versa is significantly over 50% for most traces for all time scales, exceeding 80% or even 90% in some cases. This observation is directly related to the distinct negative correlation of the throughput process after differencing, as depicted in Figure 6.3. The distribution of per-trace values is depicted in Figure 6.6.

6.5.4 Estimating the Relative Prediction Error

In order to estimate the download success probability, it is not sufficient to perform a time series prediction because the uncertainty of such a prediction can be quite high (as shown previously) and because it can vary across different network environments. Although there are approaches that allow to explicitly predict a distribution such as

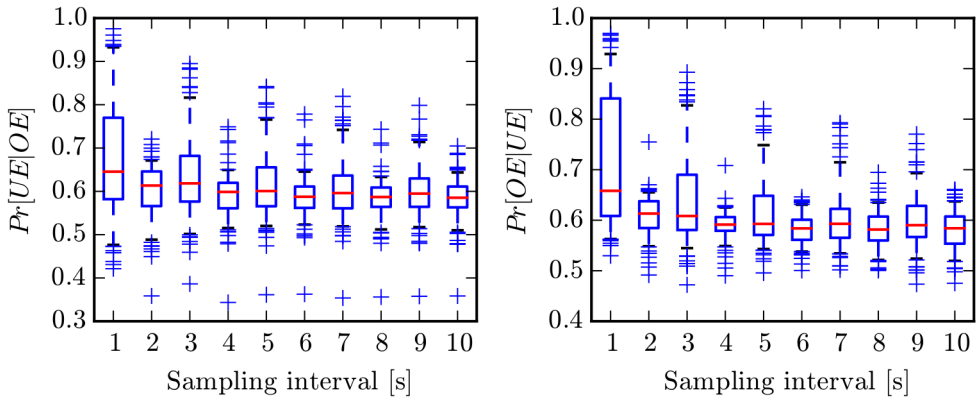

Figure 6.6: Per-trace probability that an underestimation is followed by an overestimation and vice versa. Horizontal line: median, box: quartiles, whiskers: 0.5 and 0.95 quantiles, flier points: outliers. See Section 6.5.3 for details.

Gaussian process method [161], approaches such as SMA do not have this capability. Therefore, we complement the predicted value by an estimate of the relative prediction error distribution. A straightforward approach is to use the ECDF of past prediction errors, and to account for the long-term non-stationarities by discarding values whose age exceeds a certain threshold. Another approach is to select a distribution type and to fit its parameters dynamically from the data. In our evaluation, we will use the former method, since it results in good performance and since with the latter method the computation of the model parameters involves an optimization step, which is resource-consuming.

Nevertheless, we would like to briefly present our results on fitting several well-known distribution types to the relative prediction errors: exponential, normal, logistic, and Lomax (shifted Pareto) [101]. We observe that the Lomax distribution provides the best fit. We therefore recommend to use the Lomax distribution to model prediction errors when evaluating adaptation approaches with synthetic data, as done by Yin et al. [220], for example.

For the underestimation errors, distributions are truncated to the range $[0, 1]$, and for the overestimation errors, to the range $[0, \infty)$. The Cumulative Distribution Function (CDF) $F_{\mathrm{tr}}(\cdot)$ of a distribution truncated to $[a, b]$ is obtained from the original CDF $F(\cdot)$ as

$$F_{\mathrm{tr}}(x) = \frac{F(x) - F(a)}{F(b) - F(a)}, \quad x \in [a, b]\,.$$

We fit a distribution to the data by minimizing the squared distance (L^2-norm) between its CDF and the truncated ECDF. In order to make the fit more precise in the range which is relevant for adaptive streaming clients, we truncate ECDF's to the interval $[0.1, 5.0]$ for the overestimation errors, and to the interval $[0.1, 1.0]$ for the underestimation errors. Afterwards, Kolmogorov-Smirnov test is used to verify the goodness of the fit [74].

The results are shown in Figure 6.7. The CDF's are fitted to ECDF's over the

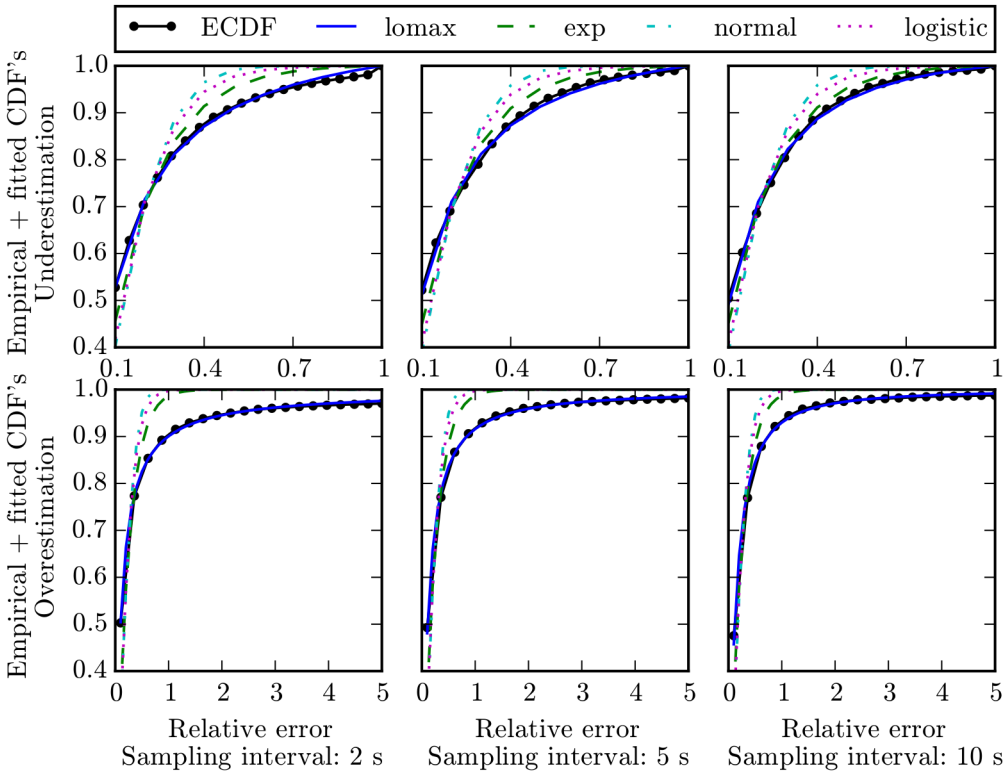

Figure 6.7: Fitting distributions for the relative prediction errors. See Section 6.5.4 for details.

joined set of data points from all traces. It turns out that both the underestimation and the overestimation errors are extremely well represented by a Lomax distribution. These findings are consistent with those obtained by fitting the prediction errors from individual traces, which are omitted here.

6.5.5 Estimating the Download Success Probabilities

In this section, we describe our approach to estimating download success probabilities $\left(P_{ij}^{p}, j \in \{0, \ldots, |\mathcal{R}| - 1\}\right)$ that at time t_i^r s_{ij} bytes can be downloaded in representation j before its playback deadline t_i^p. We denote by $T_{\max} \in \mathbb{N}$ the maximum prediction horizon in seconds. Consequently, at time $t \in \mathbb{N}$, the client computes the average application layer throughput (as defined in (4.2)) for time intervals $[t - T, t]$ for $T \in \{1, \ldots, T_{\max}\}$, followed by computing throughput predictions for time intervals $[t, t + T]$. If the throughput cannot be computed, no prediction is provided either. Finally, the client computes the relative prediction error for the interval $[t - T, t]$ as

$$\tilde{\epsilon}\left(t - T, t\right) = \frac{\max\left(\hat{\rho}\left(t - T, t\right), \rho_{\min}\right) - \max\left(\rho\left(t - T, t\right), \rho_{\min}\right)}{\max\left(\rho\left(t - T, t\right), \rho_{\min}\right)}. \tag{6.3}$$

In contrast to the definition in (6.2), $\tilde{\epsilon}\,(t-T,t) \in (-1,\infty)$ is defined without taking the absolute value for the purposes of presentation.

We assume that predictions are computed every second, so that at time t_i^r, the most recent predictions were computed at time $\lfloor t_i^r \rfloor$. In order to calculate the download success probabilities, the client determines the smallest time interval containing $[t_i^r, t_i^p]$ for which a prediction is available. Note that it is not necessarily $[\lfloor t_i^r \rfloor, \lceil t_i^p \rceil]$ since, due to the distribution of inter-request delays or due to a throughput outage, a prediction for this time interval might not be available.

Let t_i^π, $T^* \in \mathbb{N}$ be determined such that $[t_i^\pi, t_i^\pi + T^*]$ is the shortest time interval containing $[t_i^r, t_i^p]$ for which a prediction is available, and let $\hat{\rho}_i = \hat{\rho}\,(t_i^\pi,\, t_i^\pi + T)$ be the corresponding throughput prediction. $\epsilon_i = \epsilon\,(t_i^\pi,\, t_i^\pi + T)$ and $\tilde{\epsilon}_i = \tilde{\epsilon}\,(t_i^\pi,\, t_i^\pi + T)$ shall denote the relative prediction errors, as defined in (6.2) and (6.3). Further, we denote by $\Phi_i^u\,(\epsilon_i)$ and $\Phi_i^o\,(\epsilon_i)$ the estimated CDF of the underestimation and overestimation errors for $\hat{\rho}_i$, computed at t_i^π. Finally, $P_i^u \in [0,1]$ shall denote the relative frequency of underestimations.

With the introduced notation, the estimated CDF for $\tilde{\epsilon}_i$ is given by

$$\Phi_i\,(\tilde{\epsilon}_i) = \begin{cases} P_i^u \cdot \Phi_i^u\,(\epsilon_i) & \text{for } \tilde{\epsilon}_i < 0 \\ P_i^u + (1 - P_i^u) \cdot \Phi_i^o\,(\epsilon_i) & \text{otherwise}\,. \end{cases} \tag{6.4}$$

Consequently, the download success probability P_{ij}^p can be estimated as

$$P_{ij}^p = P\left[\frac{s_{ij}}{t_i^p - t_i^r} \leq \frac{\hat{\rho}_i}{1+\tilde{\epsilon}_i}\right] = \Phi_i\left(\frac{\hat{\rho}_i\,(t_i^p - t_i^r)}{s_{ij}} - 1\right)\,. \tag{6.5}$$

6.6 Evaluation

We evaluated the performance of LOLYPOP using our collected throughput traces and comparing it against the state-of-the-art algorithm FESTIVE [98] as a baseline. The setting and results are presented in the following.

6.6.1 Evaluation Setting

We implemented both LOLYPOP and FESTIVE in a streaming client prototype written in Python[2]. We equipped the developed prototype with a feature that allowed it to be executed in virtual time using a throughput trace file as input, thus allowing for a simulative evaluation using collected traces.

We used Big Buck Bunny [17] as video content, which is an animated movie of approximately 10 minutes duration. This video was selected due to the availability of raw video data, allowing us to generate representations with high MMBR's. We encoded 9 representations with MMBR's distributed between 100 and 20000 kbps with exponentially increasing intervals: 101, 194, 377, 730, 1415, 2743, 5319, 10314, and 20000 kbps, using the H.264/MPEG-4 AVC [215] compression format. The chosen intervals correspond to a roughly linear increase of the video quality in terms of PSNR [184]. The

[2] http://www.python.org

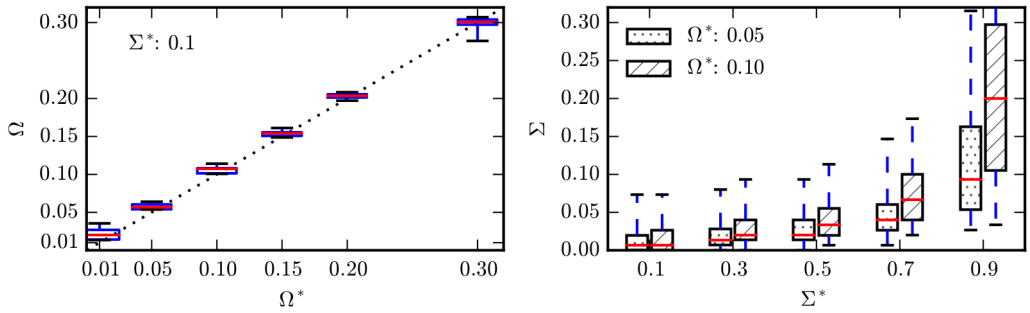

Figure 6.8: Ω as function of Ω^* (left), and Σ as function of Σ^* (right), for LOLYPOP. The distributions over the traces are shown as boxplots, where the horizontal line represents the median, the box represents the quartiles, and the whiskers represent the 0.05 and 0.95 quantiles.

encoding was performed using the avconv[3] utility using two passes, with a configuration targeting at low MMBR variations among individual segments.

The evaluation was performed using an upper bound on the transport latency of 3 seconds, corresponding to 1.5 times the segment duration. Neglecting segmentation overhead at the server and decoding overhead at the client, this corresponds to an overall live latency of 5 seconds. Each streaming session lasted for 5 minutes.

We evaluated LOLYPOP with different values for the configuration parameters Σ^* and Ω^*. The goal was to explore the range of operating points with $\Sigma \in [0, 0.1]$ and $\Omega \in [0, 0.5]$. We used $\Sigma^* \in \{0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$ and $\Omega^* \in \{0.001, 0.005, 0.008, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.15, 0.2, 0.3, 0.5\}$. In total, we evaluated 442 configurations. Note that we used Σ^* values that are much higher than the values for Σ we want to achieve. This is due to the observation that tight restrictions on the number of quality transitions Ω^* results in much lower numbers of skipped segments than the value used for Σ^*.

The FESTIVE adaptation algorithm was evaluated with a broad range of values around the default configuration evaluated by Jiang et al. [98]. We vary the values for α (controlling the trade-off between the average quality and the quality fluctuations), p (safety margin between the estimated bandwidth and the selected MMBR), and k (controlling the amount of quality fluctuations by enforcing a minimum distance between quality transitions). We used $\alpha \in \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20\}$, $p \in \{0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$, and $k \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50\}$. In total, we evaluated 2880 configurations. We disabled the randomizer feature of FESTIVE since it requires delaying requests. With low-delay streaming, the randomizer feature can lead to an increased number of skipped segments. We did not relax the restriction of FESTIVE that it switches the representation at most one step at a time since we considered it as one of its core features. Finally, we would like to point out that FESTIVE bases its decisions upon the knowledge of the MMBR of a representation, while LOLYPOP uses the segment size of the next segment.

[3] http://libav.org/avconv.html

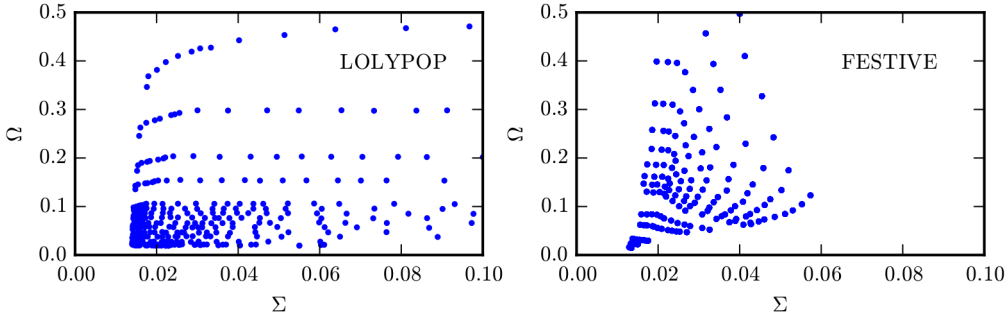

Figure 6.9: Scatter plots of covered (Σ, Ω) regions for LOLYPOP (left), and FESTIVE (right).

6.6.2 Evaluation Results

The evaluation goals are to understand the dependency of the reached operating point on the algorithm configuration, to explore the region of reachable operating points, and to evaluate the average video quality as a function of the operating point.

First, we study the dependency of the reached operating point, specifically of (Σ, Ω), on the input parameters Σ^* and Ω^*. Figure 6.8 (left) illustrates the ability of LOLYPOP to satisfy the upper bound on the number of quality transitions Ω^*, by depicting Ω as a function of Ω^* exemplarily for $\Sigma^* = 0.1$. The graphs for other values of Σ^* are almost identical and are omitted. We observe that LOLYPOP is able to enforce the upper bound on the number of quality transitions quite accurately. One reason for the slight overshoot is that we always allow downward quality transitions. Also note that a value of $\Omega = 0.01$ means that during the whole streaming session, there are only 3 quality transition, which, in a wireless network, is an extremely low value. Figure 6.8 (right) illustrates the dependency of Σ on Σ^* for two values of Ω^*: 0.05 and 0.1. We observe that Σ is significantly lower than Σ^* and that a lower value for Ω^* decreases Σ even further. The intuition behind that is that whenever Ω^* is exceeded during the course of a streaming session, only downward quality transitions are permitted.

Next, we evaluate the region of reachable operating points. The broader this region the more flexible the algorithm can be tuned to the QoE requirements defined for a streaming session. Figure 6.9 shows scatter plots of achieved (Σ, Ω) values for LOLYPOP (left) and FESTIVE (right). We observe that the studied LOLYPOP configurations cover a broader range of (Σ, Ω) values and thus enable a more flexible adjustment to user and/or application profiles. Note that a low value of Σ and/or Ω alone is not an indicator of high QoE since it might be achieved by selecting an unnecessary low video quality.

The fact that Σ values below 0.01 were not achieved by either algorithms is in part explained by the throughput outages of several seconds durations contained in several traces. In order to quantify these "unavoidable" fractions of skipped segments, we simulate streaming sessions using an adaptation algorithm that always selects the lowest quality. We observed that out of 92 used traces, 66 support streaming at lowest quality without skipped segments. Furthermore, 7 traces have a Σ below 0.01, further 10 below

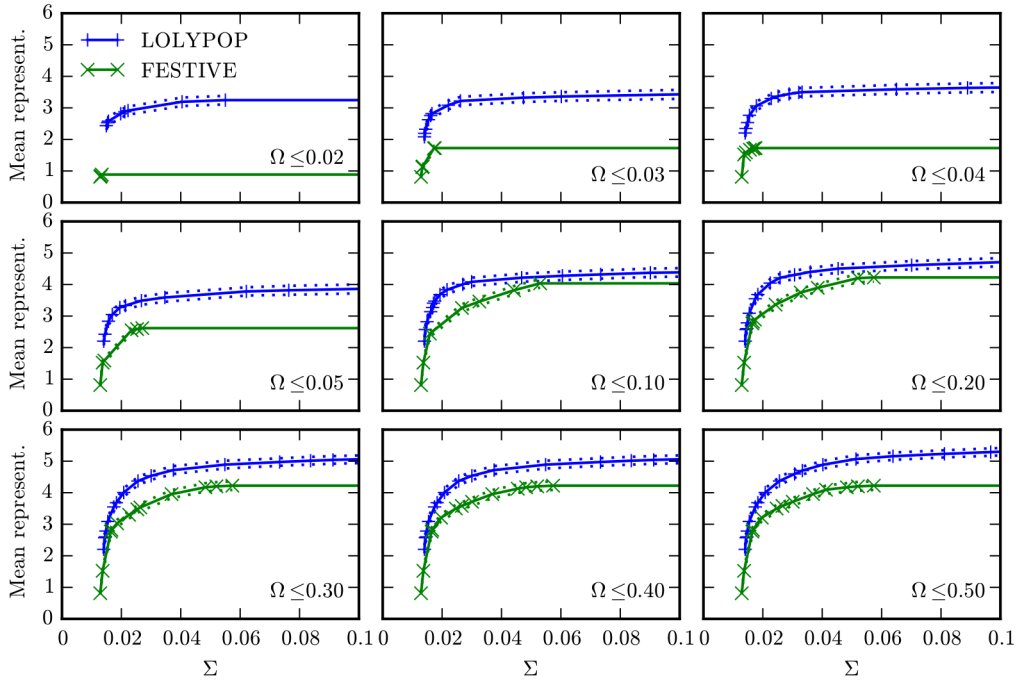

Figure 6.10: Average video quality as a function of the number of skipped segments Σ for different numbers of quality transitions Ω. Dashed lines represent the confidence intervals for the confidence level 0.95.

0.05, further 7 below 0.1, one had 0.12 and one has the highest Σ of 0.15.

As the main result of our evaluation, we would like to characterize the average video quality as a function of the operating point. Figure 6.10 visualizes the average video quality as a function of quality transitions for different numbers of skipped segments. For different values of Σ (on the x-axis), we first determine the configuration that (i) achieves the highest average video quality over all traces, (ii) whose average fraction of skipped segments is less or equal to Σ, and (iii) whose average (relative) number of quality transitions is less then or equal to Ω, where $\Omega \in \{0.02, 0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$. The convex hulls of the resulting curves are depicted in Figure 6.10. It is accompanied by the confidence interval for the confidence level of 0.95, computed after subsampling the data with a subsampling factor of 0.1 to remove temporal correlation [119].

We observe that LOLYPOP achieves a higher average quality at all operating points. The difference is particularly pronounced for small numbers of quality transitions, where LOLYPOP achieves an up to 3 times higher average quality. For high numbers of quality transitions, the difference slightly increases with the number of skipped segments. An interesting observation is that all plots have a more or less pronounced "knee", after which the curve goes into saturation and the quality does not increase significantly. In contrast, before the knee, a small increase in the number of skipped segments can bring a huge increase in video quality. In the evaluated network environments, the "knee" is typically slightly below $\Sigma = 0.02$. In other words, accepting 0.5 to 1 percent more skipped segments, which corresponds to one to two more skipped segments every 400

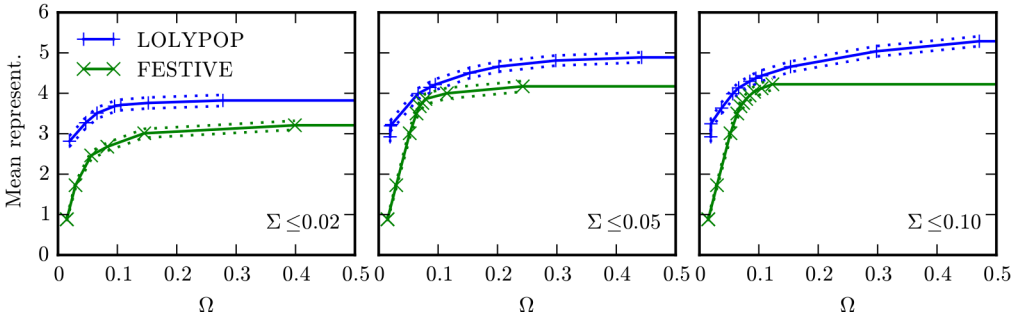

Figure 6.11: Average video quality as a function of the number of quality transitions Ω for different numbers of skipped segments Σ. Dashed lines represent the confidence intervals for the confidence level 0.95.

seconds, can result in an up to twofold improvement in video quality (e.g., for $\Omega \leq 0.2$).

While Figure 6.10 shows mean values over all 92 traces, we generated similar plots for each trace individually. In 31 traces, all 9 considered Ω thresholds resulted in both curves having the same ranges and could thus be compared pointwise. In 21 out of the 31 traces, all 9 curves for LOLYPOP were pointwise strictly greater than the corresponding curves for FESTIVE, while there existed no trace where all 9 curves were pointwise greater for FESTIVE. Furthermore, in order to perform a trace-by-trace comparison across all traces, including those in which some curves had different ranges or were intersecting, we compared the integrals of the curves. This comparison revealed that for $\Omega = 0.2$, in 53% of traces, LOLYPOP had a higher integral than FESTIVE; in 38% of traces, FESTIVE had a higher integral; and in the remaining traces, the values were equal. The corresponding values for $\Omega \in \{0.03, 0.04, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ are (76%, 22%), (82%, 16%), (76%, 22%), (78%, 20%), (86%, 12%), (87%, 11%), (87%, 11%), (89%, 9%). We thus observe that for all considered Ω thresholds, in the majority of traces, the performance of LOLYPOP averaged over the considered range of Σ values is higher than the performance of FESTIVE.

Similarly to Figure 6.10, Figure 6.11 presents plots of quality vs. quality transitions for different levels of skipped segments. Here, we observe a similar situtation, albeit the "knee" effect is less pronounced in the case of LOLYPOP due to the relatively high achieved video quality for low values of Ω.

Finally, Figure 6.12 depicts four example runs illustrating the behavior of the proposed algorithm with different configurations. For each run, three plots are shown. The top one depicts network throughput and segment MMBR. The middle one depicts the representations selected for individual segments and the mean value. The bottom one depicts the buffer level at playback deadlines (a value of 0 results in a skipped segment). In the three upper left subplots, we see a run with low values for both Σ^* and Ω^*. Setting Ω^* to 0.001, we effectively restrict the number of upward transitions to 1, since the whole streaming session has less than 1000 segments. We observe that the algorithm reacts not only to decreased throughput, as seen between seconds 30 and 50, but also to increased uncertainty in throughput dynamics as seen after the strong downward fluctuation at second 170. A higher Σ^*, as seen in the top right subplots, results in a more

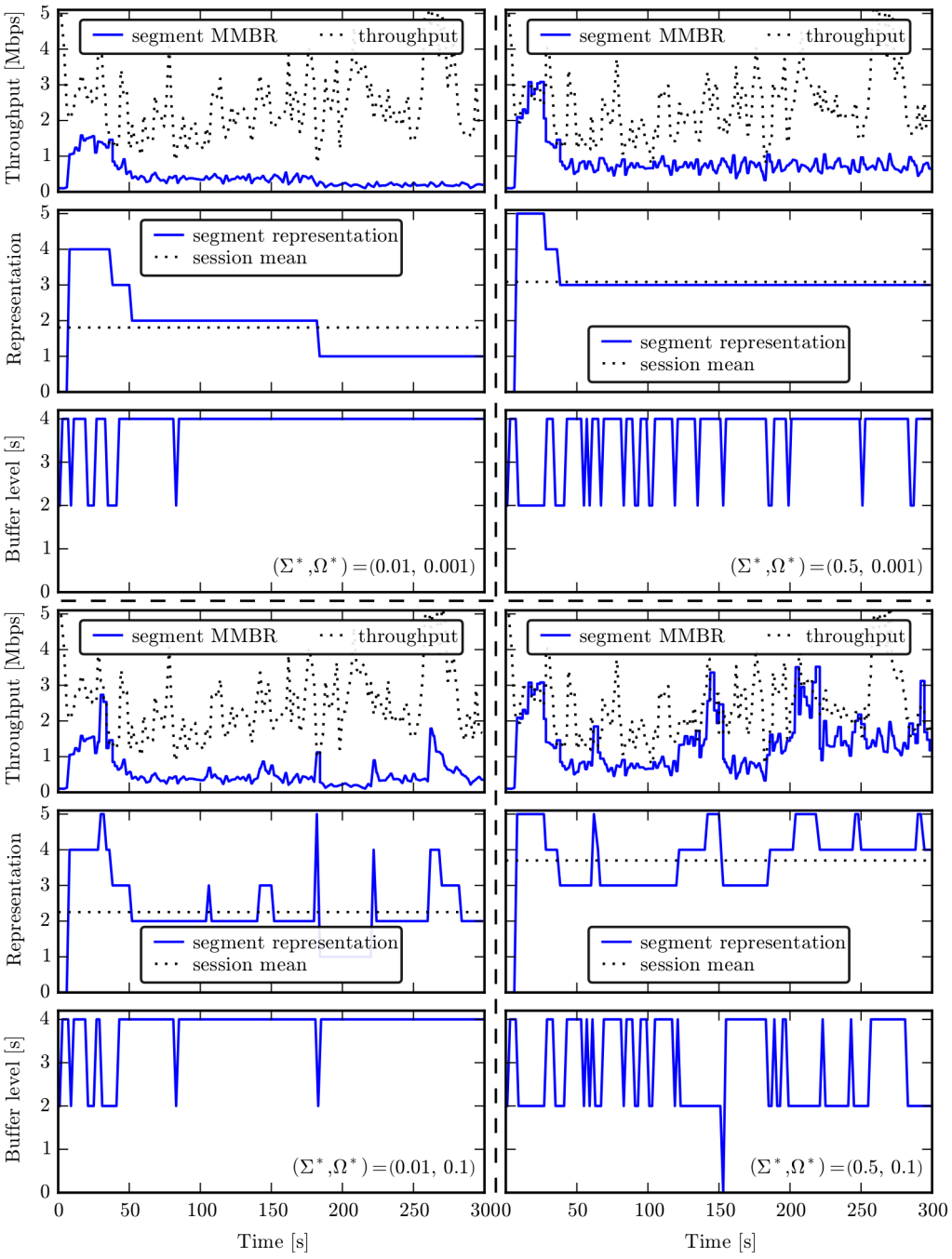

Figure 6.12: Four example runs with different algorithm configurations. See Section 6.6.2 for details.

aggressive behavior accepting a higher probability for skipping a segment and a higher average quality. The two sets of subplots at the bottom depict runs with $\Omega^* = 0.1$. The algorithm is allowed to have more quality transitions, resulting in a further improvement of the average video quality.

Chapter 7

Adaptation Algorithm for Video on Demand

In this chapter, we present a novel adaptation algorithm for HAS-based VoD called TOBASCO (**T**hresh**o**ld-**B**ased **A**daptation **Sc**heme for **o**n-Demand Streaming). In contrast to the system presented in Chapter 5, TOBASCO does not rely on a cooperation with the network, and is therefore suited for a standalone deployment in any network environment. It also does not rely on cross-layer information, and can thus be deployed on a broad range of platforms, including Set-Top Boxes (STB's) and web browsers. TOBASCO has a flexible parametrization, and thus can be adjusted to various service provider and user requirements and QoE expectations. Moreover, even though designed for VoD, it can be parametrized to operate with a small maximum buffer level of 10 to 20 seconds, which makes it suitable for live streaming with a moderate live latency, that is, without the low delay requirement.

7.1 Introduction

There is little doubt that techniques that leverage cross-layer information can greatly improve both the performance and efficiency of multimedia applications [170, 204, 222, 228]. Still, the deployment of such techniques is often problematic due to the required communication with the lower layers of the protocol stack or the required support from the network infrastructure. After all, the strict modularization of the Open Systems Interconnection (OSI) reference model [233], and the end-to-end design principle [171] were among the factors that paved the way for the success of the Internet.

Consequently, in this chapter, we present our design of an adaptation algorithm for

VoD that operates as a standalone application, which means that it does not require cross-layer information nor any coordination or support from the network. The only information used as input is the past throughput dynamics, and the buffer level dynamics, which can both be monitored by the streaming client itself.

A major design goal when developing TOBASCO was to obtain a flexible approach that can be parametrized to support a wide range of performance requirements that correspond to different service provider policies and business models, as well as diverse QoE requirements and expectations of the users. For example, TOBASCO can resolve the trade-offs between the amount of playback interruptions, the continuity of the video quality, and the maximization of the video quality in multiple ways, controlled by its configuration parameters.

Last but not least, TOBASCO was designed to support a variety of network environments. It can be adjusted to efficiently operate over a stable and fast dedicated wired link, or exposed to a highly fluctuating low throughput in a busy wireless cell.

While offering a high degree of flexibility, the parametrization of the presented algorithm is straightforward and intuitive enough to simplify the deployment and to enable its integration into various streaming solutions. Finally, TOBASCO has a low computational complexity which helps reducing the energy consumption and increase its practicality since some entertainment devices supporting video do not have the computational power of PC's or smartphones.

TOBASCO has been integrated in several video player frameworks, such as the GStreamer [66], VLC [206], and TAPAS [36]. It has been evaluated in emulated and real-world network environments, using the commercial HAS player Microsoft Silverlight [142] as the baseline approach. Further, its performance has been evaluated against the optimal adaptation trajectories computed by the omniscient client presented in Chapter 8. The evaluation shows that TOBASCO allows to efficiently avoid playback interruptions, provides a smooth viewing experience by avoiding excessive video quality fluctuations, achieves a high level of network resource utilization, and provides a fair resource allocation in a multi-user environment. Moreover, it minimizes start-up delays, which is particularly important for services, where users tend to frequently start new video sessions. In particular, in the network environment used for the evaluation, the developed algorithm achieves an average video bit rate which is by up to 35% higher than the baseline approach, and within up to 85% of the optimum, at an up to an order of magnitude smaller rebuffering duration.

7.2 Design Goals

Depending on the targeted deployment scenario, a streaming client may have different design goals. In most cases, the main goal is to deliver the best possible QoE, subject to the availability of the network resources. In addition, however, it is typically desirable to maintain basic fairness when operating in multi-user environments. Finally, a streaming client might be required to minimize costs engendered by content that is downloaded but not presented to the user. (The latter happens when a user prematurely quits a streaming session but also when, based on an adaptation decision, prebuffered low-quality content is discarded to be replaced by the same content in a higher-quality representation.)

For the algorithm presented in this chapter, the highest weight was put on maximizing the QoE for the individual users, as is typical for applications operating over best-effort networks, such as the open Internet. It is also worth noting, that due to the operation on top of TCP, basic fairness of end-to-end resource allocation is already provided by its built-in congestion avoidance and congestion control mechanisms. Nonetheless, in our design, we explicitly account for design aspects important for operating in multi-user environments.

As described in Section 2.2, quantifying QoE for an adaptive streaming session is a complex and partially open research question. For our general-purpose algorithm, we therefore focus on the four most important factors influencing user's perception: (i) start-up delay, (ii) rebuffering, (iii) mean video quality, and (iv) video quality fluctuations. Note that these goals consitute several trade-offs. E.g., downloading each segment in best possible representation results in frequent changes of playback quality whenever the dynamics of the available throughput exhibit strong fluctuations. Also, minimizing start-up delay implies minimizing initial video quality.

For the presented algorithm, the goal of minimizing rebuffering has been assigned the highest priority since it was shown to have a dramatic impact on QoE. Further, we tried to achieve a balance between mean quality and quality fluctuations, which can be tailored to specific deployment requirements by a proper parameterization. We minimize start-up delay by always downloading the first segment in lowest quality but, at the same time, we introduce a fast start phase that quickly increases video quality in a more aggressive way than during normal operation. For all performance aspects we introduce configuration parameters that allow to tailor the behavior to individual requirements such as, e.g., user's viewing preferences, deployment environments, or specifications of the content provider.

7.3 TOBASCO — Adaptation Algorithm for Video on Demand

In this section, we describe the operation of TOBASCO in details. We present the pseudocode, introduce the configuration parameters, and describe in details the two operation modes: the adaptation phase, and the fast start. We use the notation defined in Chapter 4 with minor extensions specific to the presented approach that are summarized in Table 7.1.

7.3.1 General Idea

Similar to the ideas that led to the design of JINGER presented in Chapter 5, the general idea behind TOBASCO is to stabilize the buffer level around a certain target value, which is high enough to prevent buffer underruns during the periods of low throughput or link outages.

Recall that the buffer level, the media bit rate of the transmitted video segment, and the network throughput are connected by the following continuous-time approximation

$$\dot{\beta}(t) = \frac{\rho(t)}{r(t)} - 1\,.$$

Here, $\beta(t)$ is the buffer level at time t, representing the duration of the stored video content in seconds (defined in (4.1)), $\rho(t)/r(t)$ is the buffer filling rate due to the incoming video data, while -1 is the buffer depletion rate due to the ongoing playback. In contrast to the design of JINGER, the only variable that can be controlled by a standalone client such as TOBASCO is the media bit rate $r(t)$, or, more precisely, the segment MMBR, as defined in Chapter 4.

Based on insights from the control theory, in our design we take into account both the deviation from the target buffer level and the rate of buffer level change, which is affine to the network throughput.

Roughly speaking, we increase the video quality when the buffer level is "high enough" and decrease it when it is "too low", similar to the proportional component of a PID controller. Keeping it as simple as that, however, would inevitably result in excessive quality fluctuations, annoying the user and degrading the QoE. Consequently, instead of using a single target buffer level, we use a target buffer *interval*, within which the video quality is not changed. Thus, we introduce a configurable hysteresis effect into the dynamics of our system that prevents it from reacting to short-term throughput variations, if the risk of a buffer underrun is low.

Another aspect we need to address is related to the fact that segment MMBR can only take values from a discrete set, determined by the set of available representations. It is, therefore, not possible to exactly match it to the available throughput. As a result, without specific countermeasures, buffer level will be constantly increasing whenever the MMBR of the selected representation is smaller than the throughput, and constantly decreasing, whenever it is greater than the throughput. At the same time, however, we expect a video client to stick to one representation in a network that offers a constant throughput, or a throughput fluctuating with low amplitude or high frequency. The only way to achieve that is to introduce inter-request delays whenever the selected MMBR is less than the expected throughput but the next-higher MMBR is already too large.

On the other hand, introducing inter-request delays can lead to undesired effects when multiple streaming clients share a common bottleneck resource. When a client enters an inter-request delay, due to TCPs constant probing for free bandwidth, other clients will try to utilize the freed resources by increasing their throughput. This can lead to various effects and reduce overall fairness and efficiency. In our work, we try to minimize the duration of individual delays by uniformly distributing them across subsequent segment downloads.

7.3.2 Algorithm Description

The listing in Algorithm 2 presents the pseudo-code for the developed algorithm. In the following, we will explain its individual elements in detail.

We assume that the algorithm is invoked at time t^c_{i-1}, immediately after the download of segment $i-1$ has been completed. In order to efficiently adapt the video quality to the throughput dynamics the algorithm takes two types of input: (i) past throughput dynamics, and (ii) past playback buffer dynamics.

The algorithm computes two output values: (i) a representation for the download of the next segment i, and (ii) the inter-request delay. The latter is expessed as the

$0 \leq B_{\text{min}} < B_{\text{low}} < B_{\text{high}}$	Minimum, low, and high buffer level thresholds
$\mathcal{B}_{\text{tar}} = [B_{\text{low}}, B_{\text{high}}]$	Target buffer interval
$B_{\text{opt}} = 0.5\,(B_{\text{low}} + B_{\text{high}})$	Optimum buffer level
$\Delta_\beta > 0$	Duration of the time interval for computing β_{min}
$\Delta_t > 0$	Duration of the time interval for thrpt. averaging
$0 < \alpha_1, \ldots, \alpha_5 \leq 1$	Safety margins for throughput estimation

Table 7.1: Notation extensions for TOBASCO

maximum buffer level B_{delay}, above which the download of the next segment shall not be started. The purpose of the inter-request delays is twofold. On the one hand, they are reuqired to bound the buffer level from above if we already arrived at the highest representation. But even more importantly, they are required to stabilize the buffer level in the middle of the target interval when the selected MMBR does not exactly match the network throughput, which is the typical scenario, since the set of the available representations is finite. In the following sections, the operation of TOBASCO will be explained in more details.

TOBASCO can be configured using the following set of parameters: $0 \leq B_{\text{min}} < B_{\text{low}} < B_{\text{high}}$, $\Delta_\beta > 0$, $\Delta_t > 0$, $0 < \alpha_1, \ldots, \alpha_5 \leq 1$. They will be described in more details in the following.

At the start of a streaming session, the algorithm always selects the lowest representation for the first segment to be downloaded. The advantage of this approach is that it minimizes the prebuffering delay $t_0^p - t_0^r$. The disadvantage is, that the first seconds of the video may be downloaded at a lower quality than what can be sustained by the network. In order to mitigate this effect, we introduce a *fast start* phase at the beginning of the streaming session, with the goal to increase the video quality in a more aggressive manner. In the following, we first describe the operation of the algorithm during the phase which follows the fast start phase, which we call the *adaptation phase*.

7.3.3 Adaptation Phase

We define three thresholds for the buffer level: $0 \leq B_{\text{min}} < B_{\text{low}} < B_{\text{high}}$. We use $\mathcal{B}_{\text{tar}} = [B_{\text{low}}, B_{\text{high}}]$ to denote the target interval, and $B_{\text{opt}} = 0.5\,(B_{\text{low}} + B_{\text{high}})$ to denote its center point. The algorithm tries to keep the buffer level close to B_{opt}. Note that the only way to increase $\dot{\beta}(t)$, that is, the rate at which the buffer level is changing, is to switch to a lower representation, while there are two ways to decrease $\dot{\beta}(t)$. The first is to switch to a higher representation, while the second is to introduce inter-request delays.

While $\beta(t) \in \mathcal{B}_{\text{tar}}$, we never switch to a different representation. The reason behind it is not to react to short-term variations of the available throughput. If we observe a positive or negative short-term throughput spike on an otherwise constant channel, we do not want to change to a different representation since shortly afterward we would

Algorithm 2: TOBASCO

Input: $\beta\left(t_{i-1}^{c}\right)$, ρ_{i-1} ▷ *Current buffer level, last segment throughput*
Input: $\rho\left(t_{i-1}^{c}-\Delta_t,\, t_{i-1}^{c}\right)$ ▷ *Mean throughput for past Δ_t seconds*
Input: $\left(\beta_{\min}\left(k\Delta_\beta,(k+1)\Delta_\beta\right),\ k=0,\ldots,\lfloor t_{i-1}^{c}/\Delta_\beta\rfloor\right)$ ▷ $\beta_{\min}$ *time series*
Output: r_i, B_{delay} ▷ *Selected representation, inter-request delay*

```
static fast_start := true                                   ▷ Initial mode is the fast start
B_delay := 0                                                ▷ Default behavior: no inter-reqeuest delay
r_i := r_{i-1}                                              ▷ Default behavior: no quality change
if fast_start                                               ▷ Continuing fast start phase if (i) not at
  ∧ r_{i-1} ≠ r_max                                         ▷ highest representation yet, (ii) β_min
  ∧ β_min(kΔ_β,(k+1)Δ_β) ≤ β_min(k'Δ_β,(k'+1)Δ_β), ∀ 0 ≤ k < k' ≤ ⌊t^c_{i-1}/Δ_β⌋
  ∧ r_{i-1} ≤ α_1 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then      ▷ increasing, (iii) throughput high enough.
    if β(t^c_{i-1}) < B_min then                            ▷ Buffer level below B_min
      if r^↑_{i-1} ≤ α_2 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then   ▷ Throughput high enough
        r_i := r^↑_{i-1}                                    ▷ Increase video quality
    else if β(t^c_{i-1}) < B_low then                       ▷ Buffer level in [B_min, B_low]
      if r^↑_{i-1} ≤ α_3 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then   ▷ Throughput high enough
        r_i := r^↑_{i-1}                                    ▷ Increase video quality
    else                                                    ▷ Buffer level above B_low
      if r^↑_{i-1} ≤ α_4 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then   ▷ Throughput high enough
        r_i := r^↑_{i-1}                                    ▷ Increase video quality
      if β(t^c_{i-1}) > B_high then                         ▷ Buffer level above B_high
        B_delay := B_high − τ                               ▷ Delay next request
else                                                        ▷ We are in the adaptation phase
    fast_start := false                                     ▷ Set state variable
    if β(t^c_{i-1}) < B_min then                            ▷ Buffer level below B_min
      r_i := r_min                                          ▷ Selecting lowest quality
    else if β(t^c_{i-1}) < B_low then                       ▷ Buffer level below B_low
      if r_{i-1} ≠ r_min ∧ r_{i-1} ≥ ρ_{i-1} then           ▷ Throughput too low
        r_i := r^↓_{i-1}                                    ▷ Decrease quality
    else if β(t^c_{i-1}) < B_high then                      ▷ Buffer level in [B_low, B_high]
      if r_{i-1} = r_max                                    ▷ Already at highest quality, or
        ∨ r^↑_{i-1} ≥ α_5 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then   ▷ throughput too low to switch up
        B_delay := max(β(t^c_{i-1}) − τ, B_opt)             ▷ Delay request if buffer level above B_opt
    else                                                    ▷ Buffer level above B_high
      if r_{i-1} = r_max                                    ▷ Already at highest quality, or
        ∨ r^↑_{i-1} ≥ α_5 · ρ(t^c_{i-1} − Δ_t, t^c_{i-1}) then   ▷ throughput too low to switch up
        B_delay := max(β(t^c_{i-1}) − τ, B_opt)             ▷ Delay next request
      else                                                  ▷ We're not at highest quality
        r_i := r^↑_{i-1}                                    ▷ Increase the quality
```

be forced to change back. This behavior would decrease QoE by introducing quality fluctuations. Thus, we configure the sensitivity of the algorithm to throughput spikes by adjusting the size of the target interval $\mathcal{B}_{\text{tar}}$.

Now, assume that we observe a decrease of the available throughput. Once the buffer level falls below B_{low}, we react to this change by selecting a lower representation. We continue to switch to a lower representation as long as we are below B_{low} *and* the

segment throughput of the last downloaded segment is smaller than the MMBR of the currently selected representation. We use the latter condition as an approximation to the condition that the instantaneous buffer level changing rate is positive:

$$\frac{\rho_{i-1}}{r_{i-1}} \approx \frac{\rho\left(t_{i-1}^{c}\right)}{r\left(t_{i-1}^{c}\right)} > 1 \Leftrightarrow \dot{\beta}\left(t_{i-1}^{c}\right) > 0\,. \tag{7.1}$$

Condition (7.1) prevents the client from excessively undershooting the available throughput – if we are below B_{low} but the buffer level is increasing, there is no need to change the representation. This corresponds to the influence of the derivative action of a PID controller.

Next, assume that we observe an increase of the available throughput. Once the buffer level increases beyond B_{high}, we react based on the following two options. We can either decide to stay with the current representation and to delay the subsequent download, or we decide to select the next higher representation. Our decision here depends on the mean segment throughput measured during the last Δ_t seconds. If the MMBR of the next higher representation is below a preconfigured fraction of the mean segment throughput: $\bar{r}_{i-1}^{\uparrow} < \alpha_5 \cdot \rho\left(t_{i-1}^{c} - \Delta_t, t_{i-1}^{c}\right)$, we prefer to keep the current representation and to delay the request. Otherwise, we select the next higher representation. Note that in this expression, selecting $\alpha_5 < 1$ increases the robustness of the algorithm to measurement uncertainties. If we decide to keep the representation, we need to delay the subsequent download in order to bring the buffer level back to the target interval.

The presented mechanisms already allow to efficiently avoid playback interruptions by stabilizing the buffer level within the configured interval. Leaving it like that, however, would result in a buffer level that is constantly fluctuating around B_{high}, instead of staying close to B_{opt}, thus unnecessarily increasing the sensitivity of the client to positive throughput spikes. In fact, if the buffer level is above B_{opt} and rising but we know that the conditions for increasing the representation are not fulfilled, there is no reason to wait until $\beta(t)$ reaches B_{high} before introducing inter-request delays. Therefore, we add the following mechanism. Whenever the buffer level is in $[B_{\text{opt}}, B_{\text{high}}]$ but the expected network throughput is not high enough to increase the representation, we delay subsequent downloads until the buffer level falls back to B_{opt}. By keeping the buffer level close to B_{opt} we achieve a symmetric sensitivity to both positive and negative throughput spikes.

Here, we would like to point out that although we could bring the buffer level to its optimum value much faster by delaying a download until the buffer level falls back to B_{opt}, this approach has a strong disadvantage that comes into play when multiple clients share a common link. In fact, if one of the clients delays a request by a certain time, the TCP congestion control of the other client will be able to increase its throughput, potentially triggering an increase of the video quality. Once, however, the first client resumes download, both instances of TCP will again converge to the fair share of the available bandwidth forcing the second client to adjust his representation once again. Consequently, in order to avoid these unnecessary fluctuations, we try to minimize the duration of individual inter-request delays.

Finally, as an additional precaution to prevent buffer underruns, we immediately switch to the lowest available bit rate, whenever the buffer level falls below the config-

urable threshold $B_{\min}$. The reason is that underruns have a dramatic impact on QoE and thus their suppression must have the highest priority. Whenever $\beta(t) < B_{\min}$, there is a high probability of a buffer underrun in the presence of throughput fluctuations.

Note that we restrict the client to switching one representation at a time in order to avoid abrupt quality transitions with a high magnitude. If video content is offered with a high number of representations, such behavior would increase the risk of underruns since the time to reach the desired representation might be too long. However, we assume that due to the high storage costs and the little benefit from offering a dense representation set, video services will be typically provided with a small or moderate number of representations.

In summary, roughly speaking, available configuration parameters influence the adaptation strategy in the following way. The lower threshold of the target buffer interval influences the average buffer level. Thus, low values increase the risk of buffer underruns but they increase the "liveness" of the streaming session, which is important for transmission of live content. The size of the target buffer interval influences the sensitivity to throughput fluctuations. Low values let the video client switch the video quality more often in order to more closely follow throughput fluctuations and thus increase the average video quality by better utilizing the available bandwidth. On the other hand, high values provide a more steady video quality. Δ_t has a similar impact on the sensitivity to throughput spikes, especially to positive ones, since the client uses average throughput from past Δ_t seconds in order to decide if an increase of video quality can be sustained by the network.

7.3.4 Fast Start Phase

As already mentioned, at the beginning of a streaming session, the algorithm selects the lowest representation for the first segment in order to minimize the delay between the user's request to watch the video and the actual start of the playback. In order to quickly ramp up to the best quality that is feasible with the current throughput dynamics, we introduce a fast start phase that is more aggressive than the adaptation phase presented in the previous section. Entering adaptation phase right from the start of the streaming session would force the user to wait until the buffer level reaches B_{high} before increasing the quality of the video for the first time. Setting B_{high} to, e.g., 50 seconds would imply that the user will watch the first 50 seconds of the stream in lowest quality even on a high-speed link.

The fast start phase works as follows. For each subsequent download we select the next higher representation as long as its MMBR is below a certain fraction of segment throughput measured over the last Δ_t seconds. The fraction varies in three steps depending on the current buffer level. For $\beta\left(t_{i-1}^c\right) < B_{\min}$, we switch to a higher representation if $\bar{r}_{i-1}^{\uparrow} \leq \alpha_2 \cdot \rho\left(t_{i-1}^c - \Delta_t, t_{i-1}^c\right)$. For $\beta\left(t_{i-1}^c\right) \in [B_{\min}, B_{\text{low}})$ we use $\alpha_3 \geq \alpha_2$. Finally, for $\beta\left(t_{i-1}^c\right) \geq B_{\text{low}}$ we use $\alpha_4 \geq \alpha_3$. Thus, the algorithm becomes more and more aggressive the higher the buffer level.

The fast start phase terminates, when one of the following conditions is violated:

(i) $\bar{r}_{i-1} \neq \bar{r}_{\max}$,

(ii) $\beta_{\min}\left(k\Delta_\beta, (k+1)\Delta_\beta\right) \leq \beta_{\min}\left(k'\Delta_\beta, (k'+1)\Delta_\beta\right),\ \forall\, 0 \leq k < k' \leq \lfloor t_{i-1}^c/\Delta_\beta \rfloor$,

(iii) $\bar{r}_{i-1} \leq \alpha_1 \cdot \rho\left(t^c_{i-1} - \Delta_t, t^c_{i-1}\right)$.

Condition (i) lets us terminate the fast start phase after we arrive at the highest available representation. Condition (ii) ensures that we enter the adaptation phase if the buffer level minima over equidistant intervals are not monotonically increasing, which is more robust than to require monotonicity of the instantaneous buffer levels. Finally, condition (iii) forces us to quit the fast start phase when the MMBR of the selected representation approximates the mean segment throughput.

7.4 Evaluation

In this section we present the settings and the results of two evaluation campaigns for TOBASCO that have been performed using prototypical implementations. One implementation has been performed at the Technische Universität Berlin, where TOBASCO has been implemented as a plugin for the open source multimedia player VLC [206], using the C++ programming language. In addition, TOBASCO has been implemented it as a platform-independent software library using, again C++. The library provides an Application Programming Interface (API) that can be used to interface it with HTTP streaming clients. In particular, it has been integrated with one of the first MPEG-DASH client prototypes developed by STMicroelectronics[1], which is based on the open source multimedia framework GStreamer [66]. The GStreamer plugin has been developed for the Linux operating system and was intended to run on a STB box platform.

7.4.1 Evaluation Using an Emulated Wireless Cell

In this section, we present the setting and the results for the performance evaluation of TOBASCO, performed using an emulated IEEE 802.11a WLAN cell. The evaluation was performed using the plugin for the multimedia player VLC. As a baseline approach, we used the Microsoft Silverlight player [142]. In addition, we compared the performance with optimal adaptation trajectories computed by an omniscient client presented in Chapter 8.

7.4.1.1 Setting

For the evaluation we emulated a typical Internet path, starting with an IEEE 802.11a indoor WLAN access link. The setup is illustrated in Figure 7.1.

For the emulation of the wireless network, we used a very detailed site-specific model, developed based on the BOWL testbed [4], and implemented in the NS-3 network simulator [167]. The model consists of eight nodes, and each of the possible 56 unidirectional links is modeled separately. Seven of the nodes are used as stations, and one as the access point. Of the resulting downstream links, two offer a rather low average long-term TCP throughput of 1.4 and 1.7 Mbps, while the remaining five links offer around 20 Mbps on average. (Note that these values represent the throughput of the individual links in the absence of cross-traffic.) The emulation runs on a desktop PC with an Intel Core i7

[1] http://www.st.com

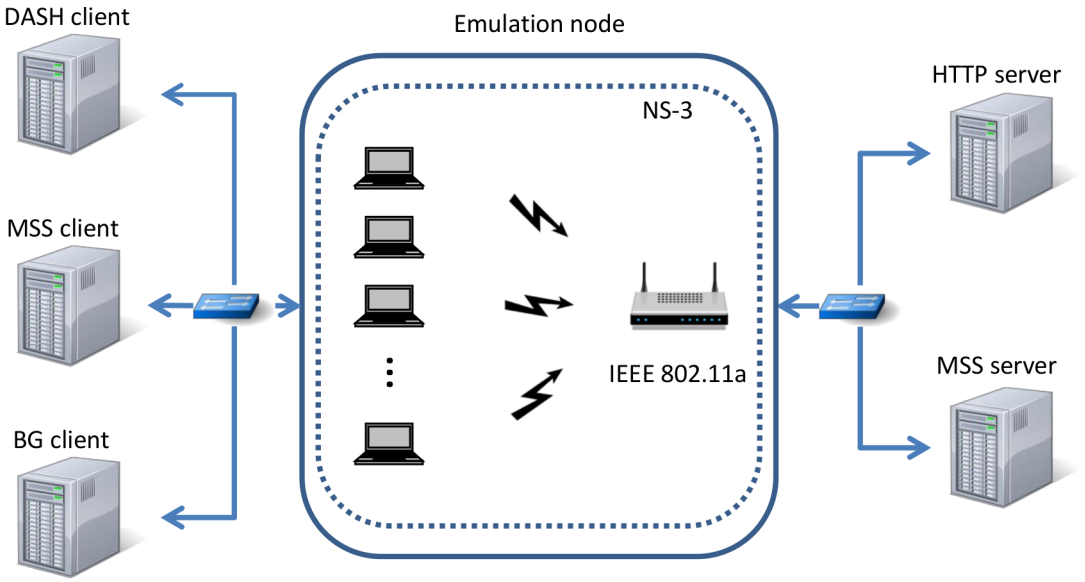

Figure 7.1: Evaluation setup. The access network is an emulated IEEE 802.11a WLAN, based on a site-specific model of the BOWL testbed [4].

CPU and 8 GB RAM, under the Linux operating system (Ubuntu 13.04). We will call this machine the *emulation host* in the following. The one-way delays on the emulated links connecting the stations and the access points with the Ethernet interfaces of the emulation host were set to a constant value of 1 ms.

The setting further contains two PC's, called *server hosts* in the following, hosting a Linux-based plain HTTP server (Apache 2), and a Windows-based Microsoft Internet Information Services (IIS) server. The former is used to serve MPEG-DASH-based video streams, as well as HTTP-based background traffic, while the latter is used to serve Microsoft Smooth Streaming video streams, used for a performance comparison. Finally, the setting is completed by three further PC's, called *client hosts* in the following, which are used to run MPEG-DASH clients (Linux-based), Microsoft Smooth Streaming clients (Windows-based), and HTTP background clients (Linux-based), respectively. The emulation host is connected to the client hosts and server hosts via a switched Gigabit Ethernet network.

The video traffic was always routed via the station with the second lowest maximum throughput of 1.7 Mbps. In addition to the video traffic, we generated synthetic background traffic mimicking the behavior of 14 HTTP clients (two per emulated wireless station), based on the stochastic model from Pries at al. [159].

The video sequence that we used was Big Buck Bunny [17]. We encoded the raw video data in 6 representations, distributing the MMBR's of the individual representation logarithmically between 100 kbps and 5 Mbps. We set the GOP size to 2 seconds, which is the maximum value allowed for Microsoft SmoothStreaming (MSS). The encoded data was split into segments of two seconds duration, and two manifest files were generated, one for the MPEG-DASH client, and one for MSS. We assured that the differences in segment sizes between the two alternative stream versions was small enough

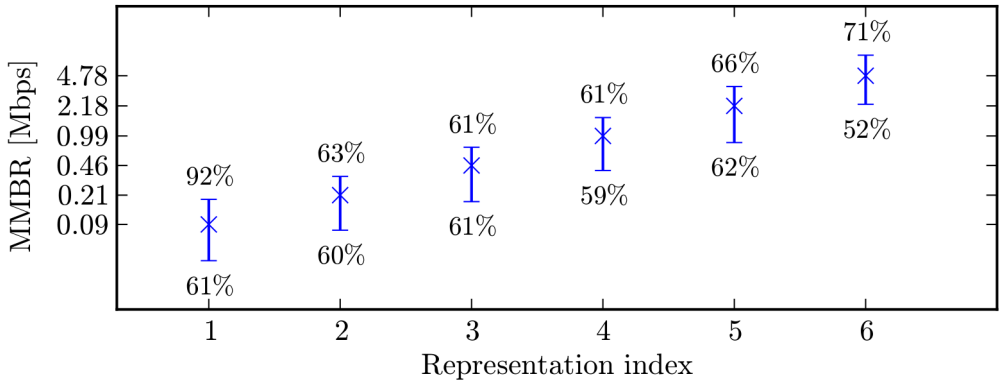

Figure 7.2: MMBR variation across segments: mean, minimum, maximum (the latter two are shown as percentage of the mean). (Note the logarithmic scale of the y-axis.)

to be negligible. It constituted on average 0.17%, while the maximum was 1.34%. The distribution of segment bit rates (which are proportional to the segment sizes in bytes) is illustrated in Figure 7.2. The complete video sequence is slightly longer than 596 seconds so we obtained 298 full segments and one short segment, which was omitted from the setting.

Note that media bit rate fluctuations across segments of the same representation may significantly affect the performance of a streaming client. Depending on the format of the manifest file, the client might not know the actual segment media bit rate (and thus its size in bits) in advance, but has to base its decisions upon the MMBR of the representation only. Due to VBR encoding, however, the segment media bit rate may easily fluctuate by up to a factor of 10 and more. Our dataset was encoded such as to keep these fluctuations small. To be precise, they are bounded by 95% of representation's mean values, as shown in Figure 7.2.

In order to have a fair comparison, we padded the MPEG-DASH manifest file with random characters to make it of the same size as the joint size of the three files that must be downloaded by the MSS client: the Hypertext Markup Language (HTML) file, the Silverlight Application Package (XAP) file and the manifest file.

In addition to comparing the performance of the two studied streaming clients with each other, we compared them to the optimal performance, computed using the approach presented in Chapter 8. Here, we proceeded as follows. Each experiment with a video client was followed by an experiment under the same conditions, where the video client was replaced by a TCP flow lasting for the duration of the video sequence. The throughput process of the TCP flow was then used as input $V(t)$ for the optimization problems OP1 and OP2, presented in Chapter 8, to calculate optimal adaptation trajectories. We did not use the throughput process as recorded by the video client since it may be sub-optimal due to the fact that streaming clients typically introduce inter-request delays in order to prevent their buffer level from exceeding certain limits.

We conducted two sets of experiments. In the first set, one streaming client shares

the wireless link with 14 other clients, generating HTTP traffic. In the second set, in order to analyze fairness aspects, two streaming clients are studied in the otherwise equal setting.

All experiments were repeated approximately 50 times. Confidence intervals in some of the figures are omitted to improve the readability.

7.4.1.2 Single Client

In this set of experiments, we let one streaming client share the wireless link with HTTP background traffic. We compared the performance of the proposed algorithm using different configurations against the performance of the MSS client, and with the performance of the omniscient client.

In all TOBASCO configurations, the minimum buffer threshold was set to $\beta_{\text{min}} = 2\,s$. The lower threshold β_{low} of the target buffer interval was varied between $5\,s$ and $20\,s$. The size B of the target buffer interval was varied between $5\,s$ and $20\,s$. Further, we varied the time period Δ_t for averaging the past throughput from $5\,s$ to $10\,s$. Other parameters were fixed to the following values: $\alpha_1 = 0.75$, $\alpha_2 = 0.8$, $\alpha_2 = 0.8$, $\alpha_2 = 0.8$, $\alpha_2 = 0.9$.

The results are presented in Figure 7.3. The figure shows the MMBR, the number of quality changes, the total rebuffering time, and the mean buffer level for the individual clients, averaged over multiple instances of the experiment. We observe that TOBASCO always outperforms the MSS client w.r.t. the MMBR (78% to 90% of the optimum vs. 62%). It also outperforms the MSS client w.r.t. the mean per-client rebuffering duration, and, for some of the configurations, w.r.t. the number of quality changes. The evaluation suggests that for the given setting, good values for the configuration of TOBASCO are: $\beta_{\text{min}} = 10s$, $B = 20s$, and $\Delta_t = 5s$.

Note that the optimal trajectory is able to almost perfectly utilize its fair share of the link capacity. Its MMBR almost equals the achievable TCP throughput on that link. Also note that the average number of quality transitions required by the optimal trajectory is as low as approximately 2 (and, in fact, might be even lower since the presented values are upper bounds to the optimum, as described in more details in Section 8.4).

Another interesting observation is that TOBASCO is able to achieve its performance with a lower average buffer level than the MSS client. For example, for the configuration $\beta_{\text{low}} = 5\,s$, $B = 20\,s$, $\Delta_t = 10\,s$, the DASH client achieves a high MMBR, a low rebuffering time and number of quality changes, with an average buffer level of approximately 17 seconds.

By analyzing the individual adaptation trajectories produced by TOBASCO we observe that some rebuffering events are caused by segments whose size significantly exceeds the average segment size of the representation. Since the segment size of the individual segments is not known to the client before it issues the request, it might too late notice that it is not possible to download the segment before its playback deadline. A possible solution for this issue would be to retrieve the segment sizes, taking into account a certain communication overhead, via HTTP HEAD requests prior to starting the streaming session or in parallel to it, or to use an MPD format containing information about the

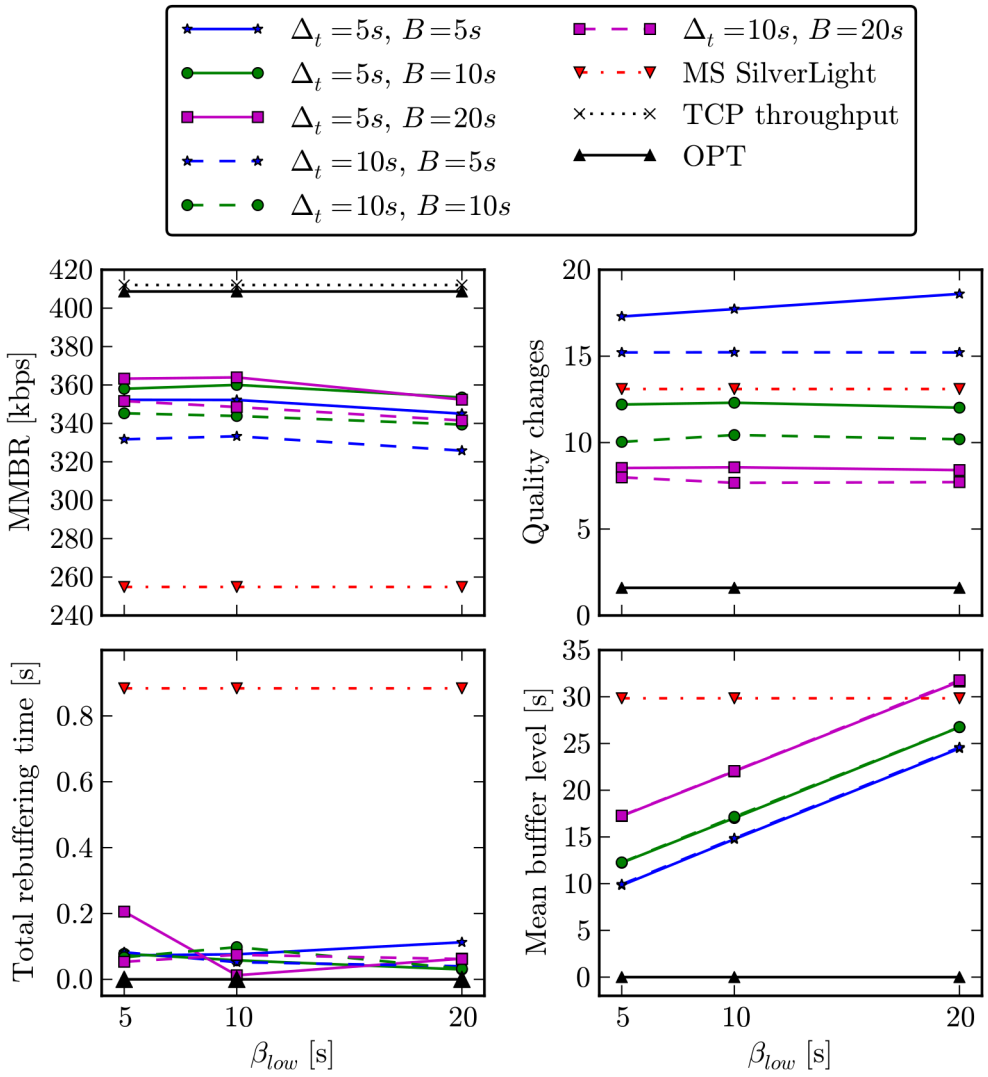

Figure 7.3: Performance of a single streaming client sharing the wireless link with HTTP background traffic.

individual segment sizes.

7.4.1.3 Multiple Clients

In this experiment, we added a second streaming client to the setting used in the set of experiments described in Section 7.4.1.2, in order to evaluate the adaptation strategies w.r.t. fairness. The traffic of the two clients was routed over the same wireless station.

Figure 7.4 shows the MMBR, the number of quality changes, the total rebuffering time, and the mean buffer level. Shown values are averages over the two clients and over multiple instances of each experiment. We observe that the conclusions from the setting

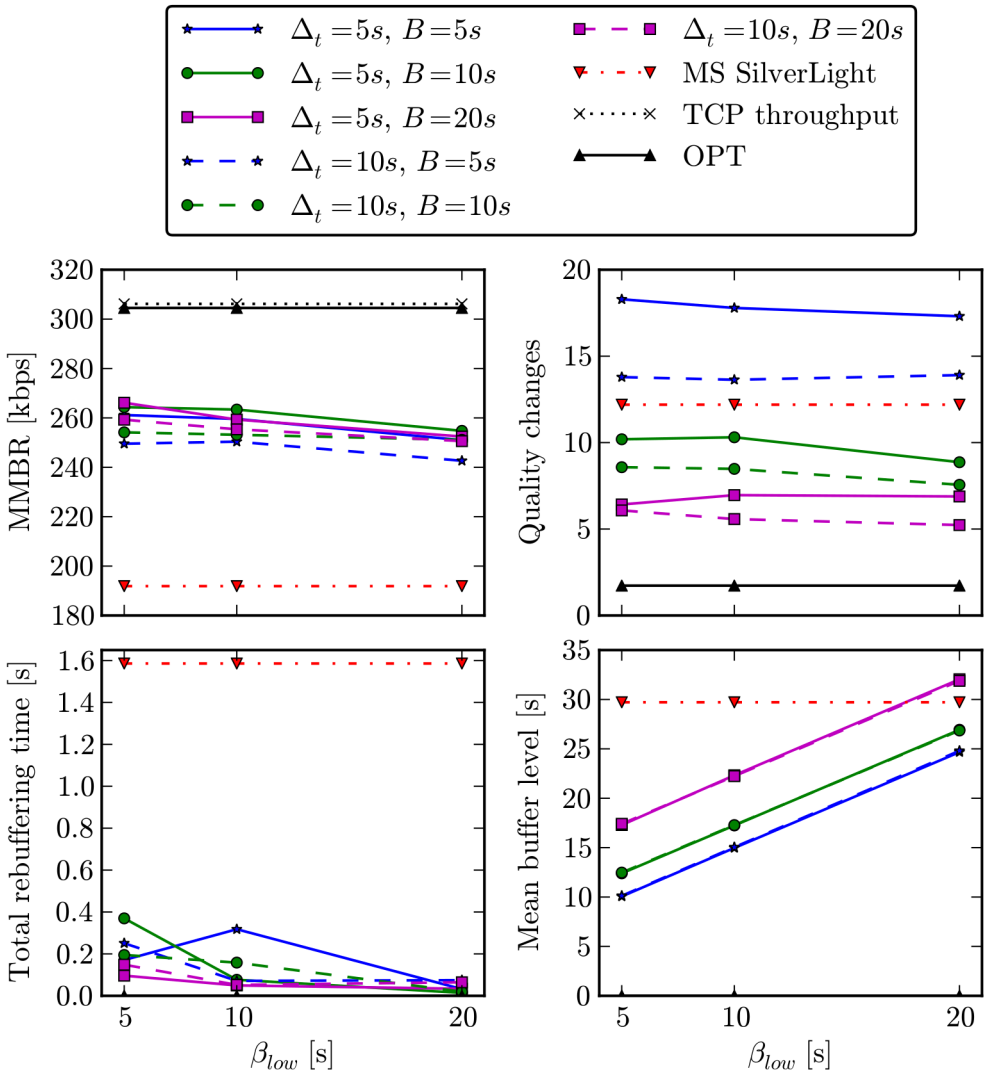

Figure 7.4: Performance of two video clients sharing the wireless link with HTTP background traffic.

with a single client still hold when two clients share the wireless bottleneck link.

Figure 7.5 shows the difference among the two clients averaged over the individual runs. We observe that optimal trajectories have almost perfect fairness since all values are close to 0. Further, we observe that the fairness of the DASH client is comparable or better than that of the MSS client.

7.4.2 Evaluation Using Real-World Measurements

A second evaluation campaign was performed using several artificial and real-world scenarios. On the one hand, we performed experiments streaming over the local loop with-

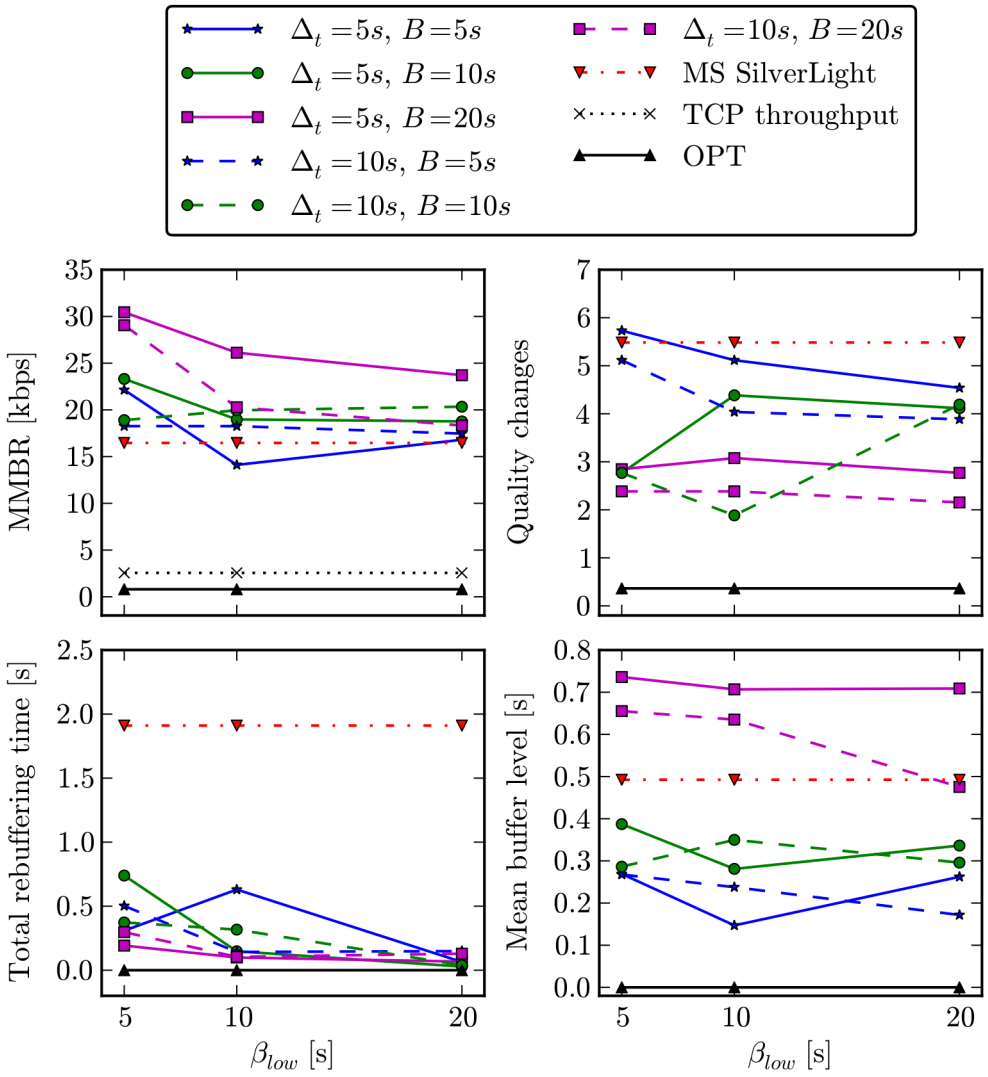

Figure 7.5: Fairness among two video clients sharing the wireless link with HTTP background traffic.

out cross-traffic, artificially limiting the throughput using the open source tool DummyNet [22]. On the other hand, we performed experiments in a busy domestic WiFi with a high level of interference and heavy cross-traffic. We observed that the algorithm performs remarkably well even under extremely challenging network conditions. Further, it exhibits a stable and fair behavior when two clients share a common network path.

In all experiments we used the following values for the configuration parameters. We set the minimum buffer level, below which we immediately switch to the lowest representation, to $B_{\min} = 10$ s. We set the target interval to $[20, 50]$ s. The discretization parameter for the buffer level was set to $\Delta_\beta = 1$ s. The available throughput was measured and averaged over the past $\Delta_t = 10$ s. The safety margins were set to

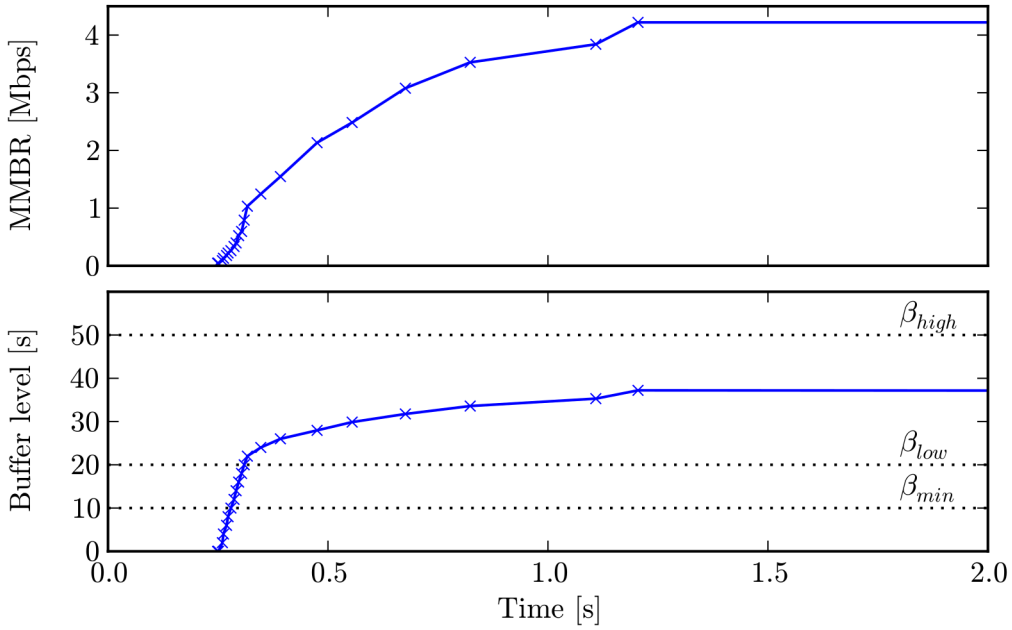

Figure 7.6: Single client, unrestricted throughput

$(\alpha_1, \ldots, \alpha_5) = (0.75, 0.33, 0.5, 0.75, 0.9)$.

As the video sequence we used the computer-animated movie Big Buck Bunny [17] that was formatted according to the MPEG-DASH specification using the open source MPEG-DASH content generation tool DASHEncoder [120] and made available at the web site of University of Klagenfurt, Austria. The sequence has a duration of approximately 600 seconds. The segment size is $\tau = 2$ s. The available mean media bit rates are 45, 89, 131, 178, 221, 263, 334, 396, 522, 595, 791, 1033, 1245, 1547, 2134, 2484, 3079, 3527, 3840, and 4220 kbps.

Figure 7.6 shows the convergence properties of the algorithm on a link with "unlimited" bandwidth (local loop). The top subfigure shows the selected MMBR, the bottom subfigure shows the dynamics of the buffer level $\beta(t)$. The playback started approximately 250 ms after the begin of the download. After 1.25 s, the algorithm started to download segments from the highest available representation.

In order to avoid abrupt quality changes, the proposed algorithm change representation one level at a time. In configurations with a large number of representations, as the presented one (20 representations), convergence speed could be further improved by allowing the algorithm to skip representations while adapting quality. In real deployments, however, the number of available representations will typically be lower due to storage costs on the one hand, but also because performance improvements from additional representations become smaller with a growing number of representations (see also the results in Section 8.4).

Figure 7.7 shows the reaction of the algorithm to persistent throughput changes. The top subfigure shows the artificial throughput limitation, the measured segment

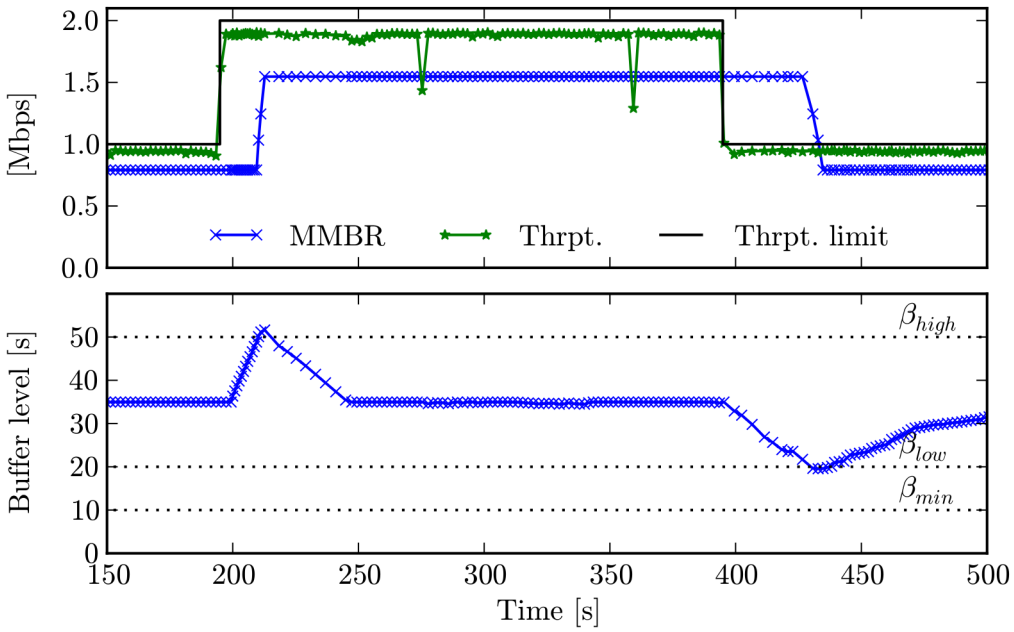

Figure 7.7: Single client, persistent throughput changes (200 s).

throughput, and the selected MMBR. The bottom subfigure shows the dynamics of the buffer level. After the initial phase, where the throughput was limited to 1000 kbps, the throughput limitation was shifted to 2000 kbps for 200 seconds, and then back to 1000 kbps. The algorithm reacted by adapting the MMBR of the stream after a moderate delay.

Figure 7.8 shows the results of a run, where the client was exposed to periodic short-term throughput spikes lasting for 5 s each. The desired behavior here is not to follow the individual spikes but rather to stay with a representation that is sustainable with the available network resources. This is exactly the behavior exhibited by the algorithm.

Figure 7.9 shows an experiment in a shared indoor IEEE 802.11bg WLAN, in a domestic environment, with a high level of interference and heavy cross-traffic. We observe that the algorithm is able to follow the dynamics of the network throughput in a robust manner.

The last two figures, 7.10 and 7.11, show an artificial and a real-world scenario where two players share a common link. In the first scenario, the test is performed over the local loop with total throughput restricted to 2000 kbps. In the second scenario, the clients stream over a domestic WiFi, as described above. In both scenarios, we observe that the clients are able to share the available bandwidth in a stable and fair manner.

Finally, we remark that in all performed runs no buffer underruns occured.

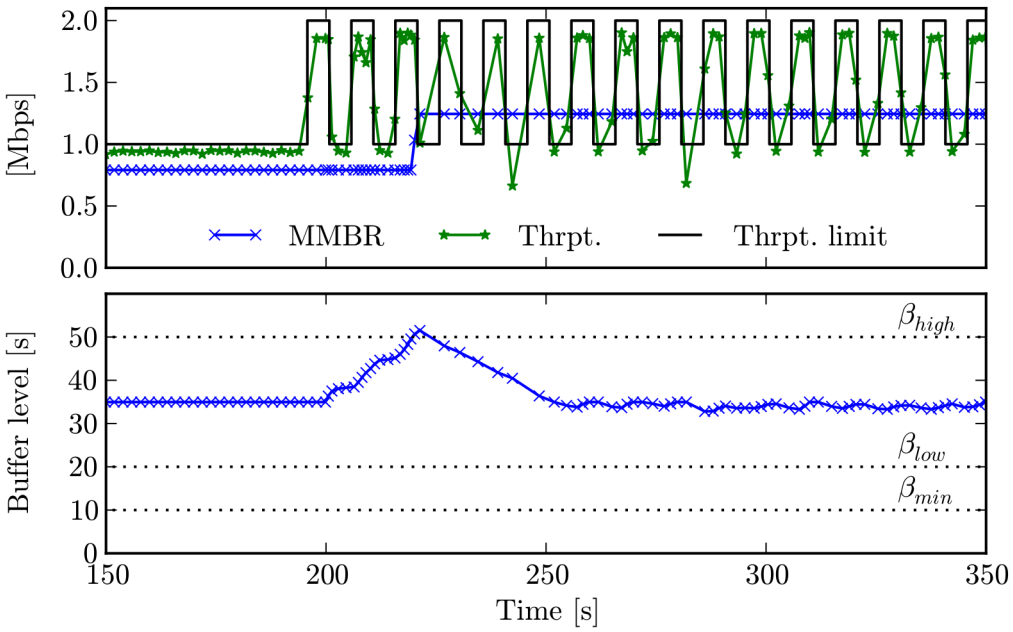

Figure 7.8: Single client, periodic throughput fluctuations (periodicity: 5 s).

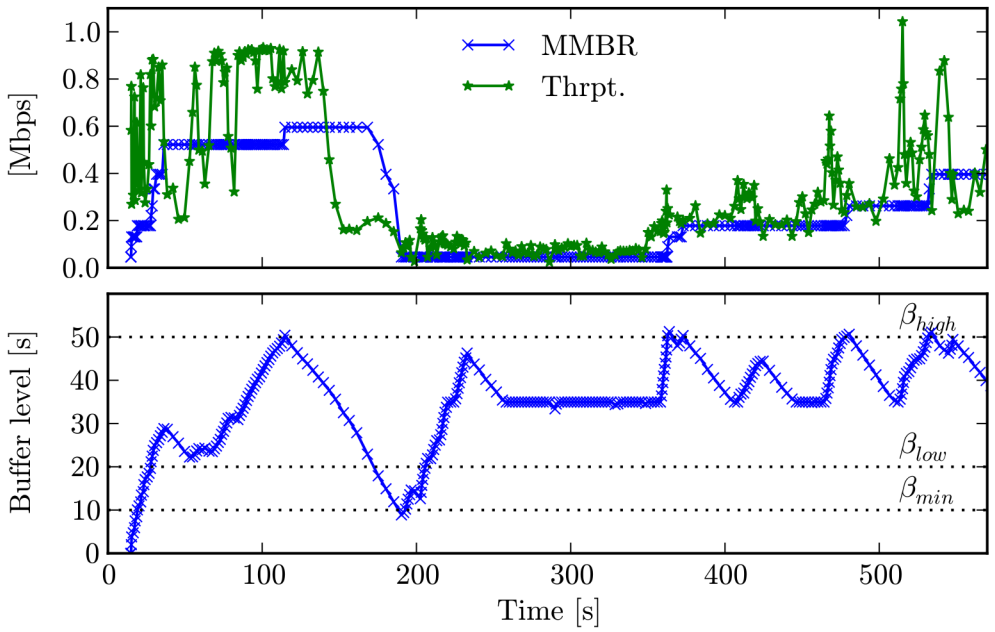

Figure 7.9: Single client, shared indoor WiFi in residential area (high interference, heavy cross-traffic).

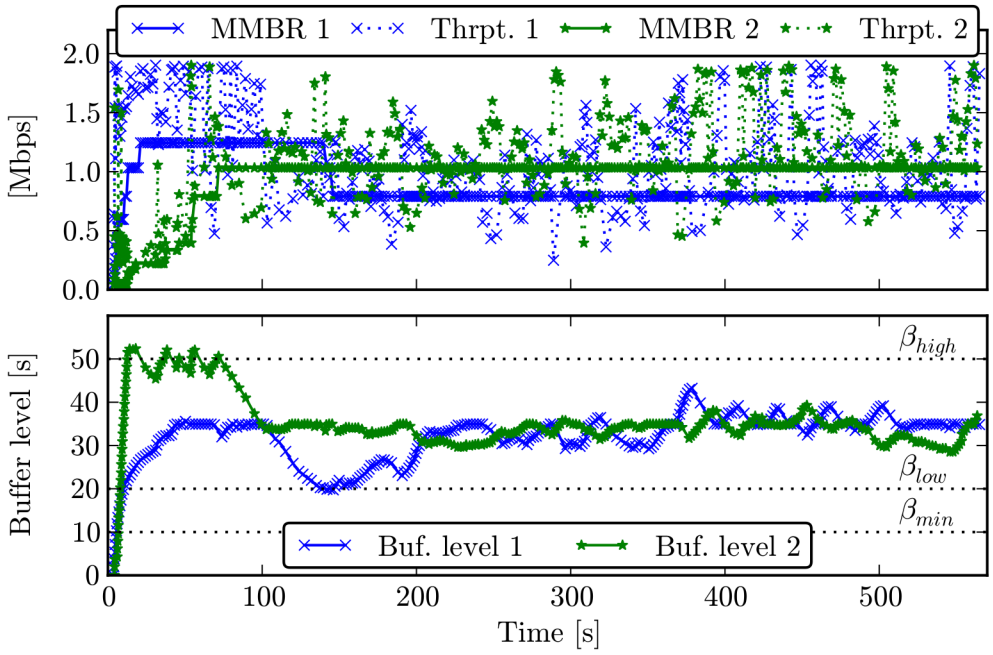

Figure 7.10: Concurrent clients, total throughput restricted to 2 Mbps.

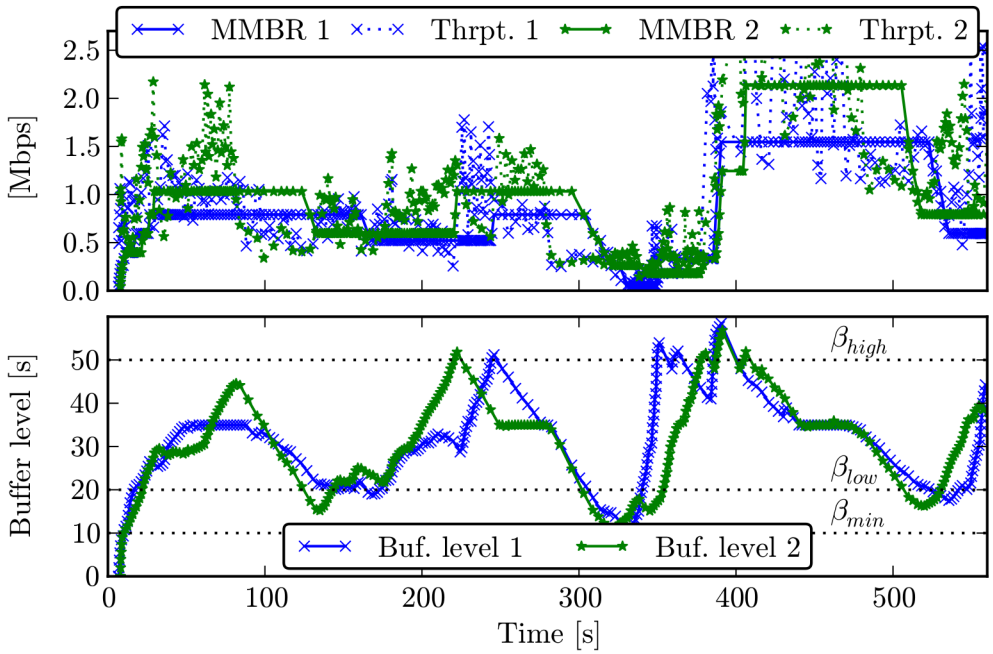

Figure 7.11: Concurrent clients, shared indoor WiFi in residential area (high interference, heavy cross-traffic).

Chapter 8

Optimal Adaptation by an Omniscient Client

Performance evaluation is an art [95]. In the case of adaptive streaming, there exist several specific challenges. One of them, discussed in Section 2.2, is the lack of an established QoE metric. Another one is the dependency of the performance on the particular network environment. Meanwhile, there exists a large body of literature on adaptation algorithms for HTTP-based video streaming. Unfortunately, most of the results of the individual studies are hardly comparable. In this chapter, we propose an approach to computing optimal adaptation trajectories that can be used as a benchmark for the evaluation of adaptation approaches for VoD.

8.1 Introduction

One of the open issues in the area of adaptive video streaming is the methodology for performance evaluation of adaptation algorithms. In a typical generic performance evaluation study, we either compare the performance to some predefined requirements, or we measure its gain against state-of-the-art solutions, or we compare its performance to the optimum performance that can be achieved under given conditions.

To the best of our knowledge, there currently exist no widely accepted benchmarks to measure the performance of adaptive streaming clients in best-effort networks. Neither is any of the existing clients widely accepted as a basis for performance comparison. Finally, little has been done on developing approaches to calculating optimal performance of a HAS client under given network conditions. Such an approach, however, would not only allow to benchmark the performance of any streaming client, it would also allow to study

the impact of the networking environment and of configuration parameters such as the maximum allowed start-up delay, number of available video representations, etc., on the streaming performance.

Consequently, in the following, we propose an approach to computing optimal adaptation trajectories, given the complete information on the throughput process (that is, the amount of data that can be downloaded until time t, for each t). That is, we adopt the perspective of an omniscient client.

This approach can be used in the following three ways. First, we can calculate an optimal trajectory for a throughput process that was recorded by a streaming client during a streaming session. The optimal trajectory can then be compared with the client's actual trajectory to quantify the performance. In many cases, however, streaming clients introduce delays between subsequent requests so that the recorded throughput process itself is already sub-optimal, such that it no longer can be used to compute an optimal adaptation trajectory. Thus, instead of using a trace recorded by a streaming client, we shall use a trace recorded by a continuous TCP flow under comparable network conditions. Finally, optimal trajectories can be calculated for artificial throughput processes in order to study the impact their features have on the optimal performance.

8.2 Optimization Objectives

As discussed in details in Section 2.2, the following factors have the strongest influence on the QoE for HAS in the case of VoD:

- the duration and distribution of playback interruptions,
- the distortion of the video (video quality),
- the frequency of media bit rate transitions,
- the start-up delay.

To the best of our knowledge, there exist no widely accepted or standard QoE metric, which quantifies the human perception of all these factors together. Consequently, we use the following objectives and constraints for the optimization of adaptation trajectories.

We use the start-up delay as an independent variable, that is, our approach allows to compute an optimal trajectory for a predefined start-up delay. Further, we impose a hard constraint on the absence of buffer underruns, reflecting the common understanding that such underruns have the highest impact on the QoE. As for the optimization objective, we first maximize the MMBR over the duration of the streaming session. This maximization in general results in a space of optimal solutions that are potentially prone to frequent video quality transitions. Therefore, we subsequently minimize the number of quality switches.

8.3 Computation of Optimal Adaptation Trajectories

The main challenge in designing efficient adaptation strategies for adaptive video streaming is that the throughput on a path in a best-effort network, such as the Internet, is a

$V(t)$	Data volume received during the time interval $[0, t]$
$S_{\mathcal{R}}$	Vector of all segment sizes, defined in (8.1)
S_{MPD}	Size of the MPD file
$\mathcal{S}$	Tuple of all media parameters relevant for the optimization
T_E	Time of the earliest possible start of the playback
T_S	Time of the actual start of the playback
$\tilde{T}_S = T_S - T_E$	Start-up delay component, which is controlled by the client
$x_{ij} \in \{0, 1\}$	Optimization variable indicating if segment i is downloaded in representation j

Table 8.1: Notation extensions for the omniscient client

random process. It is thus extremely challenging to reliably predict it for the relevant time horizon, typically ranging from several seconds to several tens of seconds for VoD services. In the following, we use the term *throughput process* to denote the total amount of data $V(t)$ in bytes received by a client during the time $[0, t]$.) The question that we address in this section is how to calculate an optimal adaptation trajectory, given the complete knowledge of the future throughput $V(t)$, $t \in [0, T]$.

In the following, we will adopt the notation introduced in Chapter 4. Recall that $\mathcal{R}$ denotes the set of available representations, n denotes the number of segments, τ denotes the duration of video content contained in a segment (w.l.o.g. we assume that the segment duration is constant across all segments), and s_{ij} denotes the size in bits of segment i from representation j. We extend this notation by the elements described in the following, summarized in Table 8.1. We write

$$S_{\mathcal{R}} = (s_{ij},\, i \in \{0, \ldots, n-1\},\, j \in \{0, \ldots, m-1\}) \,, \tag{8.1}$$

with $m = |\mathcal{R}|$, for the vector containing the segment sizes of all available segments. We denote by S_{MPD} the size of the MPD file in bytes. We associate with the content to be streamed the tuple $\mathcal{S} = (S_{\text{MPD}}, \mathcal{R}, n, \tau, S_{\mathcal{R}})$.

Obviously, the earliest time when the playback can start is when the MPD file and the first segment of the representation with the lowest MMBR are downloaded. We denote this time by T_E. We remark, however, that the computed optimal adaptaiton trajectories do not always require the client to download the first segment in the lowest representation. Given a throughput process $V(t)$ and a video configuration $\mathcal{S}$, T_E is always well-defined. Depending on the content format, an initialization segment might be required to be downloaded upfront in order to initialize the decoder. In these cases, T_E denotes the time when the MPD file, the initialization segment, and the first segment of the lowest representation are downloaded. For simplicity, we do not account for the initialization segment in the following formalization and refer by $i = 0$ to the first segment of a representation.

We define T_S to be the time span between the start of the download ($t = 0$) and the beginning of the playback. Obviously, we demand $T_S \geq T_E$. The important value for the optimization, however, is not T_S but the time between the earliest possible time when the playback can start and the actual start of playback, denoted by $\tilde{T}_S = T_S - T_E$, which we define as the start-up delay. The reason is that the time T_E cannot be influenced by the client, while $\tilde{T}_S$ depends on the client's adaptation strategy.

The resulting playback deadlines for the individual segments are given by $t_i^p = T_S + i\tau$. The maximum amount of data a video player can download until the playback deadline of segment i is thus given by $V(t_i^p)$.

In order to formulate the optimization problem, we denote by $x_{ij} \in \{0, 1\}$ the optimization variables stating if the client downloads segment i from representation j or not.

In the following, we first maximize the MMBR of the streaming session (which is equivalent to the maximization of the total amount of data downloaded by the video client), demanding the absence of playback interruptions. We obtain the following optimization problem.

$$\textbf{(OP1)} \quad \max \quad \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} s_{ij} x_{ij} \tag{8.2}$$

$$\text{s.t.} \quad \sum_{j=0}^{m-1} x_{ij} \geq 1 \qquad \text{for all } i = 0, \ldots, n-1 \tag{8.3}$$

$$\sum_{i=0}^{k} \sum_{j=0}^{m-1} s_{ij} x_{ij} \leq V(t_k^p) \qquad \text{for all } k = 0, \ldots, n-1\,. \tag{8.4}$$

Here, constraint (8.3) ensures that each segment is downloaded from at least one representation, while constraint (8.4) ensures that each segment is downloaded before its playback deadline. Note that constraint (8.4) implicitly accounts for the configured start-up delay (included in the definition of the playback deadlines t_i^p).

Problem OP1 has a set of optimal solutions that are more or less prone to video quality fluctuations. Unnecessary quality switches, however, negatively impact the QoE. Therefore, we subsequently solve the following optimization problem OP2 in oder to select the optimum solution of OP1 that has the minimum number of quality switches.

We denote by V^* the optimal objective value of the problem OP1.

$$\textbf{(OP2)} \quad \min \quad \frac{1}{2}\sum_{i=0}^{n-2}\sum_{j=0}^{m-1}(x_{ij}-x_{i+1,j})^2 \tag{8.5}$$

$$\text{s.t.} \quad \sum_{j=0}^{m-1} x_{ij} \geq 1 \qquad \text{for all } i=0,\ldots,n-1 \tag{8.6}$$

$$\sum_{i=0}^{k}\sum_{j=0}^{m-1} s_{ij}x_{ij} \leq V(t_k^p) \qquad \text{for all } k=0,\ldots,n-1 \tag{8.7}$$

$$\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} s_{ij}x_{ij} \geq V^*\,. \tag{8.8}$$

While constraints (8.6) and (8.7) replicate the constraints (8.3) and (8.4) in OP1, constraint (8.8) ensures that the minimization of the new objective function (8.5) (the total number of quality switches) does not result in a sub-optimal value for the MMBR.

The problem OP1 is known as a Multiple-Choice Nested Knapsack Problem (MCNKP) [9, 130]. More precisely, it is a special case, where the values of the items and the weights of the items are equal. (This variant of the Knapsack Problem is sometimes referred to as the Subset Sum Problem.) MCNKP is NP-hard but there exist pseudo-polynomial time algorithms. Problem OP2 is a Quadratic MCNKP. We use the optimization software Gurobi [68], to solve both problems.

8.4 Influence of the Number of Representations

In Chapter 7, we have shown how the optimal trajectories computed by the omniscient client can be used to evaluate the performance of an adaptation algorithm using the example of TOBASCO. Another example for the use of the optimal trajectories is evaluating the influence of the media encoding parameters, such as the number of representations, and of the adaptation parameters, such as the start-up delay, on the QoE.

While a large number of representations gives the client a greater flexibility in adapting to the network conditions, it makes the adaptation more complex, and results in higher storage costs for the content provider. Several studies perform performance evaluations using different numbers of representations, coming to a conclusion that the influence on different algorithms is strongly varying. Some algorithms even exhibit worse performance if confronted with large representation sets.

Figure 8.1 shows the influence of the number of representations, and of the configured start-up delay on the optimal performance in terms of the mean media bit rate, and in terms of the mean number of quality transitions. The evaluation setting is as described in Section 7.4.1.

We observe an initially perfectly linear dependency of the average video bit-rate on the start-up delay, followed by a saturation. Maybe somewhat surprisingly, the improvement from a start-up delay of 60 sec is only around 12%. Further, we observe no

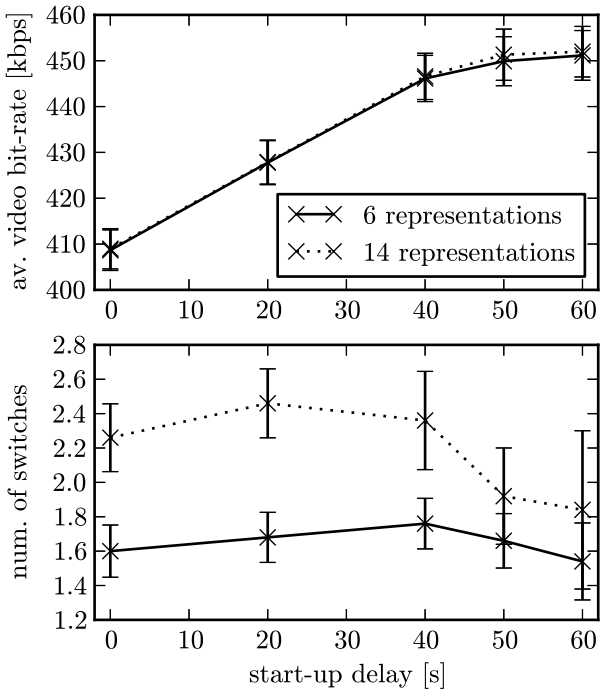

Figure 8.1: Influence of the number of representations, and of the start-up delay on the optimal adaptation trajectories. Vertical bars show the confidence intervals for the confidence level of 95%.

improvement from increasing the number of the used representations from 6 to 14. The reason is that despite of the high fluctuation of the wireless link throughput (fading plus cross-traffic), 6 representations are enough to utilize all the available bandwidth. We remark that in real deployment scenarios, a high number of available video representations might be quite unrealistic, since it results in higher encoding and storage costs for content providers.

In order to speed-up calculations when solving the optimization problem OP2, we allowed an absolute optimality tolerance gap of up to 5 (that is, the actual optimum might be even lower than the calculated value by at most 5). The average value for the gap during the evaluation added up to 1.83. For 6 representations it was 1.51, for 14 representations it was 2.15, due to the higher computational complexity. Thus, the higher amount of quality switches for 14 representations might partially be explained by the inexact computations. An important insight, however, is the remarkably small amount of switches required to achieve optimal performance despite of a highly fluctuating network throughput.

Chapter 9

Universal Streaming Client Architecture

In 2011, when the work on this thesis has been started, the HAS paradigm was just about to start into its success story. Consequently, there was a lack of reference implementations that could have been used as an evaluation platform or, at least, as a starting point for own developments. Thus, we developed own implementations that we integrated into multimedia frameworks such as GStreamer [66], VLC [206], or dash.js [45]. In addition, in order to perform performance evaluation on a large scale, some of the approaches have been implemented as simulation modules. During the course of the implementation works, a client architecture has emerged that has proved itself efficient and flexible in fitting our needs. This architecture is presented in the following.

9.1 Introduction

A video streaming client is a complex system consisting of several modules providing quite different functionality. Video data has to be downloaded, demultiplexed, decoded, and presented to the user. Prior to starting a streaming session, a HAS client has to download and parse the MPD file, which typically has XML format. In order to efficiently perform adaptation decisions, it has to monitor the network conditions dynamics, potentially including cross-layer information, e.g., internal state variables from the transport and/or MAC layers of the protocol stack. It might also make use of context information, such as, e.g., situational properties of the user environment (indoor/outdoor, mobility pattern, level of distraction). Finally, it shall support a distributed operation of individual components, in order to support cloud-assisted [211], or network-assisted [52]

operation.

The framework embracing the individual modules must be designed such that the functional blocks can be easily replaced by other ones to facilitate the evolution of the system as well as testing experimental approaches. In particular, the APIs must be well-defined and powerful enough to transport all the necessary information, while flexible enough to enable the transfer of individual functional blocks to other environments, e.g., network simulator frameworks.

Moreover, in order to reduce costs and complexity involved in the development and maintenance of multiple software packages, a streaming client is typically required to perform efficiently on a variety of platforms, from web browser environments, over Personal Computers (PC's) and mobile terminal operating systems (smartphones, tablets, wearables, etc.), to embedded devices such as TV sets and STB's.

During the course of the works presented in this thesis, several adaptation strategies were implemented, as simulation models, as prototypes, or both. During this process, the following requirements to the software design have emerged.

- The design must be suitable for low-delay live streaming and VoD.
- Modularization allowing for flexible deployment of different adaptation algorithms.
- Distributed operation of individual functional blocks.
- Support for cross-layer information exchange.
- Support for multi-interface operation.

The proposed architecture satisfying these requirements is proposed in the following.

9.2 Architecture

It turned out that the most convenient way to satisfy the identified requirements was to have an event-driven design with a controller module that implements a state machine with a predefined finite set of feasible events at each state, and with a set of corresponding actions that are executed when an event occurs in a certain state. In addition, individual modules can dynamically register own event/actions pairs, if necessary, or override the existing ones. Figure 9.1 depicts the functional blocks of the proposed architecture.

Note that the controller here is not related to the PID controller in Chapter 5. Rather, the controller presented here has the goal to coordinate the operation of the individual functional blocks of the streaming client. All the adaptation algorithms presented in the previous chapters reside in the Adaptation Engine, except for the transmission scheduling component in Chapter 5, which resides outside of the client at the central network controller.

We slightly abuse the notation by using the letter e with a subscript to denote an event, and the capital letter S with a subscript to denote the state of the client state machine. Both letters have been used in the preceding chapters but in a different context, so that a confusion can hopefully be avoided.

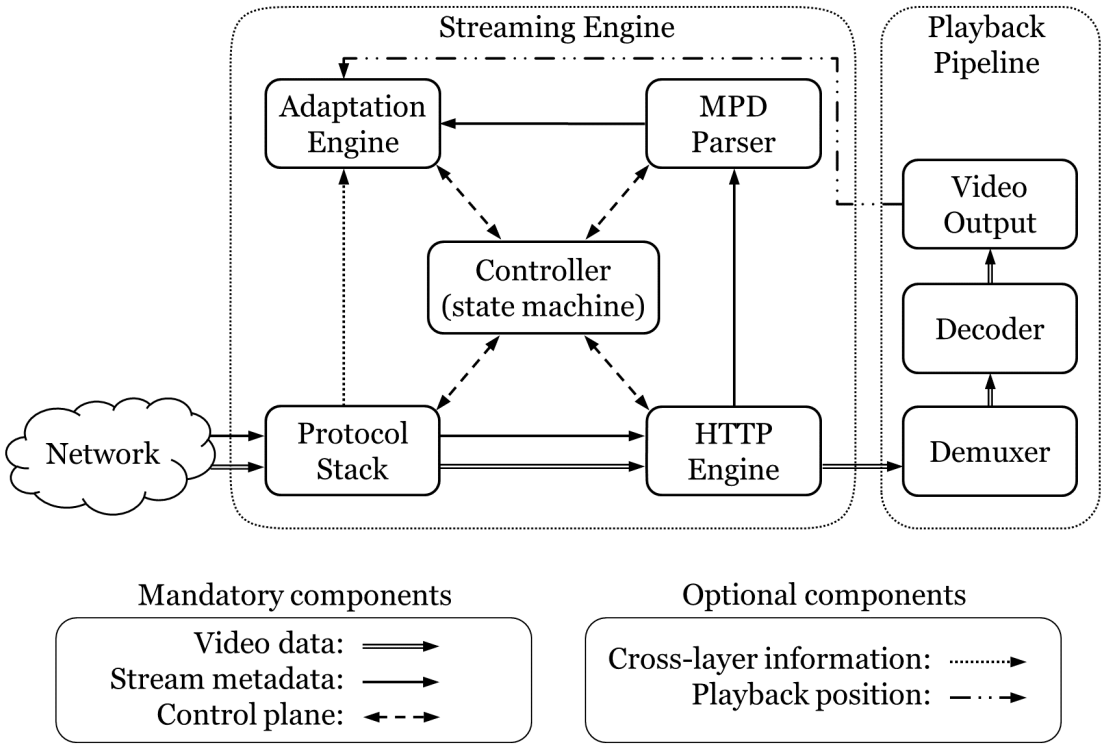

Figure 9.1: Functional blocks of the proposed streaming client architecture.

In the following, we present two example state machines: one for low-delay live streaming, and one for VoD. Note that the presented sets of states and sets of events are not meant to be complete, but should rather illustrate the usage of the proposed design. For example, they do not include the functionality required to download and parse the MPD. Further, we omit the cases of interrupted segment downloads (e.g., due to a connectivity problem), or cancelled segment downloads (e.g., due to an anticipated deadline miss by the segment being downloaded).

Also note that we abstain from specifying the details of the actions executed when a particular event occurs in a particular state since at least some of them depend on the particular implementation or adaptation approach. For example, if a live streaming client observes the event $e_{p:u}$ (playback deadline reached for a segment whose download has not been completed yet), one possible approach is to wait until the segment download is completed, increasing the live latency, while another approach is to cancel the download and continue with one of the subsequent segments in order to stick to a maximum delay bound. (The latter approach is the one we have chosen for our low-latency adaptation algorithm LOLYPOP presented in Chapter 6.)

Finally, we would like to remark that our focus was on developing and evaluating adaptation algorithm, rather than having a client supporting the numerous features of the MPEG-DASH standard, or having all the functionality required by a full-fledged video player. However, we believe that the presented design can successfully be applied

also to these cases.

9.3 State Machine for a Live Streaming Client

Table 9.1 lists and describes the states of the live streaming client. Functionality such as downloading, parsing, and updating the MPD is omitted to improve the readability of the state diagram. Note that the waiting state S_W is entered when the client completes the download of a segment but can neither proceed with the download since the next segment is not available yet, nor can it start the playback since the playback deadline of the segment has not been reached. This situation can occur when tuning into a live stream, as described in Section 6.3.2.

State	Description
S_0	initial state
S_D	downloading only (not playing)
S_P	playing only (not downloading)
S_{DP}	downloading and playing
S_W	waiting (idle, neither downloading nor playing)
S_T	terminal state

Table 9.1: States of a live streaming client

Table 9.2 lists and describes the events for the case of a live streaming client. In addition to the events representing the start and the termination of the streaming session, the set of events contains three categories: occurrence of a playback deadline, completion of a segment download, and start of the segment availability. A category may contain multiple events. For example, the event category 'playback deadline' contains one event that occurs when the playback deadline occurs while the playback buffer is empty, and another event that occurs while the subsequent segment has already been buffered. Another approach to account for these distinctions would be to define separate states, however, this would lead to a more complex state machine and a more complex implementation.

Figure 9.2 presents the state machine for the live streaming client. If a particular event is not listed for a particular state, it is infeasible in this state and results in undefined behavior.

Event	Description
e_0	start of the streaming session
$e_{p:b}$	playback deadline for a buffered segment
$e_{p:u}$	playback deadline for an unbuffered segment
$e_{d:a}$	segment download completed, next segment available
$e_{d:u}$	segment download completed, next segment unavailable
e_a	segment becomes available for download
e_T	end of playback

Table 9.2: Events of a live streaming client

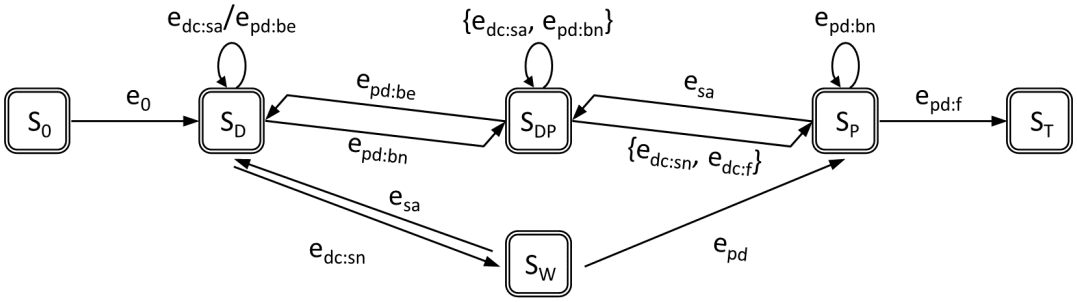

Figure 9.2: State diagram for the live streaming client

9.4 State Machine for a Video on Demand Streaming Client

Table 9.3 lists and describes the states of a VoD streaming client. The set of states is almost identical with that of the live streaming client, except that it lacks the waiting state S_W. While a VoD client may use inter-request delay to prevent the playback buffer level from exceeding a certain upper bound, it would never be completely idle. Note that a client is only idle if the buffer is empty so that the playback is interrupted, but at the same time it cannot start a new download. A potential reason for that can be that the next segment is not available yet. In a VoD streaming session, however, all segments are available for the download right from the start of the session.

Table 9.4 lists and describes the set of events for the case of a VoD streaming client. Here, the differences as compared to the live streaming client are more significant than with the set of possible states. Note that the evets $e_{p:b}$ and $e_{p:u}$ are no longer defined

State	Description
S_0	initial state, prior to starting the streaming session
S_D	downloading only (not playing)
S_P	playing only (not downloading)
S_{DP}	downloading and playing
S_T	terminal state, after the streaming session terminated

Table 9.3: States of a VoD streaming client

in terms of a playback deadline but in terms of finishing the playback of a previous segment.

Event	Description
e_0	start of the streaming session
$e_{p:b}$	playback completed, next segment is buffered
$e_{p:u}$	playback completed, next segment unbuffered
$e_{p:f}$	playback of the final segment completed
e_d	segment download completed, next segment available
$e_{d:irs}$	segment download completed, start of an inter-request delay
$e_{d:f}$	segment download completed, next segment is final
e_{irc}	inter-request delay completed
e_T	end of playback

Table 9.4: Events of a VoD streaming client

Figure 9.3 depicts an example state diagram for a VoD streaming client. Note that depending on the adaptation logic alternative state diagrams are possible. For example, if a segment download is completed in the downloading state S_D, the state diagram in Figure 9.3 suggests an immediate transition to the downloading and playing state S_{DP}. Alternatively, this transition might be replaced by an internal transition (not changing the state), while the transition to S_{DP} might be triggered by a separate event that corresponds to reaching a particular buffer level or a time deadline, where the specific buffer level or the deadline are determined by the adaptation logic.

Another example for an alternative implementation is the definition of the event e_{irc} that lets the client resume the playback by initiating a transition from the state representing an inter-request delay (playing, not downloading) to the playing/downloading state S_{DP}. As in the preceding example, this event may be defined as reaching a certain buffer level, or a time deadline, that may be determined by the adaptation logic. Related to the definition of this event is the assumption that the event $e_{p:u}$ cannot occur during an inter-request delay since that would mean that the download was paused even though the last segment in buffer was being played out.

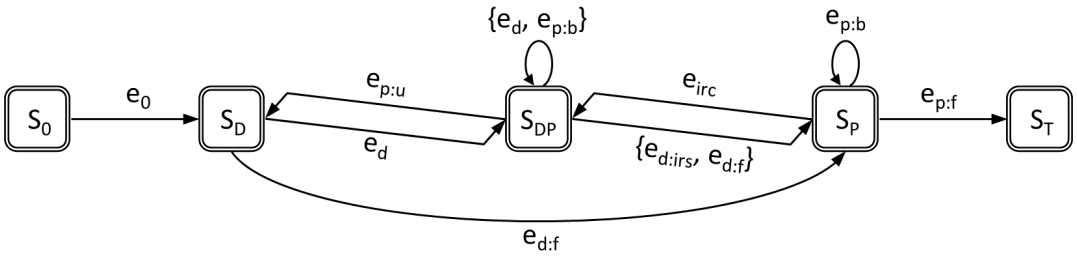

Figure 9.3: State diagram for the VoD streaming client

Chapter 10

Conclusions and Future Work

The video delivery landscape is currently an extremely fast evolving ecosystem. The classical linear television is being transformed, complemented, and partially replaced by new services that are by design interactive, individualized, and interconnected with the social platforms. One of the pillars required to support this evolution is an ubiquitous high-speed communication platform, such as the Internet. Due to its core design principles, however, the Internet does not provide any Quality of Service (QoS) guarantees, and thus, applications have to dynamically adapt their requirements to the available QoS level. In particular in wireless networks, that will soon be the dominating Internet access technology, the users, specifically if they are mobile, are exposed to continuous link quality fluctuations. Moreover, the time constraints in the case of live streaming, and the continuously increasing throughput requirements of the video services (High-Definition (HD), Ultra-High-Definition (UHD), Virtual Reality (VR), Augmented Reality (AR), etc.), further increase the challenge.

In the present thesis, we have approached the problem of maximizing the QoE of video streaming services by dynamically adapting the video characteristics to the network conditions. We have developed adaptation algorithms for three important and challenging deployment scenarios: multiple unicast sessions in a small cell wireless network, low-delay live streaming in wireless networks, and VoD in diverse network environments. All the developed adaptation algorithms have been implemented either as modules for network simulators, or as prototypes, or both. They have been evaluated in realistic network environments, and succeeded in improving the considered performance metrics as compared to the corresponding baseline approaches. In addition, we have

developed a series of optimization problems that compute optimal adaptation trajectories from the perspective of an omniscient client, and thus can serve as a benchmark for the performance evaluation of streaming clients. Finally, based on the insights gained while performing the various implementations, we have designed a flexible and efficient streaming client architecture.

We envision further potential in using big data techniques to complement and further optimize the developed adaptation approaches by analysing the performance characteristics of a large number of monitored streaming sessions, either offline or in real time. Another highly topical area of research covers the extension of adaptive streaming techniques to multi-view video, such as tele-immersion, VR, or Free-Viewpoint Television (FTV). Optimization techniques that allow to transmit only the content which is required to render the current user's perspective have the potential to greatly boost the QoE. Last but not least, developing techniques allowing to perform adaptation to the user's context, including the environment, level of activity, social context, and task context, will enable a truly individualized experience.

Appendix A

Acronyms

AIC	Akaike Information Criterion
ALM	Application Layer Multicast
AP	Access Point
API	Application Programming Interface
AR	Augmented Reality
ARIMA	Autoregressive Integrated Moving Average
BOWL	Berlin Open Wireless Lab
BS	Base Station
CAPEX	Capital Expenditure
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CIF	Context Influence Factor
CPU	Central Processing Unit
CSI	Channel State Information
CSS	Cubic Smoothing Splines
CV	Coefficient of Variation

DASH	Dynamic Adaptive Streaming over HTTP
DES	Double Exponential Smoothing
DSL	Digital Subscriber Line
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disc
ECDF	Empirical Cumulative Distribution Function
eMBMS	evolved Multimedia Broadcast Multicast Service
EWMA	Exponentially Weighted Moving Average
FDMA	Frequency Division Multiple Access
FR	Full Reference
FTP	File Transfer Protocol
FTV	Free-Viewpoint Television
GOP	Group of Pictures
HAS	HTTP-Based Adaptive Streaming
HD	High-Definition
HIF	Human Influence Factor
HLS	Apple HTTP Live Streaming
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HVS	Human Visual System
IETF	Internet Engineering Task Force
IIS	Internet Information Services
IP	Internet Protocol
IPTV	Internet Protocol Television
ITU	International Telecommunication Union
JND	Just Noticeable Difference
LOESS	Locally Weighted Scatterplot Smoothing
LOS	Line-of-Sight

LTE	Long-Term Evolution
MAC	Media Access Control
MCNKP	Multiple-Choice Nested Knapsack Problem
MCS	Modulation and Coding Scheme
MIMO	Multiple Input Multiple Output
MLE	Maximum Likelihood Estimation
MMBR	Mean Media Bit Rate
MOS	Mean Opinion Score
MPC	Model Predictive Control
MPD	Media Presentation Description
MPEG	Moving Picture Expert Group
MSE	Mean Squared Error
MSS	Microsoft SmoothStreaming
NAT	Network Address Translation
NLOS	Non-Line-of-Sight
NR	No Reference
NUM	Network Utility Maximization
OFDM	Orthogonal Frequency-Division Multiplexing
OPEX	Operational Expenditure
OSI	Open Systems Interconnection
OTT	Over-the-Top
P2P	Peer-to-Peer
PC	Personal Computer
PD	Proportional-Derivative
PI	Proportional-Integral
PID	Proportional-Integral-Derivative
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience

QoS	Quality of Service
RLM	Receiver-Driven Layered Multicast
RMSRE	Root Mean Square Relative Error
RR	Reduced Reference
RTCP	Real-Time Control Protocol
RTP	Real-Time Transport Protocol
RTSP	Real-Time Streaming Protocol
RTT	Round-Trip Time
SAND	Server and Network Assisted DASH
SES	Simple Exponential Smoothing
SIF	System Influence Factor
SINR	Signal-to-Interference-plus-Noise Ratio
SLA	Service-Level Agreement
SMA	Simple Moving Average
SSD	Sum of Squared Differences
SSI	Signal Strength Indicator
SSIM	Structural Similarity
STB	Set-Top Box
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
UHD	Ultra-High-Definition
URL	Uniform Resource Locator
VBR	Variable Bit Rate
VCR	Videocassette Recorder
VoD	Video on Demand
VQM	Video Quality Metric
VR	Virtual Reality

W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Network
XAP	Silverlight Application Package
XML	Extensible Markup Language

Appendix B

Publications

This appendix contains the publication list of the author of the presented thesis. Publications containing content included in this thesis are marked with a "*".

Journal papers:

* Konstantin Miller, Abdel-Karim Al-Tamimi, and Adam Wolisz. QoE-Based Low-Delay Live Streaming Using Throughput Predictions. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 13(1), Nov. 2016

Niels Karowski, Konstantin Miller, and Adam Wolisz. Greedy Multi-Channel Neighbor Discovery. *Submitted, arXiv peprint 1506.05255*, Nov. 2015

* Konstantin Miller, Dilip Bethanabhotla, Giuseppe Caire, and Adam Wolisz. A Control-Theoretic Approach to Adaptive Video Streaming in Dense Wireless Networks. *IEEE Transactions on Multimedia*, 17(8):1309-1322, Aug. 2015

Tobias Harks, and Konstantin Miller. The Worst-Case Efficiency of Cost Sharing Methods in Resource Allocation Games. *Operations Research*, 59(6):1491-1503, Dec. 2011

Conference proceedings:

* Konstantin Miller, Nicola Corda, Savvas Argyropoulos, Alexander Raake, and Adam Wolisz. Optimal Adaptation Trajectories for Block-Request Adaptive Video Streaming. In *Proc. of the Packet Video Workshop (PV)*, San Jose, CA, USA, Dec. 2013

***** Konstantin Miller, Emanuele Quacchio, Gianluca Gennari, and Adam Wolisz. Adaptation Algorithm for Adaptive Streaming over HTTP. In *Proc. of the Packet Video Workshop (PV)*, Munich, Germany, May 2012

Kai-Simon Goetzmann, Tobias Harks, Max Klimm, and Konstantin Miller. Optimal File Distribution in Peer-to-Peer Networks. In *Proc. of the Symposium on Algorithms and Computation (ISAAC)*, Yokohama, Japan, Dec. 2011

Konstantin Miller and Adam Wolisz. Transport Optimization in Peer-to-Peer Networks. In *Proc. of the Euromicro Conference on Parallel, Distributed and Network-Based Computing (PDP)*, Ayia Napa, Cyprus, Feb. 2011

Konstantin Miller, Thorsten Biermann, Hagen Woesner, and Holger Karl. Network Coding in Passive Optical Networks. In *Proc. of the IEEE Symposium on Network Coding (NetCod)*, Toronto, ON, Canada, June 2010

Jorge Carapinha, Roland Bless, Christoph Werle, Konstantin Miller, Virgil Dobrota, Andrei Bogdan Rus, Heidrun Grob-Lipski, and Horst Roessler. Quality of Service in the Future Internet. In *Proc. of the ITU-T Kaleidoscope*, Pune, Maharashtra, India, Dec. 2010

Tobias Harks and Konstantin Miller. Efficiency and Stability of Nash Equilibria in Resource Allocation Games. In *Proc. of the Conference on Game Theory for Networks (GameNets)*, Istanbul, Turkey, Mar. 2009

Konstantin Miller and Tobias Harks. Utility Max-Min Fair Congestion Control With Time-Varying Delays. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, Phoenix, AZ, USA, Apr. 2008

Technical reports:

***** Konstantin Miller, Abdel-Karim Al-Tamimi, and Adam Wolisz. QoE-Based Low-Delay Live Streaming Using Throughput Predictions. Technical Report TKN-16-001, Telecommunication Networks Group (TKN), Technische Universität Berlin, Mar. 2016

Niels Karowski, and Konstantin Miller. Optimized Asynchronous Passive Multi-Channel Discovery of Beacon-Enabled Networks. Technical Report TKN-15-002, Telecommunication Networks Group (TKN), Technische Universität Berlin, Mar. 2015

Konstantin Miller, Abdel-Karim Al-Tamimi, and Adam Wolisz. Low-Delay Adaptive Video Streaming Based on Short-Term TCP Throughput Prediction. Technical Report TKN-15-001, Telecommunication Networks Group (TKN), Technische Universität Berlin, Feb. 2015

Tobias Harks and Konstantin Miller. The Impact of Marginal Cost Pricing in Resource Allocation Games. Technical Report 032-2008, Combinatorial Optimization and Graph Algorithms (COGA) Group, Technische Universität Berlin, Nov. 2008

Patents:

***** Konstantin Miller and Emanuele Quacchio. Media-Quality Adaptation Mechanisms for Dynamic Adaptive Streaming. *Pending patent, no. USPTO US 13/775,885*, 2013

References

[1] A. S. Abdallah and A. B. Mackenzie. A Cross-Layer Controller for Adaptive Video Streaming over IEEE 802.11 Networks. In *Proc. of IEEE International Conference on Communications (ICC)*, 2015.

[2] K. Ahnert and M. Mulansky. Odeint - Solving Ordinary Differential Equations in C++. In *Proc. Numerical Analysis and Applied Mathematics (ICNAAM)*, 2011.

[3] S. Akhshabi, A. C. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2011.

[4] M. Al-Bado, C. Sengul, and R. Merz. What Details Are Needed For Wireless Simulations? - A Study of a Site-Specific Indoor Wireless Model. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[5] J. Alvarez-Ramirez, R. Kelly, and I. Cervantes. Semiglobal Stability of Saturated Linear PID Control for Robot Manipulators. *Automatica*, 39(6):989–995, 2003.

[6] E. Amir, S. McCanne, and H. Zhang. An Application Level Video Gateway. In *Proc. of ACM Multimedia (MM)*, 1995.

[7] J. G. Andrews. Seven Ways that HetNets are a Cellular Paradigm Shift. *IEEE Communications Magazine*, 51(3):136–144, 2013.

[8] A. Arefin, Z. Huang, K. Nahrstedt, and P. Agarwal. 4D TeleCast: Towards Large Scale Multi-Site and Multi-View Dissemination of 3DTI Contents. In *Proc. of International Conference on Distributed Computing Systems (ICDCS)*, 2012.

[9] R. D. Armstrong, P. Sinha, and A. A. Zoltners. The Multiple-Choice Nested Knapsack Model. *Management Science*, 28(1):34–43, 1982.

[10] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[11] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A Quest for an Internet Video Quality-of-Experience Metric. In *Proc. of ACM Workshop on Hot Topics in Networks (HotNets)*, 2012.

[12] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *Proc. of ACM SIGCOMM*, 2013.

[13] A. Begen, T. Akgul, and M. Baugher. Watching Video over the Web: Part 1: Streaming Protocols. *IEEE Internet Computing*, 15(2):54–63, 2011.

[14] A. Begen, T. Akgul, and M. Baugher. Watching Video over the Web: Part 2: Applications, Standardization, and Open Issues. *IEEE Internet Computing*, 15(3):59–63, 2011.

[15] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. Request for Comments (RFC) 1945, Internet Engineering Task Force (IETF), 1996.

[16] D. Bethanabhotla, G. Caire, and M. J. Neely. Adaptive Video Streaming for Wireless Networks With Multiple Users and Helpers. *IEEE Transactions on Communications*, 63 (1):268–285, 2015.

[17] Blender Foundation. Big Buck Bunny movie. http://www.bigbuckbunny.org. Accessed: 24 Jun 2016.

[18] A. Bokani, M. Hassan, and S. Kanhere. HTTP-Based Adaptive Streaming for Mobile Clients using Markov Decision Process. In *Proc. of Packet Video Workshop (PV)*, 2013.

[19] R. Bush and D. Meyer. Some Internet Architectural Guidelines and Philosophy. Request for Comments (RFC) 3439, Internet Engineering Task Force (IETF), 2002.

[20] P. L. Callet, S. Möller, and A. Perkis. Qualinet White Paper on Definitions of Quality of Experience. Technical report, European Network on Quality of Experience in Multimedia Systems and Services (QUALINET), 2013.

[21] J. Carapinha, R. Bless, C. Werle, K. Miller, V. Dobrota, A. B. Rus, H. Grob-Lipski, and H. Roessler. Quality of Service in the Future Internet. In *Proc. of ITU-T Kaleidoscope*, 2010.

[22] M. Carbone and L. Rizzo. Dummynet Revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12, 2010.

[23] G. Carlucci, L. D. Cicco, and S. Mascolo. HTTP over UDP: an Experimental Investigation of QUIC. In *Proc. of ACM/SIGAPP Symposium On Applied Computing (SAC)*, 2015.

[24] S. Carmel, T. Daboosh, E. Reifman, N. Shani, Z. Eliraz, D. Ginsberg, E. Ayal, and K. Saba. Network Media Streaming. Patent No. 6,389,473. Filed 24 Mar 1999, Issued 14 May 2002.

[25] V. G. Cerf and R. E. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5), 1974.

[26] V. Chandrasekhar, J. G. Andrews, and A. Gatherer. Femtocell Networks: A Survey. *IEEE Communications Magazine*, 46(9):59–67, 2008.

[27] C. Chatfield. *The Analysis of Time Series: an Introduction*. Taylor & Francis, 6th edition, 2003.

[28] C. Chen, X. Zhu, G. D. Veciana, A. C. Bovik, and R. W. Heath, Jr. Rate Adaptation and Admission Control for Video Transmission with Subjective Quality Constraints. *IEEE Journal of Selected Topics in Signal Processing*, 9(1):22–35, 2015.

[29] S. Chen, Z. Gao, and K. Nahrstedt. F.Live: Towards Interactive Live Broadcast FTV Experience. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2016.

[30] Y. Chen, K. Xie, F. Zhang, P. Pandit, and J. Boyce. Frame Loss Error Concealment for SVC. *Journal of Zhejiang University SCIENCE A*, 7(5):677–683, 2006.

[31] Y. Chen, K. Wu, and Q. Zhang. From QoS to QoE: A Tutorial on Video Quality Assessment. *IEEE Communications Surveys & Tutorials*, 17(2):1126–1165, 2015.

[32] Z. Chen, S.-M. Tan, R. H. Campbell, and Y. Li. Real Time Video and Audio in the World Wide Web. In *Proc. of International World Wide Web Conference*, 1995.

[33] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam. Objective Video Quality Assessment Methods: A Classification, Review, and Performance Comparison. *IEEE Transactions on Broadcasting*, 57(2):165–182, 2011.

[34] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456–1471, 2002.

[35] S. Cicalò and V. Tralli. Distortion-Fair Cross-Layer Resource Allocation for Scalable Video Transmission in OFDMA Wireless Networks. *IEEE Transactions on Multimedia*, 16(3): 848–863, 2014.

[36] L. D. Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. TAPAS: A Tool for Rapid Prototyping of Adaptive Streaming Algorithms. In *Proc. of VideoNext: Design, Quality and Deployment of Adaptive Video Streaming*, 2014.

[37] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2013 - 2018. White Paper, Cisco Systems, Inc., 2014.

[38] M. Claeys, S. Latré, J. Famaey, and F. De Turck. Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client. *IEEE Communications Letters*, 18(4):716–719, 2014.

[39] K. Cleary. Video on Demand - Competing Technologies and Services. In *Proc. of the International Broadcasting Convention (IBC)*, 1995.

[40] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. First Workshop on Economics of Peer-to-Peer Systems*, 2003.

[41] comscore. U.S. Digital Future in Focus. White Paper, comScore, Inc., 2014.

[42] G. J. Conklin, G. S. Greenbaum, K. O. Lillevold, A. F. Lippman, and Y. A. Reznik. Video Coding for Streaming Media Delivery on the Internet. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):269–281, 2001.

[43] Conviva. Viewer Experience Report. Report, Conviva, 2014.

[44] Y. Cui, B. Li, and K. Nahrstedt. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. *IEEE Journal on Selected Areas in Communications*, 22(1):91–106, 2004.

[45] dash.js. dash.js: DASH Industry Forum Reference Player. http://dashif.org/reference/players/javascript/. Accessed: 25 Jun 2016.

[46] S. Deering. Host Extension for IP Multicasting. Request for Comments (RFC) 1112, Internet Engineering Task Force (IETF), 1989.

[47] J. K. Dey-Sircar, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing VCR Capabilities in Large-Scale Video Servers. In *Proc. of ACM Multimedia (MM)*, 1994.

[48] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, 2000.

[49] I. Djama, T. Ahmed, A. Nafaa, and R. Boutaba. Meet In the Middle Cross-Layer Adaptation for Audiovisual Content Delivery. *IEEE Transactions on Multimedia*, 10(1):105–120, 2008.

[50] Dynamic adaptive. Dynamic Adaptive Streaming over HTTP (DASH), First Edition. International Standard ISO/IEC 23009, 2012.

[51] Dynamic adaptive. Dynamic Adaptive Streaming over HTTP (DASH), Second Edition. International Standard ISO/IEC 23009, 2014.

[52] Dynamic adaptive. Dynamic Adaptive Streaming over HTTP (DASH), Part 5: Server and Network Assisted DASH (SAND). International Standard ISO/IEC 23009-5, 2015.

[53] H. Erikson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[54] A. E. Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. Steinbach. Quality-of-Experience Driven Adaptive HTTP Media Delivery. In *Proc. IEEE International Conference on Communications (ICC)*, 2013.

[55] K. Evensen, T. Kupka, D. Kaspar, P. Halvorsen, and C. Griwodz. Quality-Adaptive Scheduling for Live Streaming over Multiple Access Networks. In *Proc. of Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2010.

[56] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. F. Hansen, and P. Engelstad. Using Bandwidth Aggregation to Improve the Performance of Quality-Adaptive Streaming. *Signal Processing: Image Communication*, 27(4):312–328, 2012.

[57] K. Evensen, T. Kupka, H. Riiser, P. Ni, R. Eg, C. Griwodz, and P. Halvorsen. Adaptive Media Streaming to Mobile Devices: Challenges, Enhancements, and Recommendations. *Advances in Multimedia*, pages 1–21, 2014.

[58] J. L. Feuvre and C. Concolato. Tiled-Based Adaptive Streaming using MPEG-DASH. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2016.

[59] R. Fielding, U. C. Irvine, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. Request for Comments (RFC) 2616, Internet Engineering Task Force (IETF), 1999.

[60] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. of ACM SIGCOMM*, 2000.

[61] J. W. Forgie. ST - A Proposed Internet Stream Protocol. Internet Experiment Note (IEN) 119, 1979.

[62] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proc. of International Workshop on Quality of Service (IWQoS)*, 1999.

[63] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire. Femtocaching and Device-to-Device Collaboration: A New Architecture for Wireless Video Distribution. *IEEE Communications Magazine*, 51(4):142–149, 2013.

[64] J. Goshi, A. E. Mohr, R. E. Ladner, E. A. Riskin, and A. F. Lippman. Unequal Loss Protection for H.263 Compressed Video. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3):412–419, 2005.

[65] C. Griwodz, M. Bär, and L. C. Wolf. Long-Term Movie Popularity Models in Video-on-Demand Systems or The Life of an on-Demand Movie. In *Proc. of ACM Multimedia (MM)*, 1997.

[66] GStreamer. GStreamer Open Source Multimedia Framework. http://gstreamer.freedesktop.org. Accessed: 25 Jun 2016.

[67] Y. Guo, L. Gao, D. Towsley, and S. Sen. Smooth Workload Adaptive Broadcast. *IEEE Transactions on Multimedia*, 6(2):387–395, 2004.

[68] Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual, 2015.

[69] J. Hao, R. Zimmermann, and H. Ma. GTube: Geo-Predictive Video Streaming over HTTP in Mobile Environments. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2014.

[70] Q. He, C. Dovrolis, and M. Ammar. On the Predictability of Large Transfer TCP Throughput. *Computer Networks*, 51(14):3959–3977, 2007.

[71] W. He, X. Liu, and K. Nahrstedt. A Feedback Control Scheme for Resource Allocation in Wireless Multi-Hop Ad Hoc Networks. In *Proc. of Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, 2005.

[72] W. He, K. Nahrstedt, and X. Liu. End-to-End Delay Control of Multimedia Applications over Multihop Wireless Links. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 5(2):1–20, 2008.

[73] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong. A Random Linear Network Coding Approach to Multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[74] M. Hollander, D. A. Wolfe, and E. Chicken. *Nonparametric Statistical Methods*. Wiley, 3rd edition, 2014.

[75] F. Hoppensteadt. Asymptotic Stability in Singular Perturbation Problems. II: Problems Having Matched Asymptotic Expansion Solutions. *Journal of Differential Equations*, 15: 510–521, 1974.

[76] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. Initial Delay vs. Interruptions: Between the Devil and the Deep Blue Sea. In *Proc. of Workshop on Quality of Multimedia Experience (QoMEX)*, 2012.

[77] J. Hoydis and M. Kobayashi. Green Small-Cell Networks. *IEEE Vehicular Technology Magazine*, 6(1):37–43, 2011.

[78] J. Hoydis, S. ten Brink, and M. Debbah. Massive MIMO: How Many Antennas Do We Need? In *Proc. Allerton Conference*, 2011.

[79] HTTP 0.9. HTTP 0.9. http://www.w3.org/Protocols/HTTP/AsImplemented.html, 1991. Accessed: 25 Jun 2016.

[80] H. Hu, X. Zhu, Y. Wang, R. Pan, J. Zhu, and F. Bonomi. Proxy-Based Multi-Stream Scalable Video Adaptation Over Wireless Networks Using Subjective Quality and Rate Models. *IEEE Transactions on Multimedia*, 15(7):1638–1652, 2013.

[81] K. A. Hua and F. Xie. A Dynamic Stream Merging Technique for Video-on-Demand Services over Wireless Mesh Access Networks. In *Prof. of IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2010.

[82] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proc. of ACM Internet Measurement Conference (IMC)*, 2012.

[83] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang. Challenges, Design and Analysis of a Large-Scale P2P-VoD System. *ACM SIGCOMM Computer Communication Review*, 38(4):375–388, 2008.

[84] Z. Huang and K. Nahrstedt. Perception-Based Playout Scheduling for High-Quality Real-Time Interactive Multimedia. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[85] H. Huh, G. Caire, H. C. Papadopoulos, and S. a. Ramprashad. Achieving "Massive MIMO" Spectral Efficiency with a Not-so-Large Number of Antennas. *IEEE Transactions on Wireless Communications*, 11(9):3226–3239, 2012.

[86] R. J. Hyndman and Y. Khandakar. Automatic Time Series Forecasting: The Forecast Package for R. *Journal of Statistical Software, University of California, Los Angeles, Department of Statistics*, 27(3):1–22, 2008.

[87] R. J. Hyndman, M. L. King, I. Pitrun, and B. Billah. Local Linear Forecasts Using Cubic Smoothing Splines. *Australian & New Zealand Journal of Statistics*, 47(1):87–99, 2005.

[88] ITU-T. One-Way Transmission Time. Recommendation G.114, International Telecommunication Union (ITU), 2003.

[89] ITU-T. Definition of Terms Related to Quality of Service. Recommendation E.800, International Telecommunication Union (ITU), 2008.

[90] ITU-T. Vocabulary for Performance and Quality of Service, Amendment 2: New Definitions for Inclusion in Recommendation ITU-T P.10/G.100. Recommendation, International Telecommunication Union (ITU), 2008.

[91] ITU-T. Methodology for the Subjective Assessment of the Quality of Television Pictures. Recommendation BT.500-13, International Telecommunication Union (ITU), 2012.

[92] ITU-T. Requirements for Low-Latency Interactive Multimedia Streaming. Recommendation F.746.1, International Telecommunication Union (ITU), 2014.

[93] ITU-T. Hybrid Perceptual Bitstream Models for Objective Video Quality Measurements. Recommendation J.343, International Telecommunication Union, 2014.

[94] S. Jacobs and A. Eleftheriadis. Streaming Video Using Dynamic Rate Shaping and TCP Congestion Control. *Journal of Visual Communication and Image Representation*, 9(3): 211–222, 1998.

[95] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience, 1991.

[96] D. Jannach, K. Leopold, C. Timmerer, and H. Hellwagner. A Knowledge-Based Framework for Multimedia Adaptation. *Applied Intelligence*, 24(2):109–125, 2006.

[97] D. Jarnikov and T. Özçelebi. Client Intelligence for Adaptive Streaming Solutions. *Signal Processing: Image Communication*, 26(7):378–389, 2011.

[98] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With FESTIVE. *IEEE Transactions on Networking*, 22 (1):326–340, 2014.

[99] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, Å. Kvalnes, C. Gurrin, S. Zav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwidz, and P. Halvorsen. DAVVI: A Prototype for the Next Generation Multimedia Entertainment Platform. In *Proc. of ACM Multimedia (MM)*, 2009.

[100] W. John and S. Tafvelin. Analysis of Internet Backbone Traffic and Header Anomalies Observed. In *Proc. of ACM Internet Measurement Conference (IMC)*, 2007.

[101] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*. Wiley, 2nd edition, 1994.

[102] V. Joseph and G. De Veciana. NOVA: QoE-Driven Optimization of DASH-Based Video Delivery in Networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2014.

[103] S. Jumisko-Pyykkö and T. Vainio. Framing the Context of Use for Mobile HCI. *International Journal of Mobile Human Computer Interaction*, 2(4):1–28, 2010.

[104] H. Kanakia, P. P. Mishra, and A. R. Reibman. An Adaptive Congestion Control Scheme for Real Time Packet Video Transport. *IEEE/ACM Transactions on Networking*, 3(6): 671–682.

[105] K. Kang, W. J. Jeon, K.-J. Park, R. H. Campbell, and K. Nahrstedt. Cross-Layer Quality Assessment of Scalable Video Services on Mobile Embedded Systems. *IEEE Transactions on Mobile Computing*, 9(10):1478–1490, 2010.

[106] D. Kaspar, K. Evensen, P. Engelstad, and A. F. Hansen. Using HTTP Pipelining to Improve Progressive Download over Multiple Heterogeneous Interfaces. In *Proc. of IEEE International Conference on Communications (ICC)*, 2010.

[107] J. Klaue, B. Rathke, and A. Wolisz. EvalVid - A Framework for Video Transmission and Quality Evaluation. In *Proc. of Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*, 2003.

[108] M. A. Klompenhouwer. Flat Panel Display Signal Processing. PhD Thesis, Technische Universiteit Eindhoven, 2006.

[109] C. Krasic, K. Li, and J. Walpole. The Case for Streaming Multimedia with TCP. In *Proc. of Interactive Distributed Multimedia Systems (IDMS)*, 2001.

[110] C. Krasic, J. Walpole, and W.-C. Feng. Quality-Adaptive Media Streaming by Priority Drop. In *Proc. of Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2003.

[111] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, 2013.

[112] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen. Cache-Centric Video Recommendation: An Approach to Improve the Efficiency of YouTube Caches. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 11(4):1–20, 2015.

[113] T. Kupka, P. Halvorsen, and C. Griwodz. An Evaluation of Live Adaptive HTTP Segment Streaming Request Strategies. In *Proc. of IEEE Conference on Local Computer Networks (LCN)*, 2011.

[114] P. Kyosti, J. Meinila, L. Hentila, X. Zhao, T. Jamsa, C. Schneider, M. Narandzic, M. Milojevic, A. Hong, J. Ylitalo, V.-M. Holappa, M. Alatossava, R. Bultitude, Y. de Jong, and T. Rautiainen. WINNER II Channel Models. Deliverable D1.1.2, V1.2, IST-4-027756, 2007.

[115] J. Lazzaro. Framing Real-Time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. Request for Comments (RFC) 4571, Internet Engineering Task Force (IETF), 2006.

[116] H. Le, A. Behboodi, and A. Wolisz. Quality Driven Resource Allocation for Adaptive Video Streaming in OFDMA Uplink. In *Proc. IEEE Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, 2015.

[117] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang. Buffer-Based Bitrate Adaptation for Adaptive HTTP Streaming. In *Proc. Conference on Advanced Technologies for Communications (ATC)*, 2013.

[118] H. T. Le, H. N. Nguyen, N. P. Ngoc, A. T. Pham, H. L. Minh, and T. C. Thang. Quality-Driven Bitrate Adaptation Method for HTTP Live-Streaming. In *Proc. IEEE International Conference on Communication Workshop (ICCW)*, 2015.

[119] J.-Y. Le Boudec. *Performance Evaluation of Computer and Communication Systems.* EPFL Press, 2.3 edition, 2015.

[120] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2012.

[121] T. Leighton. Improving Performance on the Internet. *Communications of the ACM*, 52 (2):44–51, 2009.

[122] B. Lewcio, B. Belmudez, T. Enghardt, and S. Möller. On the Way to High-Quality Video Calls in Future Mobile Networks. In *Proc. of Workshop on Quality of Multimedia Experience (QoMEX)*, 2011.

[123] B. Li and J. Liu. Multirate Video Multicast over the Internet: An Overview. *IEEE Network*, 17(1):24–29, 2003.

[124] B. Li and K. Nahrstedt. A Control-Based Middleware Framework for Quality-of-Service Adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9):1632–1650, 1999.

[125] B. Li, Z. Wang, J. Liu, and W. Zhu. Two Decades of Internet Video Streaming: A Retrospective View. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1s):1–20, 2013.

[126] M. Li, Z. Chen, and Y.-P. Tan. Scalable Resource Allocation for SVC Video Streaming Over Multiuser MIMO-OFDM Networks. *IEEE Transactions on Multimedia*, 15(7):1519–1531, 2013.

[127] W. Li. Overview of Fine Granularity Scalability in MPEG-4 Video Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):301–317, 2001.

[128] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming Video over HTTP with Consistent Quality. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2014.

[129] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[130] E. Y.-H. Lin. A Biblographical Survey on Some Well-Known Non-Standard Knapsack Problems. *INFOR*, 36(4):274–317, 1998.

[131] W. Lin and C.-C. Jay Kuo. Perceptual Visual Quality Metrics: A Survey. *Journal of Visual Communication and Image Representation*, 22(4):297–312, 2011.

[132] A. F. Lippman. Video Coding for Multiple Target Audiences. In *Proc. of Visual Communications and Image Processing*, 1999.

[133] C. Liu, I. Bouazizi, and M. Gabbouj. Rate Adaptation for Adaptive HTTP Streaming. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2011.

[134] C. Liu, I. Bouazizi, M. M. Hannuksela, and M. Gabbouj. Rate Adaptation for Dynamic Adaptive Streaming over HTTP in Content Distribution Network. *Signal Processing: Image Communication*, 27(4):288–311, 2012.

[135] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *Proc. of ACM SIGCOMM*, 2012.

[136] Y. Liu and J. Y. B. Lee. On Adaptive Video Streaming with Predictable Streaming Performance. In *Proc. of IEEE Global Communications Conference (GLOBECOM)*, 2014.

[137] Y. Liu, Y. Guo, and C. Liang. A Survey on Peer-to-Peer Video Streaming Systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.

[138] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann. Dynamic Adaptive HTTP Streaming of Live Content. In *Proc. of IEEE Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2011.

[139] T. L. Marzetta. Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas. *IEEE Transactions on Wireless Communications*, 9(11):3590–3600, 2010.

[140] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. *ACM SIGCOMM Computer Communication Review*, 26(4):117–130, 1996.

[141] A. Michaloliakos, R. Rogalin, Y. Zhang, K. Psounis, and G. Caire. Performance Modeling of Next-Generation WiFi Networks. *Computer Networks*, 2016.

[142] Microsoft Silverlight. Microsoft Silverlight Multimedia Player. http://www.microsoft.com/silverlight/. Accessed: 25 Jun 2016.

[143] K. Miller and T. Harks. Utility Max-Min Fair Congestion Control with Time-Varying Delays. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2008.

[144] M. Mirza, J. Sommers, P. Barford, and X. Zhu. A Machine Learning Approach to TCP Throughput Prediction. *IEEE/ACM Transactions on Networking*, 18(4):1026–1039, 2010.

[145] R. Mohan, J. R. Smith, and C.-S. Li. Adapting Multimedia Internet Content for Universal Access. *IEEE Transactions on Multimedia*, 1(1):104–114, 1999.

[146] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. QDASH: A QoE-Aware DASH System. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2012.

[147] A. F. Molisch. *Wireless Communications*. Wiley, 2010.

[148] C. Müller, S. Lederer, and C. Timmerer. An Evaluation of Dynamic Adaptive Streaming Over HTTP in Vehicular Environments. In *Proc. of Workshop on Mobile Video (MoVid)*, 2012.

[149] J.-R. Ohm. Advances in Scalable Video Coding. *Proceedings of the IEEE*, 93(1):42–56, 2005.

[150] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand. Comparison of the Coding Efficiency of Video Coding Standards – Including High Efficiency Video Coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1669–1684, 2012.

[151] E. H. Ong, J. Kneckt, O. Alanen, Z. Chang, T. Huovinen, and T. Nihtila. IEEE 802.11ac: Enhancements for Very High Throughput WLANs. In *Proc. of IEEE Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2011.

[152] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8 (2):133–145, 2000.

[153] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. of Workshop on Peer-to-Peer Systems IV*, 2005.

[154] Z. Pan, Y. Ikuta, M. Bandai, and T. Watanabe. A User Dependent System for Multi-View Video Transmission. In *Proc. of Conference on Advanced Information Networking and Applications (AINA)*, 2011.

[155] T. D. Pessemier, K. D. Moor, W. Joseph, L. D. Marez, and L. Martens. Quantifying the Influence of Rebuffering Interruptions on the User's Quality of Experience During Mobile Video Watching. *IEEE Transactions on Broadcasting*, 59(1):47–61, 2013.

[156] picochip. The Case For Home Base Stations. Technical Report, Picochip, 2007.

[157] M. H. Pinson and S. Wolf. A New Standardized Method for Objectively Measuring Video Quality. *IEEE Transactions on Broadcasting*, 50(3):312–322, 2004.

[158] J. Postel. User Datagram Protocol. Request for Comments (RFC) 768, Internet Engineering Task Force (IETF), 1980.

[159] R. Pries, Z. Magyari, and P. Tran-Gia. An HTTP Web Traffic Model Based On the Top One Million Visited Web Pages. In *Proc. of Conference on Next Generation Internet (NGI)*, 2012.

[160] H.-T. Quan and M. Ghanbari. Temporal Aspect of Perceived Quality in Mobile Video Broadcasting. *IEEE Transactions on Broadcasting*, 54(3):641–651, 2008.

[161] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, 1st edition, 2006.

[162] U. Reiter, K. Brunnström, K. D. Moor, M.-C. Larabi, M. Pereira, A. Pinheiro, J. You, and A. Zgank. Factors Influencing Quality of Experience. In *Quality of Experience*, pages 55–74. Springer International Publishing, 2014.

[163] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Congestion Controlled Playback Video over the Internet. In *Proc. of ACM SIGCOMM*, 1999.

[164] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 1999.

[165] H. Riiser, P. Halvorsen, C. Griwodz, and D. Johansen. Low Overhead Container Format for Adaptive Streaming. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2010.

[166] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 8(3):1–19, 2012.

[167] F. Riley and T. R. Henderson. The ns-3 Network Simulator. In *Modeling and Tools for Network Simulation*. Springer, 2010.

[168] F. Rusek, D. Persson, B. K. Lau, E. G. Larsson, T. L. Marzetta, O. Edfors, and F. Tufvesson. Scaling Up MIMO. *IEEE Signal Processing Magazine*, 30(1):40–60, 2012.

[169] A. Sackl, K. Masuch, S. Egger, and R. Schatz. Wireless vs. Wireline Shootout: How User Expectations Influence Quality of Experience. In *Proc. of Workshop on Quality of Multimedia Experience (QoMEX)*, 2012.

[170] H. Saki and M. Shikh-Bahaei. Cross-Layer Resource Allocation for Video Streaming over OFDMA Cognitive Radio Networks. *IEEE Transactions on Multimedia*, 17(3):333–345, 2015.

[171] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[172] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Request for Comments (RFC) 2326, Internet Engineering Task Force (IETF), 1998.

[173] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Request for Comments (RFC) 3550, Internet Engineering Task Force (IETF), 2003.

[174] S. Sesia, I. Toufik, and M. Baker. *LTE: The Long Term Evolution - From Theory to Practice*. Wiley, 2009.

[175] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hossfeld, and P. Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2014.

[176] S. H. Shah, K. Chen, and K. Nahrstedt. Available Bandwidth Estimation in IEEE 802.11-Based Wireless Networks. In *Proc. of ISMA Bandwidth Estimation Workshop (BEst)*, 2003.

[177] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire. FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers. *IEEE Transactions on Information Theory*, 59(12):8402–8413, 2013.

[178] T. Silverston and O. Fourmaux. Measuring P2P IPTV Systems. In *Proc. of Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2007.

[179] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino. Quality of Experience Estimation for Adaptive HTTP/TCP Video Streaming Using H.264/AVC. In *Proc. of IEEE Consumer Communications and Networking Conference (CCNC)*, 2012.

[180] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 18(4):62–67, 2011.

[181] W. Song and D. W. Tjondronegoro. Acceptability-Based QoE Models for Mobile Video. *IEEE Transactions on Multimedia*, 16(3):738–750, 2014.

[182] F. Speranza, F. Poulin, R. Renaud, M. Caron, and J. Dupras. Objective and Subjective Quality Assessment with Expert and Non-Expert Viewers. In *Proc. of Workshop on Quality of Multimedia Experience (QoMEX)*, 2010.

[183] T. Stockhammer. Dynamic Adaptive Streaming over HTTP – Standards and Design Principles. In *Proc. of ACM Multimedia Systems Conference (MMSys)*, 2011.

[184] G. J. Sullivan and T. Wiegand. Rate-Distortion Optimization for Video Compression. *IEEE Signal Processing Magazine*, 15(6):74–90, 1998.

[185] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[186] V. Sundaram and K. A. Hua. Seamless Video Streaming: A Light Weight Session Handoff Scheme for Dynamic Stream Merging Based Wireless Mesh Networks. In *Proc. of IEEE International Conference on Multimedia and Expo Workshops Seamless (ICME)*, 2012.

[187] P. Sweeting. Video in 2014: Going Live and Over the Top. Research Report, Gigaom, 2014.

[188] W. J. Tam, F. Speranza, S. Yano, K. Shimono, and H. Ono. Stereoscopic 3D-TV: Visual comfort. *IEEE Transactions on Broadcasting*, 57(2):335–346, 2011.

[189] M. Tamai, K. Yasumoto, N. Shibata, M. Ito, and K. Nahrstedt. Transcasting: Cost-Efficient Video Multicast for Heterogeneous Mobile Terminals. In *Proc. of IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, 2008.

[190] B. Tan and L. Massoulié. Optimal Content Placement for Peer-to-Peer Video-on-Demand Systems. *IEEE/ACM Transactions on Networking*, 21(2):566–579, 2013.

[191] W.-t. Tan and A. Zakhor. Real-Time Internet Video Using Error Resilient Scalable Compression and TCP-Friendly Transport Protocol. *IEEE Transactions on Multimedia*, 1(2): 172–186, 1999.

[192] M. Tanimoto. Overview of Free Viewpoint Television. *Signal Processing: Image Communication*, 21(6):454–461, 2006.

[193] K. Tappayuthpijarn, T. Stockhammer, and E. Steinbach. HTTP-Based Scalable Video Streaming Over Mobile Networks. In *Proc. of IEEE International Conference on Image Processing (ICIP)*, 2011.

[194] S. Tarbouriech and M. Turner. Anti-Windup Design: An Overview of Some Recent Advances and Open Problems. *IET Control Theory and Applications*, 3(1):1–19, 2009.

[195] D. B. Terry and D. C. Swinehart. Managing Stored Voice in the Etherphone System. *ACM Transactions on Computer Systems*, 6(1):3–27, 1988.

[196] T. C. Thang, H. T. Le, A. T. Pham, and Y. M. Ro. An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming. *IEEE Journal on Selected Areas in Communications*, 32(4):693–705, 2014.

[197] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proc. of ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.

[198] C. Timmerer and A. Bertoni. Advanced Transport Options for the Dynamic Adaptive Streaming over HTTP. *arXiv preprint*, 2016.

[199] C. Timmerer, M. Maiero, and B. Rainer. Which Adaptation Logic? An Objective and Subjective Performance Evaluation of HTTP-based Adaptive Media Streaming Systems. *arXiv preprint*, 2016.

[200] B. Trammell and J. Hildebrand. Evolving Transport in the Internet. *IEEE Internet Computing*, 18(5):60 – 64, 2014.

[201] Transmission Control Protocol. Transmission Control Protocol. Request for Comments (RFC) 793, Internet Engineering Task Force (IETF), 1981.

[202] D. Tse and P. Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[203] T. Turletti and C. Huitema. Videoconferencing on the Internet. *IEEE/ACM Transactions on Networking*, 4(3):340–351, 1996.

[204] M. van der Schaar and S. S. N. Cross-Layer Wireless Multimedia Transmission: Challenges, Principles, and New Paradigms. *IEEE Wireless Communications*, 12(4):50–58, 2005.

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

[206] VLC Media Player. VLC Media Player. http://www.videolan.org/vlc/. Accessed: 26 Jun 2016.

[207] L. Vu, I. Gupta, J. Liang, and K. Nahrstedt. Mapping the PPLive Network: Studying the Impacts of Media Streaming on P2P Overlays. Technical Report UIUCDCS-R-2006-2758, University of Illinois at Urbana-Champaign, 2006.

[208] B. W. Wah, X. Su, and D. Lin. A Survey of Error-Concealment Schemes for Real-Time Audio and Video Transmissions over the Internet. In *Proc. of Symposium on Multimedia Software Engineering (MSE)*, 2000.

[209] M. Wang and B. Li. R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming. *IEEE Journal on Selected Areas in Communications*, 25(9):1655–1666, 2007.

[210] X. Wang and H. Schulzrinne. Comparison of Adaptive Internet Multimedia Applications. *IEICE Transactions on Communications*, E82-B(6):806–818, 1999.

[211] X. Wang, T. T. Kwon, Y. Choi, H. Wang, and J. Liu. Cloud-Assisted Adaptive Video Streaming and Social-Aware Video Prefetching for Mobile Users. *IEEE Wireless Communications*, 20(3):72–79, 2013.

[212] Y. Wang and Q.-F. Zhu. Error Control and Concealment for Video Communications: A Review. *Proceedings of the IEEE*, 86(5):974–997, 1998.

[213] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[214] S. Wei and V. Swaminathan. Low Latency Live Video Streaming over HTTP 2.0. In *Proc. of Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2014.

[215] 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.

[216] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha. Streaming Video over the Internet: Approaches and Directions. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):282–300, 2001.

[217] W. Wu, A. Arefin, R. Rivas, K. Nahrstedt, R. M. Sheppard, and Z. Yang. Quality of Experience in Distributed Interactive Multimedia Environments: Toward a Theoretical Framework. In *Proc. of ACM Multimedia (MM)*, 2009.

[218] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S.-H. Jung, and R. Bajscy. TEEVE: The Next Generation Architecture for Tele-Immersive Environments. In *Proc. of IEEE Symposium on Multimedia (ISM)*, 2005.

[219] C. Yim and A. C. Bovik. Evaluation of Temporal Variation of Video Quality in Packet Loss Networks. *Signal Processing: Image Communication*, 26(1):24–38, 2011.

[220] X. Yin, V. Sekar, and B. Sinopoli. Toward a Principled Framework to Design Dynamic Adaptive Streaming Algorithms over HTTP. In *Proc. of ACM Workshop on Hot Topics in Networks (HotNets)*, 2014.

[221] L. Yitong, S. Yun, M. Yinian, L. Jing, L. Qi, and Yang Dacheng. A Study on Quality of Experience for Adaptive Streaming Service. In *Proc. of IEEE International Conference on Communications Workshops (ICC)*, 2013.

[222] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets. GRACE-1: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. *IEEE Transactions on Mobile Computing*, 5(7):799–815, 2006.

[223] A. Zambelli. IIS Smooth Streaming Technical Overview. Microsoft Corporation, 2009.

[224] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 7(5):8–18, 1993.

[225] M. Zhang, S. Member, Q. Zhang, S. Member, and L. Sun. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *IEEE Journal on Selected Areas in Communications*, 25(9):1678–1694, 2007.

[226] Q. Zhang, G. Wang, W. Zhu, and Y.-Q. Zhang. Robust Scalable Video Streaming over Internet with Network-Adaptive Congestion Control and Unequal Loss Protection. In *Proc. of Packet Video Workshop (PV)*, 2001.

[227] X. Zhang, J. Liu, B. Li, and Y.-S. Yum. Coolstreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, 2005.

[228] M. Zhao, X. Gong, J. Liang, W. Wang, X. Que, and S. Cheng. QoE-Driven Cross-Layer Optimization for Wireless Dynamic Adaptive Streaming of Scalable Videos over HTTP. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(3):451–465, 2015.

[229] C. Zhou, X. Zhang, L. Huo, and Z. Guo. A Control-Theoretic Approach to Rate Adaptation for Dynamic HTTP Streaming. In *Proc. of IEEE Visual Communications and Image Processing (VCIP)*, 2012.

[230] C. Zhou, C. W. Lin, X. Zhang, and Z. Guo. A Control-Theoretic Approach to Rate Adaption for DASH Over Multiple Content Distribution Servers. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(4):681–694, 2014.

[231] L. Zhou, N. Xiong, L. Shu, A. Vasilakos, and S.-S. Yeo. Context-Aware Middleware for Multimedia Services in Heterogeneous Networks. *IEEE Intelligent Systems*, 25(2):40–47, 2010.

[232] X. Zhu, Z. Li, R. Pan, J. Gahm, and R. Ru. Fixing Multi-Client Oscillations in HTTP-based Adaptive Streaming: A Control Theoretic Approach. In *Proc. IEEE Workshop on Multimedia Signal Processing (MMSP)*, 2013.

[233] H. Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.

[234] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective Impression of Variations in Layer Encoded Videos. In *Proc. of International Workshop on Quality of Service (IWQoS)*, 2003.

[235] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can Accurate Predictions Improve Video Streaming in Cellular Networks? In *Proc. of Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2015.