Adaptive Bitrate Video Streaming over HTTP in Mobile Wireless Networks

Haakon Riiser

June 16, 2013

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1372

ISSN 1501-7710

Cover: Inger Sandved Anfinsen.
Printed in Norway: AIT Oslo AS.

Produced in co-operation with Akademika Publishing.
The thesis is produced by Akademika publishing merely in connection with the thesis defence. Kindly direct all inquiries regarding the thesis to the copyright holder or the unit which grants the doctorate.

Abstract

The topic of this dissertation is bitrate adaptive media streaming to receivers in mobile wireless networks. This work was motivated by the recent explosion in popularity of media streaming to mobile devices. Wireless networks will always be bandwidth limited compared to fixed networks due to background noise, limited frequency spectrum, and varying degrees of network coverage and signal strength. Consequently, applications that need to move large amounts of data in a timely manner cannot simply assume that future networks will have sufficient bandwidth at all times. It is therefore important to make the applications themselves able to cope with varying degrees of connectivity.

In order to understand the requirements of streaming in 3G mobile networks, we perform a large number of measurements in Telenor's 3G network in and around Oslo. Using bandwidth traces from these field experiments, we compare commercial adaptive media streaming clients by Adobe, Apple, and Microsoft in challenging vehicular (bus, ferry, tram and metro) streaming scenarios.

In this comparison, we reveal problems with buffer underruns and unstable video playouts. We therefore develop our own adaptive bitrate media client, and design a new quality adaptation scheme that targets the requirements of mobile wireless networks, reducing the number of buffer underruns and improving stability. We also observe that network conditions are highly predictable as a function of geographical location. Simulations on bandwidth traces from field experiments indicate that the video playout can be made even more stable: A media player that knows its future (bandwidth availability and the duration of the streaming session) can use its buffer more intelligently. Fluctuations in bandwidth can be smoothed out through sophisticated buffering algorithms, resulting in a higher quality video playout with fewer interruptions due to buffer underrun.

Again using our collection of bandwidth traces, we develop a bandwidth lookup service and a new algorithm for quality scheduling that uses historic bandwidth traces to plan ahead, thus avoiding most underruns and offering a far more stable playout with fewer visually disturbing fluctuations in quality. We show that this prediction-based approach greatly improves the performance compared to our best results with non-predictive quality schedulers. Finally, we show how multi-link streaming can be employed to increase the network capacity available to the video receiver, thus improving perceived video quality even further.

All algorithms are developed and tested using custom made simulation tools, and are later verified in real world environments using a fully functional prototype implementation. We demonstrate that our proposed algorithms greatly improve performance in vehicular mobile streaming scenarios.

Acknowledgments

The work presented in this dissertation owes much to a number of different contributors. I wish to express my gratitude to the following:

Tore Langedal Endestad for developing the simulation tool used to evaluate new quality adaptation algorithms, for implementing GPS support in the Netview media client, for ideas and collaboration on papers, and for help in collecting bandwidth data through field trials.

Paul Vigmostad for collaboration on papers, and for tirelessly helping to gather bandwidth data in the field under harsh winter conditions.

Håkon Schad Bergsaker for developing the software used to compare different proprietary media players, and for meticulously performing those time consuming comparisons.

Frank Sola Hestvik for his contributions in developing reactive quality scheduling algorithms.

Kristian Evensen and Andreas Petlund for collaboration on the multi-link project, and for doing the integration work required to combine their *MULTI* framework with our adaptive media client.

Our collaborators at Telenor R&I, Harald Loktu, Bjørn Olav Hestnes and Svein Heiestad, for giving us information about Telenor's infrastructure and for giving us early access to their newly deployed 3G network.

Dag Johansen and the rest of the iAD guys for including us in their projects.

The Norwegian Research Council and Innovation Norway for their financial support, and for allowing us to redefine our project goals several times, as well as extending the project period.

Finally, but above all else, I am grateful for the assistance provided by my advisors, Prof. Carsten Griwodz and Prof. Pål Halvorsen, whose contributions are too many for an exhaustive list. Thank you for guidance, support and ideas at times where the direction of my work was unclear, for invaluable collaboration on research papers, and for your patience and understanding when progress was slow due to my other responsibilities. This work could not have been done without your help.

Contents

Chapter 1

Introduction

Hand-held devices capable of displaying high definition video have become commonplace, and high-speed mobile wireless networks are available in most populated areas in developed countries. An important application of these technologies is video streaming to mobile devices, and consequently, the number of video streaming providers targeting the mobile device market has exploded.

The subject of this dissertation is to improve the utilization of available bandwidth in mobile streaming through the use of advanced buffering strategies, improved video bitrate adaptation algorithms and multi-link streaming. Better bandwidth utilization in video streaming is important because it translates to an improved quality of experience (QoE) for the viewer.

1.1 Background and Motivation

Video streaming is a highly bandwidth intensive application, but improvments in video coding efficiency and mobile wireless network bandwidth have made it possible to perform real-time video streaming to mobile receivers using currently available technology. Examples of current video services are YouTube [46], Netflix [27], Hulu [22], TV 2 Sumo [41], BBC iPlayer [12], ESPN Player [17], Comoyo [15] and live streaming of major sports events such as the Olympics [25], Super Bowl [38], and the FIFA World Cup [19] to millions of concurrent users.

Traditional fixed-quality media streaming technologies are failing to deliver acceptable QoE for streaming at such a scale, so all significant video streaming standards developed since 2008 have been based on *adaptive bitrate streaming*. These systems are characterized by their ability to adapt the streaming bitrate to the currently available bandwidth, and to restrict the quality according to the capabilities of the device used to view the video. The most important benefit of adaptive bitrate streaming is that it reduces the number of playout interruptions due to buffer underruns, which is an important factor in determining the QoE. Adaptive

bitrate streaming also makes it easier for a single streaming system to support everything from low-end mobile devices using a slow wireless connection, to high-end HD-capable media centers with a fast fiber optic link.

Examples of adaptive streaming formats are Microsoft's *Smooth Streaming* [167], Apple's *HTTP Live Streaming (HLS)* [127], MPEG's *Dynamic Adaptive Streaming over HTTP (DASH)* [98], and Adobe's *HTTP Dynamic Streaming (HDS)* [51]. The most successful formats commercially are currently Smooth Streaming and HLS, but these may eventually be supplanted by the MPEG DASH format, which has been adopted as a true standard by the International Organization for Standardization (ISO). It is inspired by Smooth, HLS, and HDS, but is wider in scope and offers profiles that eases transition from the other formats to DASH. Smooth Streaming is supported through Microsoft's Silverlight application framework on computers and devices running the Microsoft Windows operating system, and HDS is supported on any platform with the Adobe Flash Player. HLS is supported by devices based on Apple's iOS and Google's Android operating systems, many set-top boxes and connected TVs, and Apple's Quicktime media player. Implementations of the MPEG DASH standard are, as of 2012, still immature, but many video services that build on the new Media Source Extensions [24] framework for adaptive video streaming using HTML5/JavaScript will most likely use MPEG DASH as the underlying streaming format.

Even though all aforementioned formats except DASH were created by corporations, their specifications are freely available, so many third party implementations for other devices and operating systems are also available. Examples of third party client-side implementations include the Netview[1] Media Client [7], the VideoLAN VLC Media Player [45], and GPAC [40]. Third party server-side implementations include CodeShop Unified Streaming Platform [14], Envivio 4Caster C4 [5], Anevia ViaMotion [9] and RealNetworks Helix Universal Server [33].

To mention just a few examples of services, Smooth Streaming has been used by Netflix [27] to stream various commercial content, and by NBC [26] to stream major sports events like Super Bowl [38] and the Olympics [25]. HLS is used in Apple's iTunes [11] store for long form content and is very popular in the television industry because it builds on the widely supported MPEG-2 Transport Stream video container format [95]. While adaptive bitrate streaming only consitutes about 17 % of total Internet video traffic in 2012, it is expected to exceed 50 % by 2015 [125].

Adaptive video streaming technologies make it possible to adapt the video bitrate according to the capacity of the network, but none of the existing standards for adaptive video streaming specify how to do this, and at the start of this project, there was little research available on the subject. How to best stream and adapt video to mobile receivers in the general case was an unsolved problem. A mobile

[1]Netview Technology was co-founded by the author of this dissertation, but acquired by Opera Software in 2012.

receiver will always experience varying degrees of connectivity, and in some cases the variations can be extreme and long lasting (consider the case where the receiver travels through a tunnel without network coverage, or in an area with low base station density). Consequently, new media streaming technologies optimized for streaming to mobile devices have recently received a lot of attention from the telecommunications industry. YouTube reports [47] that "*traffic from mobile devices tripled in 2011, ... more than 20 % of global YouTube views come from mobile devices, and ... YouTube is available on 350 million devices*". Similarly, Sandvine reports that "*real-time entertainment is huge, global, and growing*" [36, 21] for mobile devices, where in North America, Latin America, Europe and Asia-Pacific the audio/video downstream mobile traffic consitutes respectively 27 %, 24 %, 17 %, and 14 % of the total bandwidth consumption. Sandvine also predicts that audio and video streaming will exceed 60 % of North America's mobile data by late 2014. For mobile devices, Cisco's Visual Networking Index predicts an 18-fold increase from 2011 to 2016 [13].

This work was motivated by the aforementioned growth in mobile video streaming, and the untapped potential in adaptive video streaming technologies that can be unlocked by exploiting more of the information that is available to a mobile receiver, most importantly, geographical location, historical bandwidth measurements, and network availability.

1.2 Problem Definition

Despite ongoing advancements in wireless transmission technologies, there is a theoretical upper limit on how many bits per second can be transferred over a communications channel with limited signal bandwidth and non-zero background noise [148]. Since all wireless networks communicate over the same air medium and the available frequency spectrum is finite, it follows that the throughput[2] limit cannot keep increasing forever. Furthermore, there will always be variations in network coverage due to differences in geography and population density. Thus, one cannot simply assume that the problem with video streaming in mobile networks will be solved by waiting a few years and hoping that sufficient bandwidth will soon be universally available. Applications will always have to deal with fluctuating network bandwidth, regardless of future developments in mobile wireless networks.

For a video streaming application, there are only three ways to handle fluctuating network bandwidth: (1) Accept loss of data, (2) try to outlast the bandwidth starved periods through advanced buffering, and (3) reduce the bitrate of the video

[2]In the remainder of this dissertation, the term *bandwidth* will refer to the potential number of bits/second that can be transferred, *not* the signal bandwidth in Hz. It will be used interchangeably with the term *throughput*.

stream according to the bandwidth that is available. Data loss is usually handled with forward error correction (often in combination with data prioritization schemes), buffering has always been an important part in any non-interactive video streaming application, and mechanisms for switching bitrates in the middle of a streaming session have existed for years, and are already used in several commercial products. However, there are still many open questions regarding policies for using these mechanisms in ways more suited to mobile wireless networks.

In this dissertation, we focus our efforts on buffering and bitrate adaptation, as these techniques are most applicable to present state of the art streaming technology (almost all use reliable network protocols to transfer data, meaning that data loss does not occur and is not relevant to our work). Our goal was to improve the QoE and bandwidth utilization when streaming video in mobile wireless networks, and in order to reach it, we have explored the following key areas:

1. *Understanding the network conditions in mobile networks is crucial when designing streaming policies.* It is not possible to develop adaptive video streaming policies without a solid understanding of the underlying network characteristics, so the first step to be taken is to experimentally gather knowledge about the network conditions experienced by mobile receivers.

2. *Adaptive video streaming policies should be designed specifically for mobile receivers.* One of the benefits of adaptive video streaming is that the same stream source can work equally well for high capacity receivers on fixed networks as for low-end mobile devices. However, it follows from the previous point that the client driven video streaming policies should probably be very different when the underlying networks are different, as compromises need to be made to make a solution robust enough for challenging mobile streaming scenarios. In particular, it will be a challenge to strike the right balance between underrun protection, how rapidly the quality can adapt to the currently available bandwidth, while at the same time considering how this affects the perceived quality (rapid switching between high and low quality can actually be perceived as lower quality than playing a fixed low quality stream [169, 124]).

3. *Varying network conditions can still be predictable.* Network conditions for a mobile receiver are highly fluctuating, but they might still be predictable based on the geographical location and time (day of week and time of day). Determining if this is the case will require a large data set of bandwidth measurements from the field, at various locations and times. Gathering this data requires custom-made tools that measure bandwidth in an adaptive HTTP streaming scenario, and even performing the experiments is a significant effort in itself.

4. *Streaming in varying network conditions can be greatly improved when network conditions are successfully predicted.* Variations in connectivity can be smoothed out over time if those variations can be successfully predicted. There are a number of problems that will have to be solved to develop good bandwidth prediction algorithms, such as how to cope with inevitable mispredictions, how to make the prediction algorithm scalable (some algorithms can be extremely expensive computationally), and how to optimize for perceived quality while at the same time avoiding buffer underruns and not wasting bandwidth (e.g., ending a streaming session with too much unused video in the buffers).

5. *Performance can be improved by utilizing multiple wireless networks at the same time.* Many mobile devices today are capable of connecting to multiple types of wireless networks. Taking advantage of multiple networks within a single streaming session should improve performance compared to simply staying on one network. The challenge is how to achieve this in an application transparent way, and how to predict different network availability.

1.3 Limitations

The subject of study for this dissertation is one-way video streaming in mobile networks using adaptive bitrate video streaming technologies. Because advanced buffering strategies is a fundamental part of this, the main use case considered in this dissertation is Video on Demand (VOD) content, not Live streaming content.

The difference between VOD and Live is that VOD is based on a library of recorded movies and programs where the viewer can access any part of the content at any time, while Live streaming is more similar to a broadcasted television program. Adaptive Live streams typically offer a sliding time window of content (typically between one minute and two hours in duration), where the end of the window can be very close to a television broadcast in terms of delay. Since Live streams are often used for sports and news where it is desirable to have as little delay as possible, the viewer of a Live stream often prefers the playout position to be near the end of the time window, thereby minimizing delay. Thus, even if the available time window is long, the media client has very little room for buffering, losing maybe the most important tool for increasing robustness in a mobile streaming scenario.

Another limitation in the scope our work is that for some our more advanced approaches, the goal was to develop a proof-of-concept implementation, not necessarily a finished product (although some of the results from this dissertation have already been implemented and deployed in a commercial product). E.g., to reduce development time, some of our implementations require the user to provide information that could – or, in a real product, *should* – be done automatically.

1.4 Research Method

The Association for Computing Machinery (ACM) Task Force on the Core of Computer Science describes in [68] three major paradigms by which computer scientists approach their work:

1. The *theory* paradigm, which is rooted in mathematics, where hypotheses are proven logically.

2. The *abstraction* paradigm, which is rooted in the experimental scientific method, where theories are formed from hypotheses after they have been confirmed experimentally by testing predictions following from the hypotheses.

3. The *design* paradigm, which is rooted in engineering, where problems are solved through the construction of a system or device.

This dissertation is highly focused on practical results and commercially viable solutions, and our approach follows both the abstraction and the design paradigms: All data used to test ideas and algorithms were gathered in empirical studies, and the results were implemented and verified in fully functional prototypes used in real-world field trials.

Before any work was done on developing new streaming technologies, we performed a large number of measurements and experiments in the field. These tests were performed in the mobile wireless networks that were available at the time, mostly Telenor's 3G/High-Speed Downlink Packet Access (HSDPA) network in and around Oslo. We performed measurements using both the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), to observe both the high-level behavior that the application experiences and the low-level packet transmission characteristics that explain it. We developed our own software for UDP testing, so that we could track everything (packet loss patterns, latency, jitter, transmission errors and congestion handling). TCP performance was tested using Linux' TCP implementation, standard HTTP file transfers and tcpdump.

The result of these experiments was a large data set that made it possible to run simulations that reproduce the behavior of a real mobile network, separating this project from most related work on the subject which use synthetic bandwidth data. The data set was used to evaluate different commercial adaptive video streaming products under challenging (but realistic) network conditions. This was achieved by developing a bandwidth throttling module for the Apache web server, making it possible to reproduce the same real-world streaming session multiple times on different media players.

When developing and evalutating new algorithms, we performed experiments with a custom made network simulator. A custom made simulator was written because knowing the application, it could be made vastly more efficient than a general

network simulator such as ns-3 [28]. The correctness of the simulator was verified by comparing its results to a prototype implementation used in a real network. All developed technology was implemented in a fully functional prototype, and verified in real-world field trials.

1.5 Main Contributions

The work presented in this dissertation addresses several issues in the field of mobile video streaming. Mobile receivers are troubled by fluctuating bandwidth, making it difficult to achieve satisfactory QoE in video streaming applications. We present in this dissertation several innvative solutions to the problem, where we extend existing adaptive bitrate streaming technologies with new algorithms for quality adaptation, bandwidth prediction and multiple network utilization. A fully functional prototype implementation was developed, proving the efficiency of our suggested solutions. The following list summarizes briefly our contributions to the problems stated in section 1.2:

1. *Gathering of network data from a real 3G network.* We spent a considerable amount of time collecting data on network characteristics in a real-world 3G network. As part of these experiements, we also showed that network conditions, despite being highly variable, are actually quite deterministic as a function of geographical location.

   The collected data on network characteristics was successfully used to develop improved technologies for streaming under such conditions, and has been made available to other researchers performing similar work.

2. *An in-depth comparison of existing commercial products.* To evaluate the performance of existing commercial products in adaptive video streaming under challenging network conditions, we performed a comprehensive set of tests using the data set mentioned above. This helped expose several weaknesses in current technologies.

3. *A better quality adaptation scheme for mobile receivers.* Knowing the weaknesses of existing adaptive video streaming products made it possible to develop a quality adaptation scheme that, while directly comparable in system complexity, offers a significantly improved performance in mobile streaming scenarios, resulting in fewer buffer underruns and more stable quality.

4. *Showing that deterministic bandwidth can be used to improve performance in video streaming.* Equipped with a custom-made bandwidth prediction service based on the data set collected in the 3G network measurement phase of the project, we were able to extend our quality adaptation algorithm mentioned

above with information about future network conditions. This information made it possible to compensate for variations in the network, averaging out the quality over time and thus greatly improving QoE for the viewer.

5. *Showing that multi-link streaming is a feasible way to improve performance in an adaptive video streaming client.* We showed that using multiple different wireless networks at the same time could further improve QoE in an adaptive bitrate media client by increasing the average network capacity available for video streaming.

1.6 Outline

In chapter 2, we give an overview of developments in video streaming, in particular how and why the technology has evolved to the adaptive bitrate streaming protocols that are dominant today. Chapter 3 presents a series of experiments that expose the characteristics of 3G mobile wireless networks, which is necessary to understand and solve the problems experienced with streaming video in such networks. Chapter 4 starts with an experimental comparison of existing adaptive video streaming solutions in challenging mobile streaming scenarios, and then introduces a new quality scheduling algorithm that improves performance under such conditions. At this point, we have observed that the network conditions are highly predictable, especially with regard to geographical location. Chapter 5 presents a novel way to utilize bandwidth prediction, greatly improving the performance of the purely reactive quality scheduler introduced in chapter 4. Observing also that multiple wireless networks often are available, chapter 6 shows how multi-link streaming can be combined with the technology developed in the preceding chapters to further improve the QoE when streaming video in a mobile wireless network. Finally, we conclude our work in chapter 7.

Chapter 2

Adaptive Bitrate Streaming over HTTP

Protocols for streaming video over the Internet have existed for decades, and a large number of different protocols have been used in various degrees. This chapter will briefly go through the evolution of video streaming protocols that resulted in the adaptive bitrate streaming technologies that are most popular today, and then discuss adaptive bitrate streaming over HTTP in more detail.

2.1 A Brief History of Video Streaming

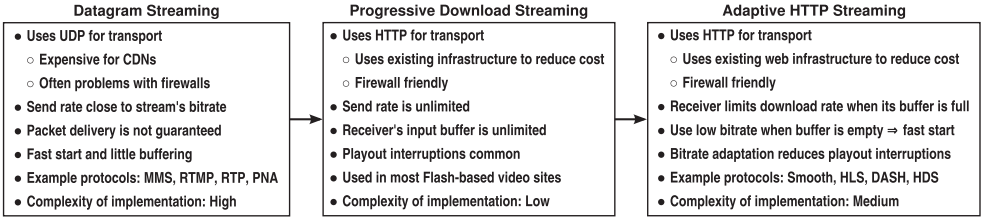


Figure 2.1: The evolution from datagram streaming to adaptive HTTP streaming.

It used to be common knowledge that real-time video over a best-effort network like the Internet would have to be streamed using a datagram protocol, giving the streaming application packet-level control. When video was streamed over the Internet, this meant in practice that it should be carried by UDP [130], not TCP [131].

Proprietary (non-open) protocols that are typically built on top of UDP include the Microsoft Media Server (MMS) protocol [120], Real Player's Progressive Networks (PNM/PNA) protocol, and Adobe's Real Time Messaging Protocol (RTMP) [50]. Non-open protocols are no longer frequently used, having largely been replaced by open standards such as the Real-time Transport Protocol (RTP) [144]. The RTP protocol is another media delivery protocol that typically uses UDP as the carrier,

but other transport protocols such as the Datagram Congestion Control Protocol (DCCP) [108, 129] and the Stream Control Transmission Protocol (SCTP) [149] are also supported, as RTP is designed to be independent of the transport protocol. RTP streaming systems can have full control over packet retransmission, enabling them to optimize for streaming applications where packet loss can be preferable to delay (e.g., audio and video conferencing). A problem with RTP is that new media codecs cannot easily be supported because its payload format is not codec agnostic. To support a new codec in RTP, a new payload format standard must be agreed upon. Furthermore, RTP requires out-of-band signaling, and different protocols exist for this, such as the Real Time Streaming Protocol (RTSP) [145] and the Session Initiation Protocol (SIP) [141]. In addition to this, protocols based on datagram streaming are in general afflicted with several major problems:

- Packet-level control means that the implementation becomes very complicated, having to deal with flow and congestion control, packet loss, out-of-order delivery, etc.
- Firewalls and network address translation (NAT) routers frequently cause problems with datagram transport protocols. This even applies to UDP, the most common datagram-based transport protocol. In access logs from the streaming service of VG Nett [43] (the largest online news service in Norway), the failure rate of UDP streaming attempts was observed [59] to be 66 %.
- The cost of the infrastructure becomes significantly higher because content delivery networks (CDNs) require specialized solutions for caching and load balancing (almost all deployed infrastructure optimizations target HTTP because of its massive popularity [117]).

Because of these problems, most of the industry adopted progressive download streaming using HTTP, the second evolutionary step in figure 2.1. While this is actually a step back compared to datagram streaming in terms of potential features, its simplicity has made it the streaming protocol most commonly used today (e.g., by YouTube). With this approach, the client simply downloads a media stream as a file in a normal media container format such as MP4 [97], and plays back the video while it is downloading. There are several benefits to this simple approach: The implementation is straightforward, it can pass through almost any firewall thanks to HTTP's universal support, all CDNs support it, and it can automatically take advantage of transparent web caching to improve performance. The downsides to progressive streaming compared to datagram protocols are that playout interruptions are more likely to occur, a significantly larger buffer is required (limiting progressive streaming's suitability for real-time communication) and multicast is not an option. However the inability to use multicasting is no longer considered a big loss, since there is no widely available multicast infrastructure. Consequently, one

of the biggest arguments for datagram protocols for non-interactive streaming is now mostly irrelevant.

A lot of research has been done on reducing latency when using reliable protocols such as TCP, and several papers [59, 85, 86] show that high latency is not inherent in TCP, but results from throughput-optimized TCP implementations. Regardless, latency performance is not particularly interesting within the scope of this dissertation, as we focus on one-way streaming for VOD and Live content services, where throughput is by far the most important property. Wang, Kurose, Shenoy, and Towsley show [156] that TCP performs well in this regard, offering good streaming performance when the achievable throughput is twice the media bitrate. TCP throughput will suffer in environments where packet loss is caused by high bit error rates [57], but because of the importance of good TCP performance, modern wireless networks such as 3G have techniques for working around this limitation, such as adaptive signal modulation schemes to reduce bit error rates [164], and active queue management to reduce latency [54].

A limitation with most implementations of traditional streaming protocols is that they rarely supported dynamic bitrate adaptation, preventing them from effectively compensating for significant variations in bandwidth, which is a major problem in mobile wireless networks, as will be shown in chapter 3. This problem lead to the development of *adaptive bitrate streaming*, which made bitrate adaptation a central part of the streaming protocol specification. The following section describes various approaches to bitrate adaptive streaming.

2.2 Adaptive Bitrate Streaming

The general idea with adaptive bitrate streaming is that the bitrate (and consequently the quality) should be allowed to change according to currently available resources (on a reasonable timescale). What is meant by "available resources" in this context is usually network bandwidth, but other variables could also be taken into account. Examples of such variables include CPU load [111], battery capacity [48], and screen size [161].

Bitrate selection is usually controlled by the client, but there are also server-driven systems offered by companies such as QuavLive [32], StreamOcean [37], and Akamai [71]. Server-side adaptation is mostly used by advanced servers to provide adaptive streaming to older media players, while client-side adaptation is by far the most popular in recent systems. The reason why it is better to let the client control the bitrate adaptation is that all the information that is relevant when choosing which quality to use, e.g., network conditions, screen size, remaining battery, and CPU load, is available to the client, not the server. Server-side adaptation logic can of course get this information from the client, but periodically sending messages about current network conditions introduces delay in the adaptation scheme,

which can lead to buffer underruns. On the other hand, an argument for server-side adaptation logic is congestion control (a busy server might want to restrict quality levels to reduce its load), but this can easily be done in combination with client-side quality adaptation.

One way to facilitate quality adaptation is using scalable media coding formats. Examples of such formats include Multiple Description Coding (MDC) [87], Scalable Video Coding (SVC) [146], the SP/SI-frame extension [104] to the H.264/AVC video coding format [96, 160], and scalable MPEG (SPEG) [110].

MDC uses a coding technique where a single media stream is fragmented into substreams referred to as "descriptions". An arbitrary subset of descriptions can be used to decode the stream, but the quality depends on the number of descriptions used. MDC has high fault tolerance, but also significant overhead, especially at the network layer [82].

SVC is comparable to MDC, but uses layered coding where each layer N can only be decoded if its subordinate layer $N - 1$ was also decoded. Thus, nothing can be decoded without the lowest layer (the *base layer*), and the more layers that are available, the higher the quality will be. [67] presents a performance comparison of layered and multiple description coding, and concludes that MDC is superior to SVC both in terms of compression ratio and robustness.

The SP/SI-frame extension to H.264 introduces two new picture types, SP and SI, which are the "switching" variants of the standard P- (temporal prediction coding, i.e., describing an image by how it is different from previous images) and I-frames (intra coding, i.e., a stand-alone decodable image) used in H.264 [160]. Any standard H.264 stream already has valid switching positions, as decoders can always start with a clean slate at instantaneous decoder refresh (IDR) frames [160], which are simply I-frames that serve as barriers across which no temporal prediction references are allowed. IDR-frames are usually used as random access positions for seeking, but because these frames cannot exploit temporal redundancy, their coding efficiency is poor, and thus, they are used sparingly (typical IDR-frame intervals are 2–10 seconds [126]).

The purpose of SP/SI-frames is to reduce the bandwidth cost of stream switching points, so that there can be more places in the stream where stream switching is allowed. With SP/SI-enabled H.264 streams, IDR-frames are no longer the only valid switching points; now there is a new frame type, SP, that also fills this role. SP-frames utilize temporal prediction, and can be inserted in place of P-frames to get more places where streaming switching can be done. Due to SP-frames' support for temporal prediction, their coding efficiency is far higher than that of IDR-frames, and can even approach the coding efficiency of regular P-frames [104].

When switching between two SP-frames in different streams, the media player requests a *secondary* switch frame that serves as a "bridge" between the two SP-frames. If temporal predictions across the two streams make sense (e.g., if the two

streams represent the same content in different qualities), the secondary switch frame is also of the SP type (i.e., uses temporal prediction to increase coding efficiency). If the switch is between completely different streams, where cross-stream prediction makes no sense, the secondary switch frame is of the SI type (i.e., no redundancy to exploit across the two different streams). Because these secondary switching frames need to *perfectly* reproduce the reference frame that is expected in the target stream, they are quite large compared to normal P- and I-frames (usually twice as many bits as their non-switching counterparts [147]), but their cost is *only* incurred when a switch actually does occur, *not* for every potential switch point, so coding efficiency is improved in normal use-cases when using SP- instead of only IDR-frames for streaming switching points (Setton and Girod observe [147] an improvement of 40 % with their encoding parameters).

SPEG describes a way to extend currently available compression formats with priority dropping. This means that when the streaming server is notified of an impending buffer underrun, it can reduce the media stream's bitrate by dropping the least important data first. This provides a far more graceful degradation in quality than random dropping of data.

A problem with most codec-based approaches to adaptive streaming is that they have virtually no support in the device market, where hardware accelerated decoding is necessary to limit power consumption, and to reduce unit cost by doing the computationally expensive decoding operation with a cheap dedicated decoder chip instead of a powerful and expensive general CPU. Therefore, another bitrate adaptation mechanism based on traditional codecs such as H.264 and the ubiquitous HTTP protocol has achieved far greater popularity. This technology is the final evolutionary step in figure 2.1, and will from here on be referred to as *adaptive HTTP streaming*. Adaptive HTTP streaming solutions are offered by companies such as Move Networks [121], Microsoft [167], Apple [127], Adobe [51], and Netview [7], and is described in more detail in the following section.

2.3 Adaptive HTTP Streaming

Adaptive HTTP streaming and progressive download streaming are similar in many respects, but in the former, a stream is split into a sequence of file segments which are downloaded individually, instead of performing one large file download per stream. Possibly the earliest mention of this type of streaming is a patent that was filed in 1999 [64].

Each segment is typically 2–10 seconds of the stream [49]. The segment data can be either a multiplexing container format that mixes data from several tracks (audio, video, subtitles, etc.), or it can contain data from just a single track, requiring the receiver to download and process segments from several tracks in parallel.

The video track is usually available in multiple different bitrates, each representing a different quality level. The quality can only change on segment boundaries, so the adaptation granularity is the same as the segmentation granularity, usually 2–10 seconds. Although high granularity switching is desirable, 2–10 seconds is acceptable, because the media player will – most of the time – have more than one segment in its input buffer. Hence, despite the delayed switch to a lower bitrate stream, it can usually be done in time to avoid a buffer underrun. Note that Live streams tend to use shorter segments than VOD streams, because they have less buffered data, and thus need finer granularity switching.

To allow seamless quality switching, every frame in a segment must be encoded without any references to neighboring segments. Note that, because each segment can be considered a stand-alone video clip, playback can start at any segment in the stream. Thus, the segment granularity is also the seek granularity.

Downloading a segment is exactly the same as downloading any other file using HTTP. Standard HTTP GET requests are sent for every segment, and the URLs for these requests contain information such as the timestamp of the first video frame in the segment (or a sequence number), track descriptors, language codes, bitrate values, and anything else that is needed to uniquely identify the segment to be downloaded.

An adaptive HTTP stream normally consists of hundreds or thousands of different segments (typically a segment for every 2–10 seconds, for every available quality level of every track). Each segment is separately downloadable using its own URL but the playout of a concatenated sequence of segments should be seamless. To reduce the entire stream to a single URL, most adaptive formats use a *manifest file* to describe the stream's structure. The manifest file includes information such as:

- General stream meta information (e.g., total stream duration, encryption information, if it is VOD or Live, etc.)
- Which types of streams are available (e.g., audio, video, subtitles, etc.)
- A segment index for each stream, listing URLs for each media segment, and information about the segments' durations and start times.
- Which quality levels are available for each stream. Here one will find information about codec types and encoding parameters such as resolution, frame rate, sample rate, number of audio channels, etc.
- Information about alternate renderings (e.g., different languages for audio and subtitle tracks, different camera angles for video tracks, etc.)

Thus, a media player needs only the URL to the manifest file to start playing video, because all the segment URLs will be known after it has downloaded and parsed the manifest. This workflow is illustrated in figure 2.2.

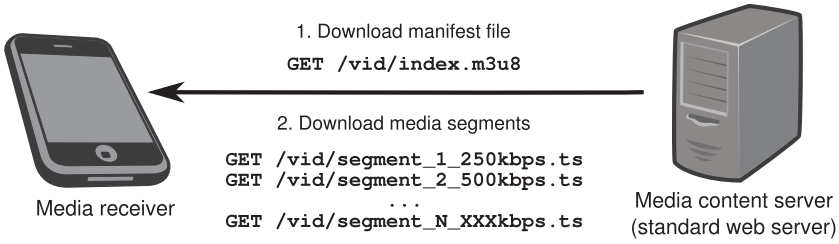


Figure 2.2: The workflow in most adaptive HTTP streaming systems.

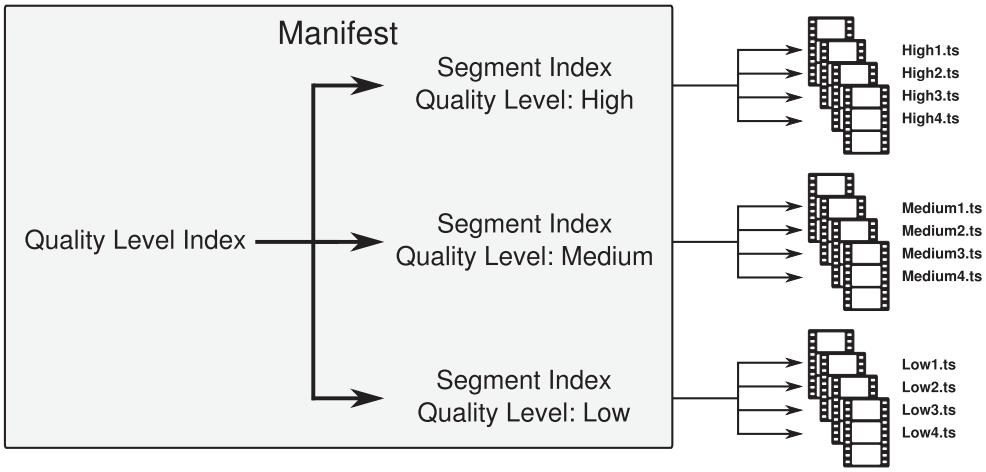


Figure 2.3: A typical layout of a manifest file in adaptive HTTP streaming. This illustration shows a stream with three quality levels (low, medium, high), where each quality level has four media segments. Each segment index describes that quality level's segments, and how to download them (filenames and URLs).

The most interesting parts of the manifest, from an adaptive streaming perspective, are the quality level and segment indexes, as this is what enables quality adaptation and the actual downloading of the media segments. A high-level view of the quality level and segment indexes in a typical manifest file is shown in figure 2.3, and a graphical representation of the quality as a function of time is illustrated in figure 2.4. Both figures use as an example a short stream with three quality levels and four segments. In figure 2.4 the first two segments are played in the lowest quality level, the third segment is played in the medium level, while the fourth segment is played in the highest level. Because adaptive HTTP streaming is a pull-based approach, the receiver is in charge of quality adaptation, which is beneficial to the responsiveness of the bitrate adaptation, as almost all information relevant to the process is immediately available to the receiver, not the server.

The reason HTTP has become the most common transport protocol for adaptive streaming is that it inherits all the benefits of progressive downloading streaming mentioned in section 2.1, while at the same time offering a solution for the most

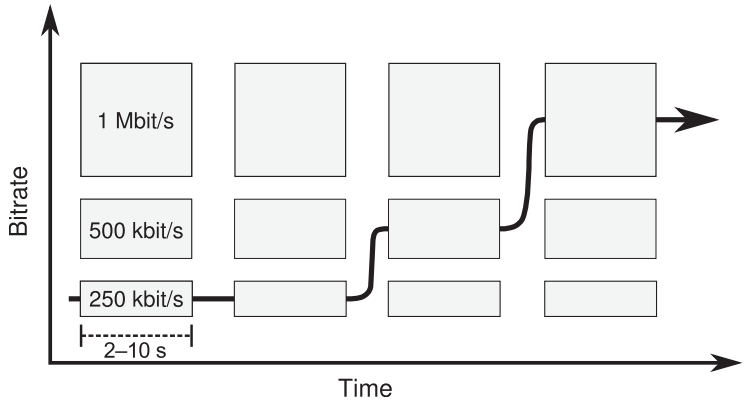


Figure 2.4: The structure of an adaptive HTTP stream. A row of boxes make up a single quality level of the entire stream, and a single box represents a segment of the stream (usually somewhere between 2–10 seconds). Segments in the same column represent exactly the same content, but in different encoding bitrates (qualities). The red arrow represents the video playout, indicating which quality was used for each segment in the streaming session.

significant problem with streaming over HTTP: fluctuating bandwidth. Being able to switch seamlessly to a stream with a lower bitrate whenever the buffer fullness is too low makes HTTP much more usable for real-time streaming, especially on mobile devices. Because packet loss is frequent when streaming to mobile devices, and because packet retransmission is expensive, traditional protocols for mobile video streaming were based on UDP and allowed packet loss to happen. Redundancy and robustness in the video encoding, often in combination with forward error correction, was used to minimize the negative effect of packet loss. After adaptive bitrate video streaming became commonplace, the complexity of UDP-based protocols with lots of redundancy for packet loss became less attractive. TCP could now be used, since the bitrate could be lowered according to the network capacity. Because of the simplicity and ubiquity of adaptive HTTP streaming in fixed networks, it seems likely that this is also the future of mobile video streaming.

Several adaptive HTTP streaming formats are currently available, most notably Apple's *HTTP Live Streaming (HLS)* [127], Microsoft's *Smooth Streaming* [167], Adobe's *HTTP Dynamic Streaming (HDS)* [51], and the ISO/MPEG standard *Dynamic Adaptive Streaming over HTTP (DASH)* [98]. Even though HLS, Smooth and HDS were created by private companies, their specifications are open, and several other companies develop server and client implementations of the different standards.

The standards share the properties described in this section, but differ in manifest syntax, segment URL conventions and media container formats. However, the biggest difference in performance between different systems comes from the clients' quality adaptation strategies, not from which streaming standard is used. Chap-

ter 4 compares the quality selection algorithms (which are agnostic to the adaptive streaming formats) of different media players under challenging streaming scenarios in mobile wireless networks.

2.4 Applications of Adaptive Bitrate HTTP Streaming

The previous section showed that many commercial implementations of adaptive HTTP streaming are available. The following subsections list some of the currently available services that use these products, and goes on to describe future services and features that the technology enables.

2.4.1 Current Services

Adaptive HTTP streaming was designed to be an improvement over progressive download streaming over HTTP. The goal was to use bitrate adaptation to make it more robust against fluctuations in throughput. Thus, it is not surprising that most services using adaptive HTTP streaming today offer just the basic video streaming functionality. They support VOD and Live streaming, often with alternative language tracks, but this is just the functionality viewers have come to expect from any digital video service. Almost all new online video streaming services are based on adaptive HTTP streaming, and the following list is just a very small subset of what is available as of 2012:

- Netflix [27], an American provider of on-demand Internet streaming media that offers an extensive library of content to over 20 million streaming subscribers (as of 2012) [123].

- TV 2 [41], the largest commercial television station in Norway, uses adaptive HTTP streaming for its online video platform, which includes Premier League football. TV 2 does not publish its subscription numbers, but it is estimated [116] that they had around 60 000 subscribers at the end of 2010 (a 30 % increase since 2009).

- BBC, the largest broadcaster in the world, uses adaptive HTTP streaming in its iPlayer service [12]. BBC reports [34] 1.94 billion TV and radio program requests across all platforms in 2011.

- ESPN, an American television network focusing on sports-related programming, uses adaptive streaming in its ESPN Player [17]. ESPN does not provide much data on its viewer numbers, but [109] reports 330 000 unique viewers for ESPN's live streamed NBA Finals in 2012.

- Comoyo [15], a consumer content portal for the Nordic region launched by Telenor, offering movies, sports events, and other TV content. The service is still new, so no data on subscribers or viewers is available at the time of writing.

- Major sports events such as the Olympics [25] (106 million video requests reported [29] by BBC alone for the 2012 Olympics), Super Bowl [38] (more than 2.1 million viewers [75]), and the FIFA World Cup [19] (30 million unique viewers reported [2] by Conviva alone) are all available through adaptive HTTP streaming services.

This list shows that adaptive HTTP streaming is used by big businesses to stream premium content that draws huge numbers of viewers, indicating that the technology is both scalable and robust. Although the services listed use adaptive bitrate HTTP streaming to provide traditional streaming services, the technology also facilitates more advanced features, which will be briefly discussed in the next subsection.

2.4.2 Future Services

As mentioned in section 2.3, a property that is common to all adaptive HTTP streaming systems is that each segment is stand-alone decodable. Without this property, it would not be possible to switch between different quality levels in the middle of a stream, because dependencies between segments would mean that the segment in the new quality level would depend on preceeding segments in that quality level. The quality level that the media player is switching *from* is a completely different encoding from the one it is switching *to*, so switching quality levels in the middle of a stream would break decoding dependencies. Similarly, seeking to random positions in the stream would not be possible without stand-alone decodable segments. Workarounds with byte offsets to true random access points that do not align with segment boundaries could be added to the stream manifest, but this would increase manifest overhead and complexity of the implementation. Thus, all adaptive streaming formats make it simple by requiring the first frame in each segment to be a true random access point (an IDR-frame in H.264 terminology).

An interesting benefit of stand-alone decodable segments is that, not only can they be played in any order, but segments from completely different streams can also be concatenated and played seamlessly (provided that all the segments use the same codec, of course – e.g., a decoder could not successfully decode a concatenated sequence of H.264 and VC-1 [115] segments). It follows from this that video editing is possible through recombinations of video segments from various sources, without computationally expensive re-encoding of video. In other words, creating a custom video cut is nothing more than a custom playlist with segments from different

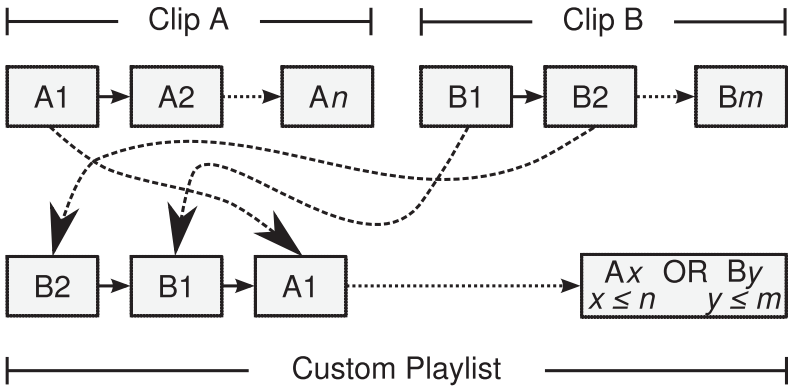


Figure 2.5: Stand-alone decodable segments can be recombined from different stream sources (*A* and *B* in this example), in any order, in effect enabling lightweight video-editing.

streams, in any order (see figure 2.5). However, note that the video editing granularity is equal to the segment durations (2–10 seconds) in our implementation. Frame accurate editing is possible in theory [84], but not practical in the general case. E.g., consider a stream with 10 second segments and video with 30 frames per second, i.e. 300 frames per segment. In adaptive HTTP content, each segment has typically only one IDR-frame (a frame encoded for random access). Thus, to be able to access the last frame in a 300-frame segment, *all* 299 frames before it must be downloaded and decoded. These 299 frames would only be used to put the decoder in the proper state for the last frame, they are not displayed. Hence, the cost of direct access to the last frame, in terms of download bandwidth and processing power required to decode it, becomes roughly 300 times higher than it would have been if it were used normally. Using only the last frame in a segment is the worst case scenario, but the same problem applies in the general case: There is always a high price to pay for random access to frames that are not encoded for random access. This cost would incur, not once when the playlist is generated, but *every time* the playlist is used. As such, we believe the cost outweighs the benefits of frame accurate editing. Nevertheless, segment-level video editing unlocks powerful features not traditionally seen in video services targeted to end-users.

Take, for example, a video search engine. Traditional video search engines such as YouTube [46] return results that match the stream as a whole, which is far from ideal if the user searched for a specific event that occurs somewhere within a long video clip. Empowered with on-the-fly lightweight video editing through advanced segment playlists, the search engine can do indexing, annotation, and tagging on individual segments, and on a search query, return exactly the segments that match the query. This makes the video search engine far more precise and enables applications that otherwise be much less useful, such as a sports event video search engine.

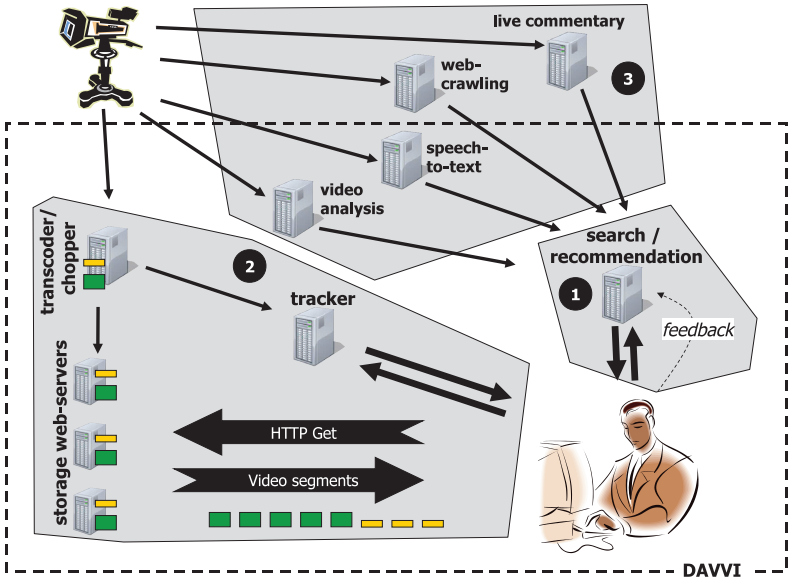


Figure 2.6: The DAVVI architecture and main components.

We created the DAVVI system [101, 100], a prototype for a next generation multimedia platform, precisely to demonstrate this point. The architecture of the DAVVI system is illustrated in figure 2.6. It provides a search interface to voluminous soccer video archives where annotation and tags are applied to the segments to which they refer, instead of the entire stream (which may be several hours long). Queried events are extracted, concatenated from several different videos, into one continuous video playout. For example, a query for all the goals made by a specific player in the last year produces a highlights reel containing just the events matching that query. Traditional video search services such as ESPN Search [18], VG Live [44] and TV2 Sumo [42] only allow the user to search for games and a few main game events, such as goals. The main drawback with these systems is that search results cannot produce a customized, on-the-fly generated personalized video playout. The DAVVI search engine can, at a user's request, automatically sequence out portions of longer videos, or aggregate parts from multiple videos, to provide a single, personalized video stream on-the-fly. Because the search results only return the segments that are relevant, DAVVI is much more precise than traditional video search engines. Additionally, users can also – through the web interface – do lightweight video editing on the content available to them, and, in theory, share their customized playouts with other users (this social aspect is not yet implemented). In summary, the DAVVI system takes advantage of lightweight segment-based video editing to provide a personalized topic-based user experience that blurs the distinction between content producers and consumers.

Note that basing such an application on adaptive HTTP streaming with stand-alone media segments is essential. One could envision using SP/SI-frames (described in section 2.2) instead of stand-alone decodable segments, but it would be very inefficient in the general case, since customized playlists would frequently require SI-frames (as segments from different sources are joined together), which usually have less than half the coding efficiency of IDR-frames [147]. Even worse, there would have to be an SI-frame for every possible combination of segments (since they serve as bridges between two SP-frames, they are not stand-alone like IDR-frames). Thus, the number of possible SI-frames is proportional to the square of the number of segments in the media database, which can easily be in the billions for a large service like YouTube. It is obviously impossible to store something in the order of 10^{18} switching frames, so the only solution would be to encode these switching frames as part of the playlist generation, which destroys the encoding-free aspect to video editing.

Another example of a service that utilizes indexed video segments is *vESP (video-Enabled Enterprise Search Platform)* [90]. Here, the content database consists of a set of presentation slides and corresponding video clips of a person presenting them. A user can select slides from different presentations to generate a customized slide deck, and the video form of this presentation is automatically generated by concatenating clips corresponding to each slide in the customized slide deck. Similar services are available, such as Altus vSearch [55] and FXPAL's TalkMiner [20], but again they lack the ability to present on-the-fly generated video for a selected set of slides.

Sadlier and O'Connor [142] propose that the metadata that facilitates services like those mentioned here could be generated by automatic video analysis, and outline an approach to automatic event detection. Although soccer is the perhaps most widely investigated topic [76, 153, 166], similar approaches exist for other sports like tennis [70], basketball [168], baseball [66], rugby [23], American football [113], and Formula 1 [154]. Such systems can be used in our metadata and annotation operations. However, their reported recall and accuracy when used alone is insufficient in the context of our target application areas where both high accuracy and low recall are required, so having professionals perform semantic annotation is often necessary. However, another option is user-generated annotation, tagging, and playlist sharing. It follows that future video search engines can benefit from a closer integration of video delivery systems, search and recommendation, and social networking.

2.5 Summary

Adaptive HTTP streaming is relatively simple to implement, benefits from the ubiquity of the HTTP protocol, and enables powerful functionality beyond traditional

streaming through its use of stand-alone decodable media segments. Even the most basic implementations work well on fixed networks, and the technology has been extremely successful in commercial video streaming services, having taken over almost the entire market in just a few years. Also, adaptive HTTP streaming makes it easy to create personalized video playouts, which has a wide range of applications, such as better video search engines or online video editing.

However, it is a different matter how well adaptive HTTP streaming performs on mobile wireless networks, where TCP-based traffic often suffers from poor performance due to frequent packet loss and large variations in latency. Streaming to mobile devices is an increasingly important scenario, as it is only recently that handheld devices powerful enough to play high quality video became commonplace, and affordable high-speed mobile Internet connections were also not available until recently.

To better understand the challenges encountered when streaming in mobile networks, the next chapter presents a study of bandwidth, packet loss and latency in Telenor's 3G/HSDPA network in Oslo, Norway. Although it is TCP's performance we are most interested in, we also study UDP traffic under the same conditions, because observations on the packet level will help explain the behavior of the TCP protocol.

Chapter 3

Performance Characteristics of 3G/HSDPA Networks

Adaptive HTTP streaming uses standard video codecs and transport protocols to provide adaptive bitrate streaming. This allows for a more robust streaming system, where playout interruptions due to buffer underruns can be greatly reduced. The technology has enjoyed great commercial success, and because it makes customized video playouts easy and inexpensive to generate, it enables features not seen in traditional video streaming services.

It is not clear, however, how efficient the technology will be in mobile wireless networks, where the bandwidth fluctuations will be much more frequent and drastic than in fixed networks. The performance of adaptive HTTP streaming in mobile wireless networks will be studied thoroughly in this chapter, but first we present a low-level study of packet reception characteristics in Telenor's 3G/HSDPA network in Oslo, Norway.

3.1 Related Work

Many studies have been performed on performance in 3G networks. Holma and Reunanen present [91] measurement results for an early implementation of HSDPA, both from the laboratory and field measurements during 2005. Derksen, Jansen, Maijala, and Westerberg present [72] results from HSDPA measurements made in a live, commercial network supplied by Ericsson, as well as future enhancements to the technology that will further improve performance.

Jurvansuu, Prokkola, Hanski, and Perälä evaluate [103] live HSDPA operational network performance from the end-user perspective, looking at both TCP and UDP performance, and focusing on Voice over IP and web applications. Prokkola, Perala, Hanski, and Piri extend this work [132] with uplink and mobility measurements in a live High-Speed Uplink Packet Access (HSUPA) network.

Isotalo and Lempiäinen study [99] performance of HSDPA in an indoor environment, and provide guidelines on HSDPA coverage and capacity planning in different antenna configurations consisting of pico cells and distributed antenna systems.

In [163], Xu, Gerber, Mao, and Pang used the predictability of human mobility patterns to develop an algorithm for accurately determining the geographical location of users. They use this algorithm to map IP-level flow records to fine-grained geographic regions, envisioning this technology to be an important tool for operators of 3G networks for the purpose of performance monitoring, network maintenance and anomaly detection.

Deshpande, Hou and Das compare [73] a nation-wide 3G network and a metro-scale WiFi network operated by a commercial ISP from the perspective of vehicular network access. They find that 3G offers somewhat lower throughput than WiFi, but a far more stable connection for mobile access. The speed of the vehicle was found to have little effect on a 3G client, but a WiFi client experienced a large drop in throughput above 20 km/h, as the WiFi handover mechanism is not optimized for speed. However, when WiFi is available, it is very likely that it outperforms 3G, so the paper concludes that a hybrid solution that aggregates 3G and WiFi would be most successful (we develop a multi-link media player in chapter 6, and confirm this conclusion empirically).

Botta, Pescape, Ventre, Biersack, and Rugel study [60] packet traces from real users in an operational 3G environment. They focus on heavy users (those that transfer large amounts of data), and try to determine the causes of varying throughput. They conclude that the most common causes of high packet loss and low throughput is either congestion in the cellular core network, or congestion at the access induced by user behavior.

Weber, Guerra, Sawhney, Golovanevsky, and Kang analyze [159] video streaming performance in live Universal Mobile Telecommunications System (UMTS) networks. They focus on the performance of MPEG-4 streaming over RTP (using UDP as the transport protocol), and compare performance in different distances from the radio tower, and performance with a mobile receiver (average speed was 34 km/h). They conclude that the audio and video streaming performance was impacted most when the network switched down the radio bearer for a more robust signal at the expense of throughput. They used non-adaptive streaming, so when the bandwidth dropped below the media bitrate, buffer underruns interrupted playback frequently.

However, none of these tests explore the performance as a function of geographical location while travelling, which prompted the study presented in this chapter. This study was also conducted with one-way video streaming in mind, meaning that we focused more on throughput than latency. The following section describes our UDP-based measurements, which were primarily intended to expose packet-level behavior in 3G networks under different conditions.

3.2 Characteristics of 3G/HSDPA Networks in a Fixed Rate UDP Streaming Scenario

When we started this project, the high-speed mobile network with the most coverage in Norway was Telenor's 3G/HSDPA network. HSDPA is a communications protocol that improves downlink performance in mobile networks based on UMTS. The theoretical maximum download rate in Telenor's 3G network was 3.6 Mbit/s at the time we performed the UDP-based experiments described in this section.

The initial 3G/HSDPA measurements were performed using a custom made analysis tool that transferred data in a fixed (but configurable) rate using UDP packets. The purpose of this low-level approach was to measure more network characteristics than we could with a TCP-based protocol. We wanted to study throughput, packet loss patterns, transmission errors, latency, and jitter. To be able to test this, each UDP packet contained three things:

1. A sequence number, enabling the receiver to precisely detect packet loss.
2. A timestamp showing when the package left the receiver. The sender and the receiver had their clocks synchronized using the Network Time Protocol (NTP) [8], enabling the receiver to measure the one-way delay and jitter.
3. A payload to make the packet size exactly 1500 bytes (equal to the network's maximum transmission unit (MTU)). The bit pattern was fixed, enabling the receiver to check for transmission errors.

Because download performance is much more important than upload performance for a one-way video streaming scenario, all tests measured the download performance of the 3G/HSDPA connection. The sender was a dedicated server with a 100 Mbit/s Ethernet connection, and the receiver was a laptop with a 3G/HSDPA connection. The server was only four hops away from the Norwegian Internet eXchange (NIX), with on average less than 2 ms packet round-trip time. Almost all of the delay between the sender and receiver was due to the mobile network.

In each test, packets were sent at a fixed rate, and the packet reception rate was logged. When the UDP packet transmission rate exceeded the available bandwidth in the network, the result would be a reduced reception rate. In other words, a straight line at 100 % means a perfect transmission where no packets were lost. Every dip in the curve represents loss of data.

3.2.1 One Receiver Utilizing Maximum Bandwidth

To study how the HSDPA network performed with one single user consuming all the bandwidth, the sender was configured to send packets at a rate that exceeded the expected maximum throughput by about 25 %. We performed multiple tests (using

the same tool) using different bitrates, and determined that the expected throughput was around 2.2–2.3 Mbit/s. Hence, the send rate was set to 2.8 Mbit/s for this test, making sure that we consume all available bandwidth. The receiver's packet reception log was used to plot the reception rate as a function of time (figure 3.1 shows a typical result, after having done multiple runs in a location very close to the base station). The overall effective reception rate when accounting for packet loss was 2.25 Mbit/s.

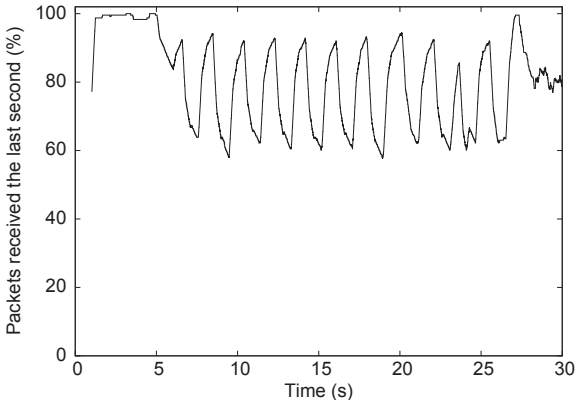


Figure 3.1: UDP packet reception rate in a 3.6 Mbit/s 3G network. A single stationary user is streaming at 2.8 Mbit/s. A straight line at 100 % would mean that no packets were lost. but since the effective bandwidth is only 2.25 Mbit/s, packets are dropped in the wireless network causing a saw toothed loss pattern.

The throughput observed here is significantly lower than the numbers reported by Ericsson [72] in a similar network, even though the location was optimal and no other users were consuming the cell's bandwidth. The explanation may be that Telenor's network had reserved some of the bandwidth for voice traffic.

The saw-toothed packet drop pattern starting about five seconds into the test indicates that the base station has a large buffer to avoid losing data due to short-lived peaks in the transmission rate. If the buffer does not overflow, packet loss can be avoided at the cost of variable delay (jitter) depending on the fullness of the buffer. If the buffer overflows, large numbers of packets are dropped.

3.2.2 Multiple Receivers Utilizing Maximum Bandwidth

When a mobile wireless network has a stated maximum bandwidth, what is meant is the total bandwidth *per cell*, not the total *per receiver*. In other words, multiple users streaming in the same cell will have to share the bandwidth. The fairness of the sharing is controlled both by the transport protocol (depending on how it responds to packet loss) and the base station. With the UDP protocol used in the tests

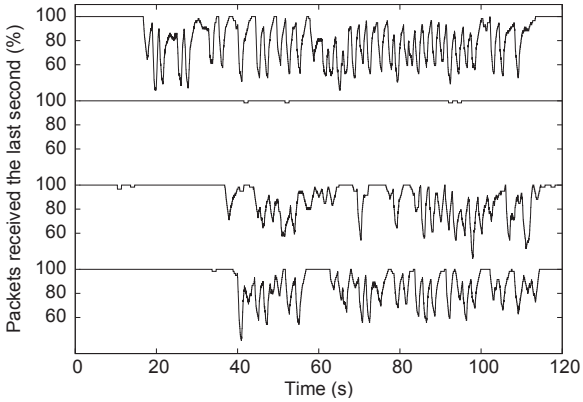


Figure 3.2: UDP packet reception rate in a 3.6 Mbit/s 3G network. Four stationary users at the same location are streaming at 0.7 Mbit/s.

described in this section (a fixed data rate that ignores packet loss), the fairness of the bandwidth sharing will be determined only by the base station.

Observing in previous tests that the maximum data rate was around 2.3 Mbit/s, and using four simultaneous receivers in the same location, the data rate was set to 0.7 Mbit/s per receiver. With a total of 4×0.7 Mbit/s $= 2.8$ Mbit/s, this was guaranteed to overflow the base station's buffers, forcing it to drop packets and expose its bandwidth sharing fairness in a resource constrained scenario.

Results from this test varied from run to run, but packet loss was generally not equally distributed among the four receivers. A typical result is shown in figure 3.2.

The packet loss patterns display the same saw-toothed appearance as the single-receiver test previously showed in figure 3.1, but here, receiver number one (from the top) experiences packet loss about 20 seconds into the test, while for three and four it does not occur before 40 seconds into the test. Receiver number two has almost no packets lost over the entire run. This shows that the base station maintained individual packet queues for each client, and did not give them equal treatment.

The aggregated effective throughput for all four receivers was 2.53 Mbit/s, over 12 % more than was achieved with a single receiver in the same location. Thus, it is clear that cross-traffic interference is not a problem, the total throughput available in a cell is at least as good with multiple receivers as with a single receiver.

3.2.3 Loss Patterns Due to Noise

In the tests described in subsections 3.2.1 and 3.2.2, packet loss was predominately caused by congestion, forcing the base station to drop packets from its queues. But even without congestion, data can be lost due to signal noise.

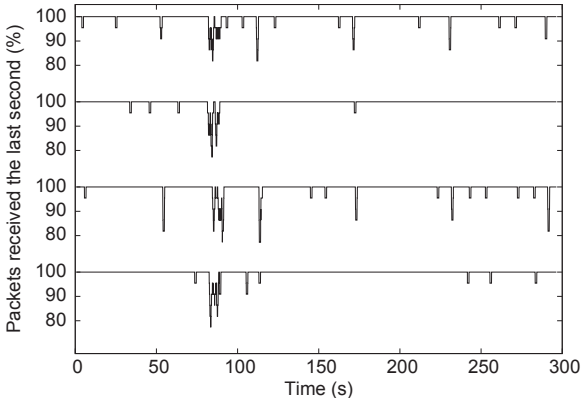


Figure 3.3: UDP packet reception rate in a 3.6 Mbit/s 3G network with observed signal noise spikes. Four stationary users at the same location are streaming at 256 kbit/s (far below the congestion threshold).

To observe this, several locations were tested with low data rates (256 kbit/s per receiver, much too low to fill the base station's buffers) to see if any of them had significant packet loss. Some locations were indeed prone to have sporadic noise spikes, and four simultaneous receivers were used to measure the packet loss caused by this phenomenon. The reason for using four receivers instead of one, was to test test if the noise spikes affect all receivers equally, as expected. The result is shown in figure 3.3. However, note that such erratic loss patterns are rare, and often not reproducible. We mention it here to show that it exists, but is insignificant compared to other causes of packet loss.

3.2.4 Loss Patterns for A Receiver Moving Between Base Stations

Subsections 3.2.1, 3.2.2 and 3.2.3 all describe results for stationary receivers, but it is interesting to also study packet loss in mobile receivers, as this is obviously an important scenario for video streaming to mobile devices.

In addition to possible congestion and noise, a mobile receiver will also experience varying signal strength due to varying geography and distance to the base station, as well as potential issues caused by handover from one base station to another.

To test the effects of varying geography and base station handover, the chosen location was a road in the sparsely populated northern parts of Oslo, near Maridalsvannet. All packet loss in this area should be due to poor base station coverage, because base stations are not densely placed in rural environments. When travelling through city streets, there will almost always be a base station sufficiently close to provide good signal strength. In the countryside, the distance to the base station

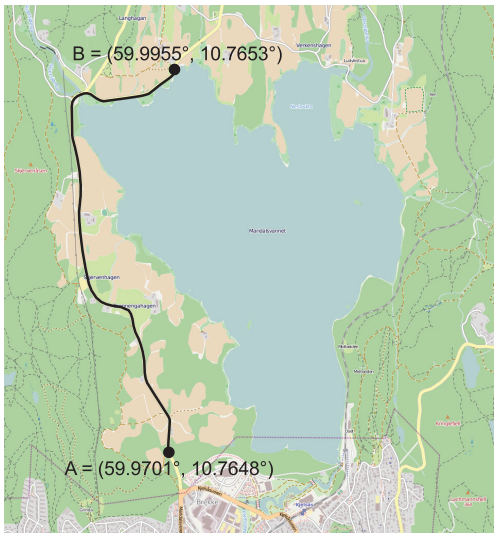
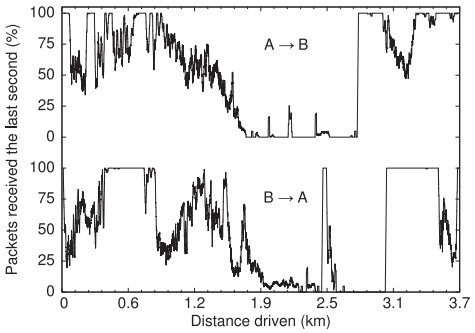
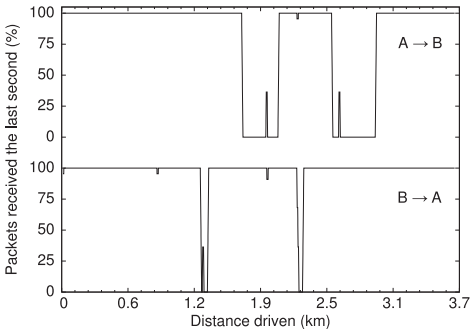


Figure 3.4: Driving route along Maridalsvannet in Oslo when testing packet loss during base station handover. The one-way path length is 3.7 km.



(a) Streaming at 2 Mbit/s (the two graphs show the same route driven in both directions).



(b) Streaming at 256 kbit/s (the two graphs show the same route driven in both directions).

Figure 3.5: Comparing UDP packet reception rate in a 3.6 Mbit/s 3G network where a single receiver streams at 2 Mbit/s (left) and 256 kbit/s (right) while driving at 50 km/h from $A \rightarrow B$ and back again (the route is illustrated in figure 3.4).

will vary a lot, causing the signal strength to fluctuate more strongly. We thought this would be an interesting scenario for testing the base station handover mechanism, as a delayed handover could mean abrupt varitations in signal strength as we are migrated from a distant base station to a close one.

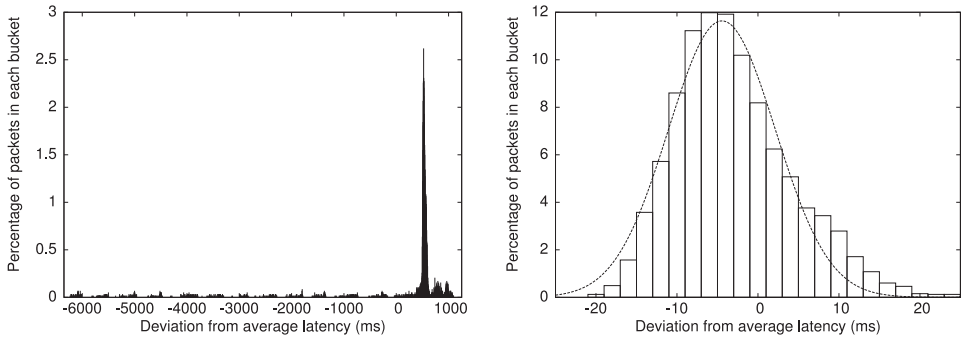
A car driving at a fixed speed of 50 km/h was used as the method of transportation used for this test. The end points A and B of the travel route are shown in figure 3.4.

Both A and B were points where a stationary receiver could receive most packets from an incoming 2 Mbit/s stream (on average 6 % packet loss at point A and 0.1 % at point B). In other words, the network connection was good at the route's endpoints, and the test would measure how the connection varies as the receiver travels between them at 50 km/h.

Figure 3.5(a) shows the packet loss when attempting to stream at 2 Mbit/s. The two graphs in the figure show that the loss patterns look similar even though they represent the same route in opposite directions. It is clear from the graph that the throughput fluctuates dramatically, and that the handover mechanism can be slow to react: About 150 seconds into the tests, the connection is almost dead, and at 220–250 seconds, it instantly bounces back up to an almost perfect signal.

Observing in figure 3.5(a) that the packet loss is much too high to be useful for normal TCP-based network traffic, the same test was done again, but with a 256 kbit/s data rate (down from 2 Mbit/s). This should allow the base station to use a more robust signal modulation that offers lower packet loss at the cost of lower throughput. Figure 3.5(b) shows the result of this test. The lower data rate almost completely eliminated the bandwidth fluctations, but there are still large gaps in the connectivity that last for up to 30 seconds.

3.2.5 Packet Delay and Jitter



(a) Distribution of latency observed by one of four receivers that are each streaming at 700 kbit/s in the same cell (congested network).

(b) Distribution of latency observed by one of four receivers that are each streaming at 256 kbit/s in the same cell (non-congested network).

Figure 3.6: Comparing jitter in a congested mobile network (left) with an uncongested one (right). Each bar in the graphs is a "bucket" of data points, and the y-axis shows the relative number of points in each bucket.

Throughout all tests performed in this section, packet latency (one-way delay) was typically in the range 70–120 ms when the base station was not congested. The variation in latency (jitter) in such a scenario is shown in figure 3.6(b). The figure

has also overlayed a Gaussian function that approximates the observed results. Its standard deviation (σ) is approximately 7 ms. This is a very good result, and tells us that under normal conditions (non-congested base station), the latency is more than good enough for non-interactive video streaming applications.

Latency measurements with other data rates below the congestion threshold were also performed, and demonstrated that the amount of jitter was constant, as long as congestion does not occur.

When congestion *does* occur, the latency distribution looks completely different. The base station's buffers are large and, when full, it can be several seconds between a packet's arrival in the base station's queue and its arrival at the receiver. To illustrate this, figure 3.6(a) shows the latency distribution experienced by the first receiver from the top in figure 3.2. Here, the base station has four users trying to stream at 700 kbit/s, thus overflowing the base station's buffers.

The difference between figures 3.6(a) and 3.6(b) is dramatic. The congested distribution (figure 3.6(a)) has little resemblance to a Gaussian curve, and the average latency is over 6 seconds compared to 0.1 seconds for the non-congested case (figure 3.6(b)). The spread in latency is quite large, which was to be expected as it must grow proportionally with the amount of data in the base station's output queue.

3.3 TCP Throughput in 3G/HSDPA Networks

To measure the TCP throughput in Telenor's 3G/HSDPA network in Oslo, field trials were performed along several popular commute routes in and around Oslo. The software used to perform these tests was Netview's own adaptive HTTP streaming client, where the buffer size was unlimited, ensuring that the player would never idle (the results would still be usable if idling was allowed to happen, but it would result in fewer data points per trip).

All but one of the selected routes use different types of public transportation: metro, bus, ferry, tram, and train where the respective paths are highlighted in figures 3.7(a), 3.8(a), 3.10(a), 3.11(a) and 3.12(a). The final route (figure 3.13(a)) was by car, but because of the length of the trip (approximately 280 kilometers), only one measurement was performed for this route, which is why we only present the observed bandwidth, not an average and the variance.

To perform the field experiments in this study, throughput was logged while downloading video segments from a dedicated high-performance web server at the maximum speed achieved by the TCP protocol. In this context, "maximum speed" means that no throttling was done at the application level, neither by our media client nor the web server hosting the content. Laptops with Global Positioning System (GPS) devices (Haicom HI-204III USB GPS) and 3G mobile Internet devices (Huawei Model E1752 HSPA USB stick) were used to perform the measurements.

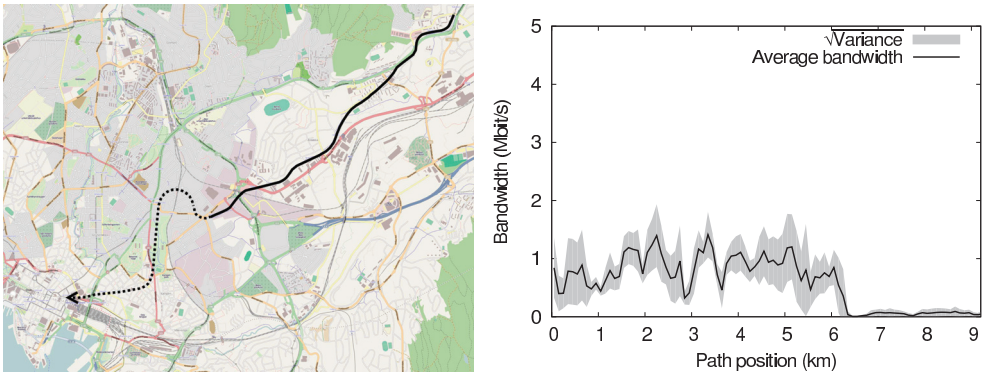
Note that in the event of GPS signal loss, the HI-204III GPS device will (for a short time) extrapolate positions based on previously recorded positions and movement vectors. To avoid getting wrong path distance numbers when generating the graphs showing bandwidth as a function of distance from the start of the path, the position/sample pairs that were clearly in the wrong place were manually corrected. A more sophisticated implementation should be able to do this automatically by querying the GPS about signal strength to identify which samples can be trusted and which can not.

For approximately every second in the test session, a data point was recorded, each containing:

1. Current time, expressed as a Unix timestamp (number of seconds since 1970-01-01 00:00 UTC).
2. Geographical position (latitude and longitude).
3. The number of milliseconds that elapsed since the last data point.
4. The number of bytes received since the last data point.

Bandwidth measurements were performed several times for every route (more than ten times for all of the interesting routes: metro, bus, ferry and tram), where the plots show the average bandwidths as a function of the path position (as traveled distance from the start) with highlighted standard deviation. All data sets presented here (and more) are available for download [140].

3.3.1 Metro railway



(a) Map (the dotted parts indicate that the train is underground).

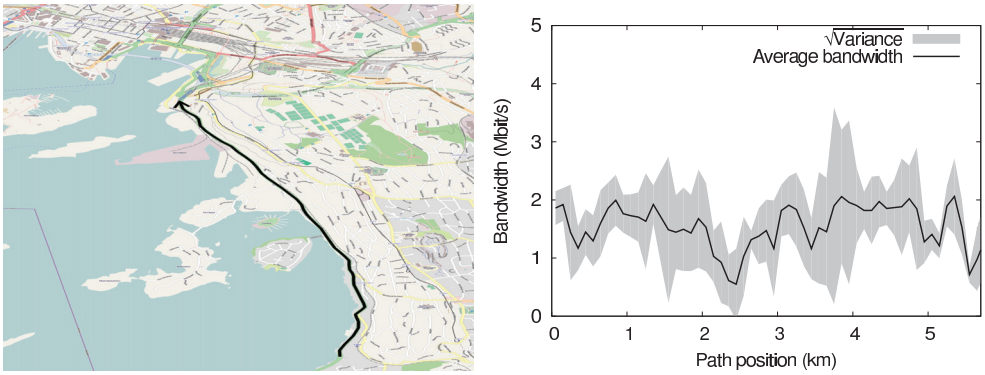
(b) Bandwidth.

Figure 3.7: Metro commute route between Kalbakken and Jernbanetorget.

A popular means of commuting in Oslo is the metro. This is an electric passenger railway, all of whose lines pass through underground tunnels in central Oslo, and above ground outside the center. The underground part of the tested metro commute route is shown with the dotted line in figure 3.7(a). When the train was underground, the signal was very poor, as expected.

Figure 3.7(b) shows the measured bandwidth along the metro path. All the measurements show the same trend, and the signal and bandwidth availability are predictable with only minor variations. The experienced bandwidth is typically around 1 Mbit/s when the train is not underground. When entering the tunnels after approximately 5.5 kilometers, both the Internet connection and the GPS signal are essentially lost (in periods without a GPS signal, the position is estimated based on the metro time table).

3.3.2 Bus

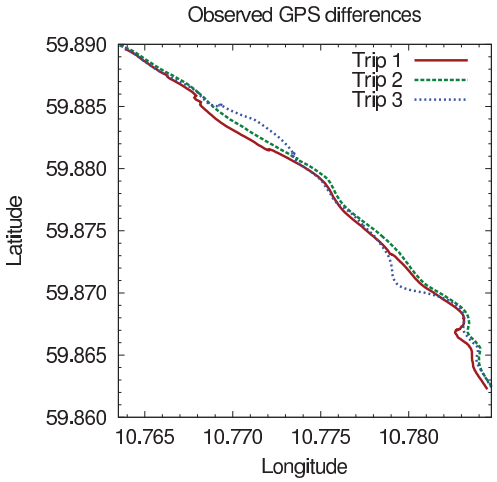


(a) Map.

(b) Bandwidth.

Figure 3.8: Bus commute route along Mosseveien, between Ljan and Oslo Central Station.

Figures 3.8(a) and 3.8(b) show a bus path going into Oslo and the corresponding bandwidth measurements. The average values in figure 3.8(b) vary greatly, but the measurements show that a streaming session of about 1.5 Mbit/s should normally be possible. However, the example also shows one of the challenges of establishing a bandwidth lookup service: The bus route has a steep hill on the east side, which prevents the reception of signals from eastern GPS satellites. The Oslofjord is in the west, leaving few possible sites for UMTS base stations on that side. Consequently, both the GPS and the UMTS signals vary a lot, which again creates uncertainty in estimating where an achieved bandwidth is measured. This may explain the wide standard deviation "belt" in figure 3.8(b). One reason for this is that the GPS device used for this test was unable to get a good signal in some experiments, but connection losses only rarely occured (only one single serious outage, typically last-



(a) Three different measurements along parts of the route.

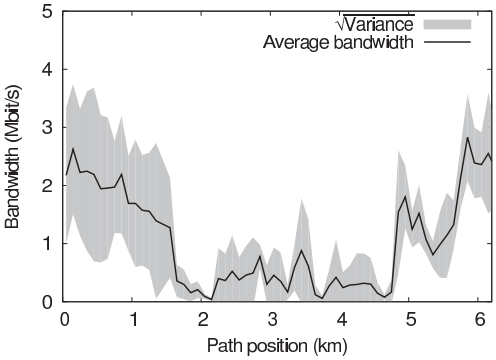
(b) Difference between trip 2 and 3 (in figure 3.9(a)) at 59.8708 North on the map.

Figure 3.9: Observed GPS differences on the bus route.

ing 20–30 seconds, occurring at a predictable location every time). To illustrate this, figure 3.9(a) shows the paths reported by the GPS devices on three trips along the same road. The deviation is probably caused by weak GPS signals and a GPS device which responds by returning extrapolated coordinates; for example, at latitude 59.8708 North (signed decimal degrees), trip 2 and 3 have longitude values of 10.7793 East and 10.7809 East, respectively. This difference translates to about 90 meters, making the bus appear to be driving in the sea (see figure 3.9(b)).

3.3.3 Ferry

(a) Map.



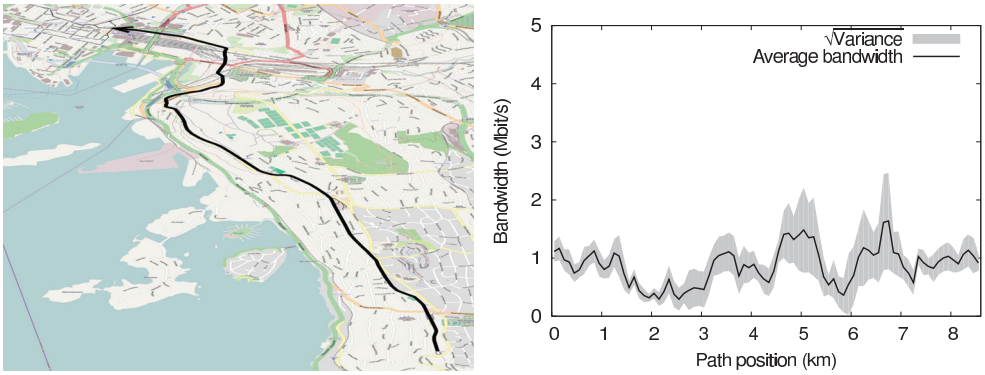
(b) Bandwidth.

Figure 3.10: Ferry commute route between Nesodden and Aker Brygge.

The third test scenario is travelling by boat (figure 3.10(a)). Most commuters from the Nesodden peninsula travel by ferry to Oslo, as traveling by car requires a large detour. With lots of space on board and tables for PCs and devices with large screens, this means of transportation is the one that is best suited for mobile video viewing of high resolution content, but signal conditions far from land can sometimes be problematic. Figure 3.10(b) shows that the available bandwidth varies quite a lot, but the signal is never completely gone while crossing the Oslofjord. As expected, the signal is strongest when the ferry is close to land (Nesodden at the start of the path, and Aker Brygge in downtown Oslo).

3.3.4 Tram

The fourth way of commuting in Oslo using public transportation is by tram. Figure 3.11(a) shows the tested tram path, whose tracks are parallel to but high above the bus route. Figure 3.11(b) presents the measured bandwidth. Along the whole path, acceptable but fluctuating bandwidth was observed. In the first part of the route, a very predictable bandwidth was observed across the different measurements. At the long downslope towards the end of the trip, the measurements vary more.



(a) Map.

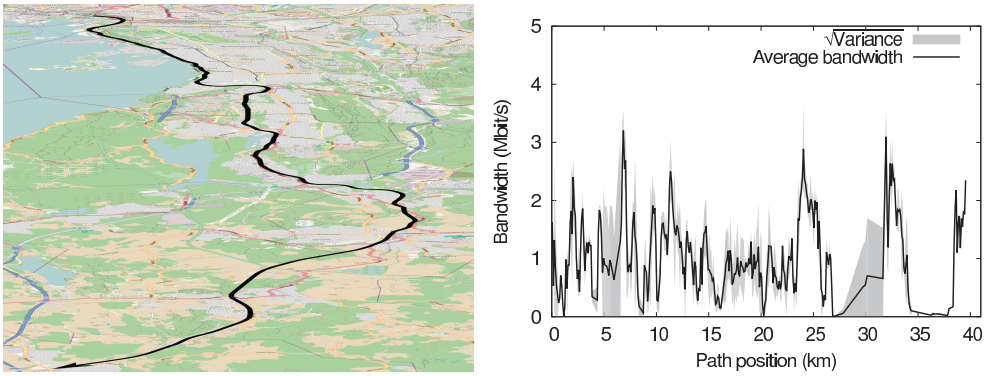
(b) Bandwidth.

Figure 3.11: Tram commute route between Ljabru and Oslo Central Station.

3.3.5 Train

The trains to and from Oslo are frequently used by people traveling longer distances, and in figure 3.12, we show the map and bandwidth plot for the Oslo–Vestby route. In our measurements, the bandwidth varies a lot, sometimes jumping between 3 Mbit/s and almost no connectivity at all. Another problem was that the signal appeared to be strongly affected by where in the train the receiver was located. Most

likely this is because the aluminum alloy used in the train's body creates an environment similar to a Faraday cage [83], blocking both the GPS and the mobile data network signals. Hence, all tests shown in figure 3.12 were done while sitting in a window seat, where the connection was sufficently stable for video streaming.



(a) Map.

(b) Bandwidth.

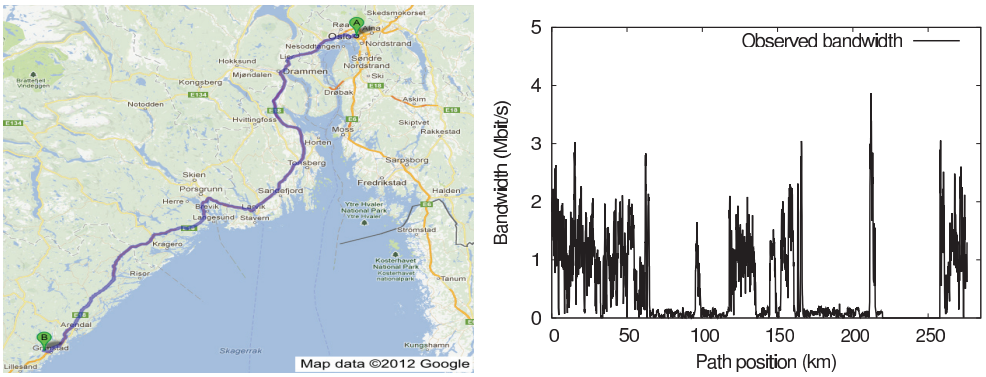
Figure 3.12: Train commute route between Oslo and Vestby.

3.3.6 Car

The route Oslo–Grimstad, shown in figure 3.13(a), is used by everyone driving from Oslo going south on the E18 highway. It is an approximately 280 kilometer drive. Figure 3.13(b) shows the achieved bandwidth, where we observed high peaks over 3 Mbit/s with an average of about 1 Mbit/s. However, as is also seen in the plot, there are several areas (65–118 km, 165–211 km and 216–258 km) where there is almost no connectivity (less than 100 kbit/s on average), certainly far too low to sustain even the lowest quality video streams. The distance between 216 and 258 km was especially bad, as here the network went completely dead. In total, the average bandwidth was insufficient for video streaming for almost 50 % of the trip. Through extreme amounts of buffering, and using a very low quality level, an adaptive streaming client could in theory maintain uninterrupted playback throughout the trip, but the viewer would have no freedom to seek or switch streams, as everything depends on filling a huge buffer in the beginning of the trip (while the network bandwidth is still high enough to both play video, and fill the buffer at the same time). In effect, the user experience would no longer be considered "streaming", being more similar to a "download movie, then play" type of client.

3.4 Performance as a Function of Time of Day

Because 3G networks are highly bandwidth limited at the access compared to fixed networks, it sounds plausible that multiple users competing for the same resources



(a) Map.

(b) Bandwidth.

Figure 3.13: Car commute route between Oslo and Grimstad.

would cause congestion in peak hours of the day (e.g., in the rush hours when most people commute to work). Thus, we expected this to be a contributing factor to fluctuations in available bandwidth. To test this hypothesis, we compared bandwidth logs along certain commute routes at different times of the day.

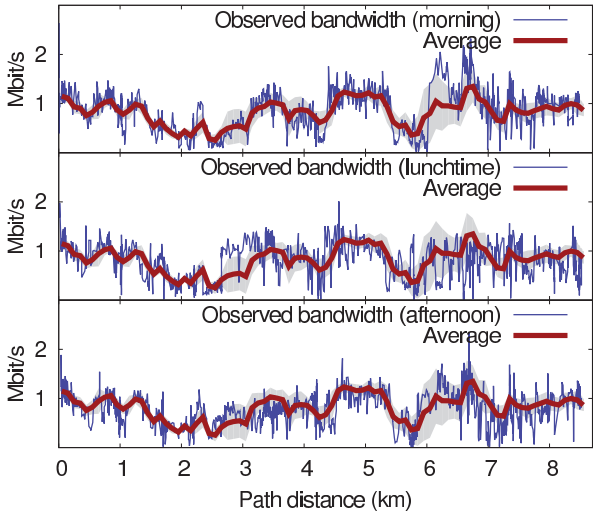


Figure 3.14: Bandwidth along the tram commute route on different times of the day. No significant differences were observed.

Surprisingly, the time of day on which the measurement was done was not found to significantly influence the results. This is shown graphically in figure 3.14, where we compare the tram route bandwidth curves measured on different times of day. This also applies to the other commute routes, but we picked the tram route to illustrate the point, because the bandwidth on the tram route had less randomness than the other routes (this randomness could not be explained by time of day differ-

ences, as we found the variance in results equally large between different runs at the same time of day as between different runs at different times of the day). Using a route with predictable bandwidth makes it easier to visually compare measurements done on different times of the day, and confirm that the results are more or less the same.

In summary, time of day differences were not an issue in 2011 (when these measurements were done), but we expect this to change in the future when more people use the streaming services available to mobile devices. However, for this initial study, when it was not yet a factor, we did not differentiate the returned results according to time or date. Nevertheless, if future measurements indicate time differences, we can easily take into account the time of day, and day of week.

3.5 Summary

This chapter presented performance measurements of a real 3G/HSDPA network, using both UDP- and TCP-based data transfers. We observed that the performance of 3G/HSDPA networks is good, as long as the capacity of the network is not congested. In a congested 3G cell, one will experience large fluctuations in bandwidth and jitter, and unfair distribution of bandwidth between multiple users sharing the same cell. When moving between cells, we found large variations depending mostly on geographical location. The TCP-based tests also showed that the variance in bandwidth between multiple runs along the same path is relatively small, meaning that the bandwidth for a specific location can often be predicted with high accuracy.

To get good performance in a mobile 3G/HSDPA network, it is important to avoid congestion and to reduce the data rate when the connection quality is poor. This reduces both latency in packet transmission and loss of data. It follows that a media streaming system for mobile networks needs to support bitrate adaptation. However, in areas with poor network coverage there are some outages that cannot be avoided, so the only way to avoid playout interruptions is to let the receiver prebuffer more content than is normal in a fixed network streaming scenario. The amount of buffering that is required for this varies a lot between different locations, so maintaining a fixed amount of data in the buffer is not optimal.

The next chapter starts by comparing the quality adaptation performance[1] of existing adaptive HTTP streaming products under the conditions observed in a real 3G/HSDPA network. Next, it presents our own adaptive HTTP streaming implementation that was designed to improve performance in these types of networks. Finally, a performance comparison is made between existing products and our own implementation.

[1]High performance in this context means high quality video streaming from a user's perspective.

Chapter 4

A Study of Bitrate Adaptation Algorithms for Adaptive HTTP Streaming Clients in 3G Networks

The previous chapter presented measurements showing that 3G/HSDPA can support video streaming, but that bitrate (and consequently quality) adaptation is necessary for mobile video receivers. Here, we will compare the bitrate adaptation algorithms in existing adaptive HTTP streaming solutions in challenging mobile streaming scenarios to see how well they solve the problems encountered. In this context, the performance criteria of a bitrate adaptation algorithm is the QoE from the user's perspective, i.e., how well it delivers an uninterrupted playout with stable quality and good bandwidth utilization. With these criteria in mind, we develop a new quality adaptation algorithm optimized for mobile networks and compare it with currently available products.

4.1 Streaming in 3G Networks using Adaptive HTTP Technology

Adaptive streaming over HTTP [150, 138] is rapidly becoming popular among commercial vendors of streaming technology. It can be implemented as a combination of simple servers and intelligent clients that make adaptation decisions based on local observations. The versatility and relative simplicity of the technology has made it successful on everything from high-speed fixed networks with HD-capable receivers to small handheld devices on mobile wireless networks. As a strictly pull-based approach, adaptive HTTP streaming is quite different from early video streaming techniques that relied on server-side decisions and multicast. Pull-based streaming has become viable because the development of the Internet infrastructure has made it possible to take cheap server capacity and backbone network capacity for

granted. Under these conditions, providing good QoE on a fixed high-speed network is not difficult any more, but it is considerably more challenging when the client's access network is a mobile wireless network with severe and frequent bandwidth fluctuations and outages. Such network conditions result in recurring buffer underruns and frequent quality switches, both harmful to the viewer's QoE [124, 169].

Under such circumstances, proper configuration of a media player's quality scheduler is both challenging and important. The overall aim is to provide the best possible viewing experience. Important steps towards this goal include:

1. Avoiding buffer underruns, as they cause interruptions in video playback.
2. Avoiding rapid oscillations in quality, as this negatively affects perceived quality [124, 169].
3. Utilizing as much of the potential bandwidth as possible to give the viewer a higher average video quality.

While actively using several commercial video systems in our other projects, we have experienced large differences in performance with regard to the above points. Adaptive video formats and streaming systems only offer the means to change the quality, and it is up to the client to determine how to use the adaptation mechanisms. Choosing when and how much data to retrieve, and which bitrates to use, are all non-trivial tasks. As mentioned above, perceived quality will be higher if changes in quality are gradual and do not occur too often.

This prompted us to conduct a more rigorous performance comparison, where we investigate how the players perform and adapt in challenging mobile streaming scenarios using typical bus, ferry, metro and tram commute routes in Oslo, Norway. In particular, we wanted to know how the different systems perform in terms of robustness against underruns, stability in quality, and bandwidth utilization.

From our experimental results using Telenor's 3G mobile network, we observe large differences in the rate adaptation policies. Apple and Adobe's players represent two opposites in that Apple focuses more on stable quality at the expense of high average quality, whereas Adobe does the opposite. Microsoft's Smooth Streaming [167] tries to achieve good stability as well as high bandwidth utilization. At the end of the chapter, we present our own implementation of a reactive rate adaption policy that improves on existing products in the use case of mobile video streaming with non-live content.

4.2 Related Work

Several studies have been performed on video streaming in vehicular mobile environments using ad-hoc networks. Asefi presents [56] new cross-layer protocols for vehicle-to-vehicle video streaming. Xie, Hua, Wenjing, and Ho propose [162]

two data forwarding schemes for highway environments. In [62], Bucciol, Masala, Kawaguchi, Takeda, and De Martin also focus on highway and urban environments, and performed transmission experiments while driving two cars equipped with 802.11b standard devices. However, video streaming over ad-hoc networks is a very specialized field, so despite sharing the vehicular mobility aspect, their results are not particularily relevant for streaming from a normal server to 3G clients using adaptive HTTP systems.

Yao, Kanhere, Hossain, and Hassan study [165] the effectiveness of adaptive HTTP streaming in vehicular mobile streaming. They used the same approach as we did, wherein a database of bandwidth traces was collected using custom-made tools in field tests with a moving vehicle, and experiments were conducted in a controlled lab environment on this data set using a bandwidth shaping tool that reproduced the conditions observed in the field. In contrast to our experiments, they used a Wireless Wide Area Network (WWAN), only had one route in their data set, only one type of vehicle (a car equipped with an external antenna to improve signal reception), and they did not include commercial adaptive HTTP streaming clients in their comparison. Instead, they used their own adaptive HTTP streaming prototype, and compared it to traditional progressive download streaming while experimenting with different parameters such as buffer thresholds and segment durations. Their results showed that adaptive HTTP streaming is effective in dealing with the bandwidth fluctuations that are inherent in vehicular mobility streaming.

Akhshabi, Begen and Dovrolis [53] compare Adobe, Microsoft and Netflix using a testbed that is similar to ours, but they use synthetic square waves as bandwidth curves, which look nothing like the strongly fluctuations bandwidth curves we have observed in real mobile wirless networks [136, 139, 137]. Netflix' streaming technology was not included in our comparison because there was no publicly available server implementation that we could integrate in our testbed. However, based on the results in [53], we believe Netflix' more aggressive behavior would put it somewhere between Adobe and Microsoft in terms of stability and bandwidth utilization.

Cicco and Mascolo [71] investigate Akamai's adaptive video streaming solution, again with a similar testbed using square wave bandwidth curves. Akamai's quality scheduler is server driven with no publicly available implementation. Thus, it was not included in our comparison.

The work that is most similar to ours was done by Müller, Lederer, and Timmerer [122], and was performed in parallel and published at the same conference as our published [135] version of this chapter. Apple, Microsoft and Adobe's adaptive streaming clients are compared with Müller et al.'s MPEG DASH implementation in a mobile streaming scenario, using real world bandwidth traces. Their test methodology is very similar to ours, using traffic shaping tools to reproduce bandwidth traces from real world experiements. Their results agree with our results.

Because several of the compared systems are closed source, we cannot know the exact algorithms, but we can still report their behavior. In earlier related work [53], this has been done by looking at the consumed bandwidth and buffer sizes, which is not necessarily closely related to video quality and user perception. In our experimental testbed, we have support for Adobe, Apple, and Microsoft's media players from a single video representation. This means that we encode the content once for each quality level, and use the same streams on all media players to be compared. Consequently, we can give a fair comparison of the resulting video playback quality.

4.3 Experiments

In this section, we present the performance comparisons that we performed on several adaptive HTTP streaming clients that were available at the time. We start by describing our testbed, which clients were included in the comparison, and how the comparisons were done. We conclude with the results of the comparison.

4.3.1 Tested Systems and Video Formats

Several competing HTTP streaming systems exist, and it is out of the scope of this dissertation to compare all of them. The Motion Pictures Experts Group (MPEG) has standardized an adaptive HTTP streaming protocol under the label DASH [150], but the standard does not specify how clients should adapt the quality. Because quality adaptation is determined by the implementation of the client, what we are comparing are actually different media players, not different adaptive HTTP streaming protocols. We have therefore selected three different, representative media players to investigate how they make adaptation decisions:

- Adobe's Strobe Media Playback (v1.6.328 on Flash 10.1 on a Windows 7 PC) using its native HTTP Dynamic Streaming format [51]. Note that this player is also known as OSMF, short for the Open Source Media Framework on which it is built. The license of the OSMF player has few restrictions, so content providers can use it without giving credit, hence we have little data on its popularity. One known commercial user is VideoPress [106].

- Apple's iPad player (iOS v4.3.3 (8J2)) using its native HTTP Live Streaming (HLS) format [127]. This is an embeddable player used in all iOS devices for all adaptive streaming, as it is required by Apple developer guidelines for all content exceeding 10 minutes in duration or 5 MB in a five minute period [6].

- Microsoft's Silverlight (v4.0.60531.0 on a Windows 7 PC), using its native Smooth Streaming format [167]. This player is used by broadcasters such as NBC for streaming major sports events like the Olympics and Super Bowl [3], by TV 2 for its online TV platform [1], and by CTV for the Winter Olympics [4].

On the server side, we used CodeShop's Unified Streaming Platform v1.4.25 [14] (integrated into the Apache v2.2.14 web server [10]) to support all streaming formats without requiring different video encodings; i.e., the only differences between the tests of the different systems were the protocol and media container formats. Differences in bitrates between formats (due to different container overhead) was always less than 12 %, meaning that the results were not significantly affected by the chosen streaming format[1]. The web server ran on a dedicated Linux box with 2 GB RAM and an AMD Athlon 64 3200+ CPU. Both the receiver and the web server were on the same 100 Mbit/s Local Area Network (LAN), using our custom-made throttling module to reproduce the behavior of a real 3G network (see section 4.3.3).

4.3.2 Video Stream Configuration

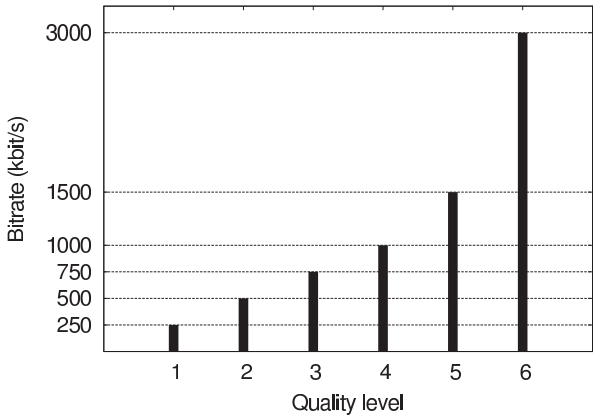


Figure 4.1: Relation of video quality levels and their average bandwidth consumption.

Ensuring fairness in the video stream means that the video segmentation has to be as equal as possible (duration and bits per segment) for all the different media formats used. We encoded video (a European football match) with a fixed segment duration of two seconds and six quality levels. The bitrates used for the six quality levels are shown in figure 4.1. Bitrates at the lower end were chosen based on Akamai's recommendations [52] for small handheld devices on mobile networks. Bitrates for the higher qualities were chosen based on subjective testing (encoding at different bitrates and comparing the encodings visually). Using this approach, we tried to achieve a linear scale in perceived quality. The reason for the larger

[1]We could have encoded an extra representation for HLS to compensate for this overhead, but sacrificing quality to achieve the same overall bitrate means that the quality levels would not be identical in all tests. Because the relative bitrate difference was less than 12 % even for the lowest quality level (where the relative overhead is highest), we deemed it negligible and decided instead to use the same representation for all tests.

leaps in bitrate between the highest three levels is diminishing returns in quality – it was necessary to double the bitrate to make level 6 visibly better than level 5.

4.3.3 Realistic and Equal Network Conditions

To perform a fair and realistic comparison of the different streaming systems, we need equal network conditions for each test which match the observed bandwidths in real mobile 3G networks. We have previously [136, 139, 137] performed a large number of real-world experiments, and while we found the bandwidth (as a function of geographical location) to be fairly deterministic, this study requires identical results on each run to achieve a fair comparison.

We found no available solution for precise traffic shaping based on a log of bandwidth samples, so we created from scratch a throttling module for the Apache web server. This module takes as input a bandwidth log (from a real-world test) that contains a single kbit/s number for every second of the session (the reason for using a granularity of one second and not smaller is that we only care about the average speed over an entire segment, which typically takes more than one second to download since the segments are two seconds in realtime). After loading the bandwidth log, the session is started by the next HTTP request. At time t after the session starts, the web server's maximum throughput for the next second will be $B(t)$, where $B(t)$ is the bandwidth at time t stated in the log that was used as input to the throttling module. In addition to bandwidth throttling, our Apache module also adds a delay of 80 ms to HTTP requests to simulate the average latency as experienced and measured in our most recent tests in real wireless 3G networks (we used the average as a constant, since the actual delay as a function of time was not part of the bandwidth logs we had previously recorded).

This approach means that each media player can be compared under exactly the same conditions, ensuring both fairness and reproducibility in our experiments, while at the same time being nearly as realistic as a field trial.

We selected four representative bandwidth logs from our database of measurements. Each log represents a typical[2] run in its respective environment. The four streaming environments are popular commute routes in Oslo (Norway) using ferry, metro, bus and tram. The maps for each route are shown in figure 4.2 and the corresponding bandwidth curves are shown in figure 4.3 (the first plot from the top for each of the routes). We can see that they represent different challenges with respect to both achieved rates and outages.

[2]Established by comparing it to the average of multiple results.

(a) Ferry

(b) Metro (dotted line in tunnel)

(c) Bus

(d) Tram

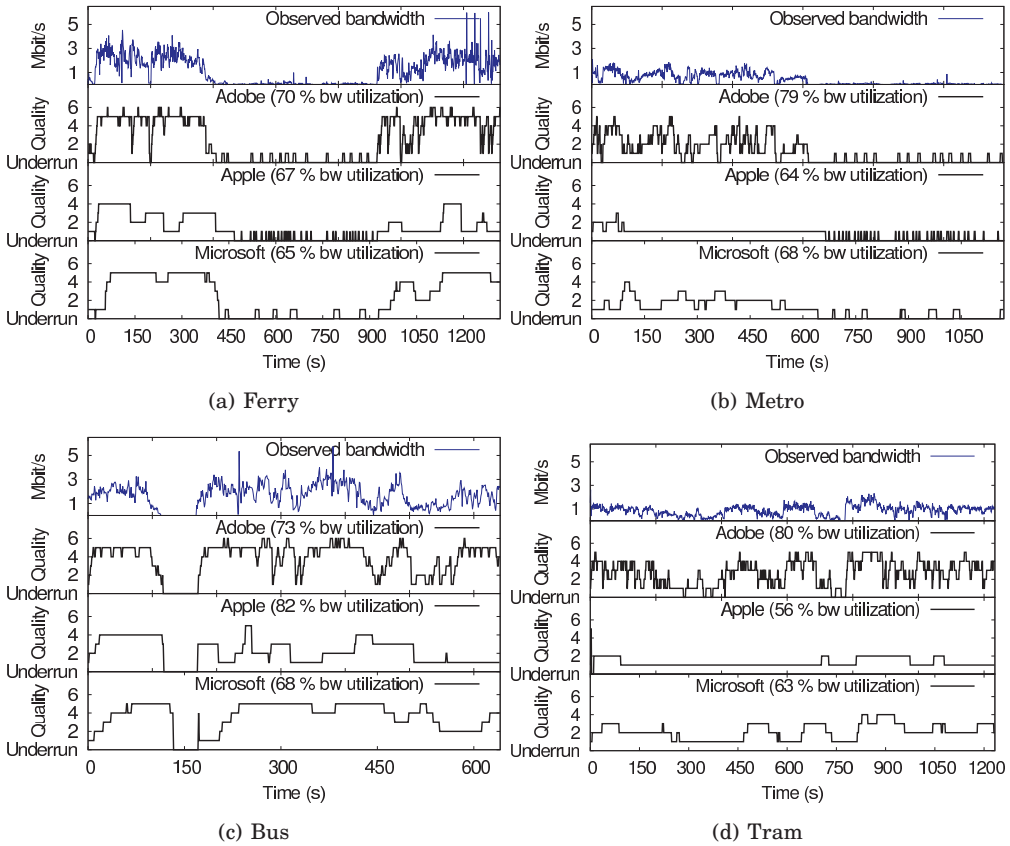
Figure 4.2: Commute routes used when comparing adaptive media players.

4.3.4 Logging the Video Quality of Downloaded Segments

While streaming, we used tcpdump [39] on the server to log every packet transmitted to the receiver, so that we could measure the actual achieved throughput (which might be less than the bandwidth cap set by the throttling module, for example if the receiver's buffers are full and it starts to idle). The packet dump contains every HTTP GET request for every downloaded segment, and because the quality information is encoded in the URLs, the packet dump also contains the information we need to plot the quality level as a function of playout time. However, because we are testing proprietary media players where we do not know the state of their internal buffers, buffer underruns were logged manually by actually watching the video in every test, and registering the times when the video stopped and resumed.

4.4 Results and Analysis

For each route, we compare the media players using (1) the video quality level as a function of playout time, (2) the amount of video data presented in each quality level including buffer underruns, and (3) the length of each playout interval at a given quality to give an indication of how stable the playout is in terms of video quality. These properties are plotted in figures 4.3, 4.4 and 4.5, respectively, for all four routes.



(a) Ferry (b) Metro

(c) Bus (d) Tram

Figure 4.3: Comparison of quality scheduling for the four commute routes. The first graph from the top in each subfigure shows the bandwidth log that was chosen for the respective routes, and the graphs under it show the quality that was achieved. Quality is plotted at playout time. This implies that effects of network outages are reflected with a delay when buffers run dry.

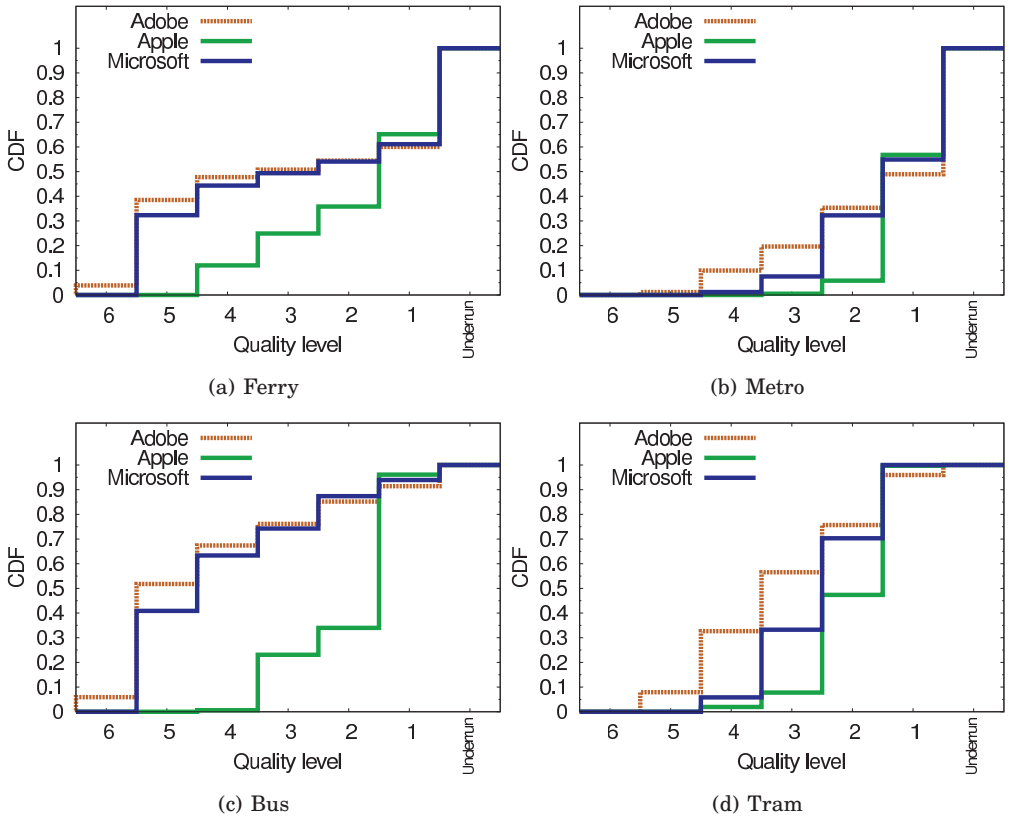


Figure 4.4: Cumulative distribution function of quality. The relative height of each step represents the total amount of time that a video is played out at a given quality level (and the last step shows the cumulative duration of playout interruptions). Higher towards top-left corner is generally better, but the frequency of quality changes (shown in figure 4.3) can also affect perceived quality, and this information is not represented in these plots.

4.4.1 Adobe

Comparing Adobe's quality level plots in figure 4.3 with the bandwidth plots, one can clearly see that their shapes are almost identical. From this, we conclude that the quality scheduler in Adobe's Strobe player bases its decisions almost exclusively on the most recent bandwidth numbers. The next segment to be downloaded is the one whose bitrate is closest to the current bandwidth, with no considerations to stability or safety margins. As a result, the users' QoE suffers due to buffer underruns and too frequent oscillations in quality (figure 4.5). Despite minimal use of buffering, the scheduler achieves decent bandwidth utilization, mainly because high bitrate segments were downloaded quite often (even when unsafe to do so), meaning more bytes per download request, and thus, less wasted bandwidth.

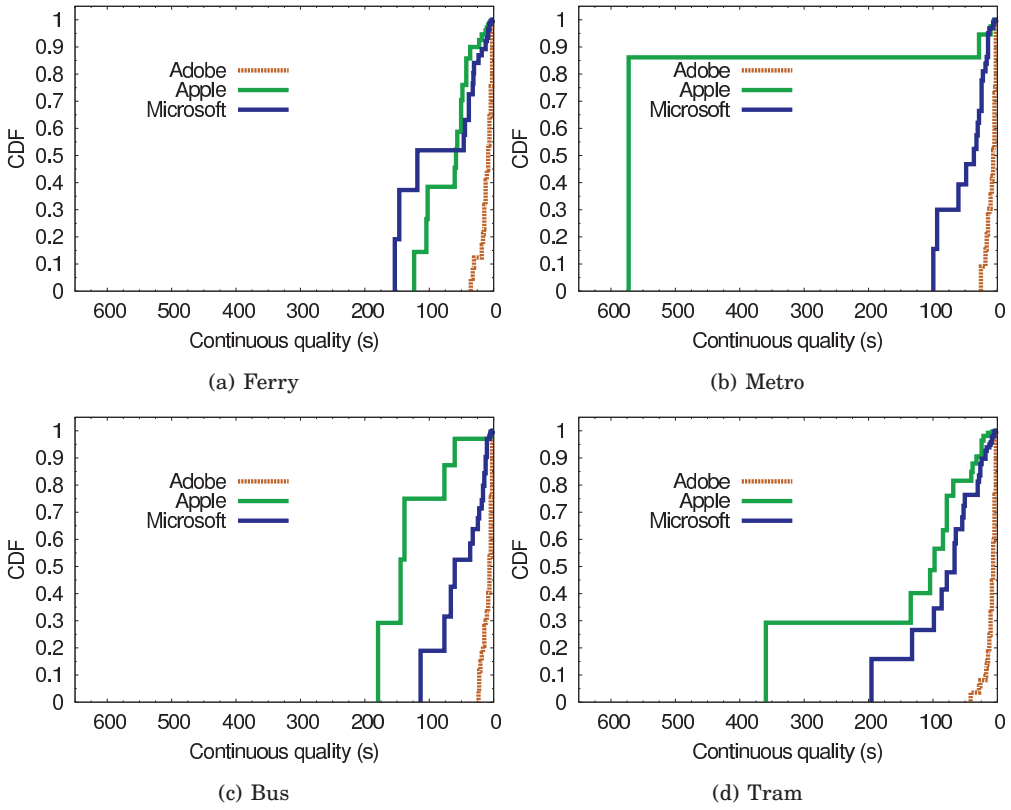


Figure 4.5: Cumulative distribution function of quality stability on the four commute routes. A point in the plot represents how much of the playout time (y-axis) has been played out with stable quality intervals larger than a given length (x-axis). Buffer underruns are not counted. *Note:* This plot considers only stability but ignores the quality of the playout interval (the long intervals of the Apple player in figures 4.5(b) and 4.5(d) are playouts at quality level 1 as seen in figures 4.3(b) and 4.3(d)).

4.4.2 Apple

The quality scheduler in Apple's iPad player stands out from the others by being more careful about increasing quality. Its frequent use of low quality segments produces stable quality (figure 4.5) with long intervals in the same quality. The tendency to pick low quality levels is clearly seen in figure 4.4. Despite having lower quality on average than the other players, Apple's bandwidth utilization is higher than one would expect. The reason for this is that the bandwidth utilization number does not take into account whether the downloaded video data was actually used. Unlike the other players, the iPad player often re-downloads segments – sometimes the same segment is downloaded two or more times, usually in different qualities, but not always. This means that some of the downloaded video data is never used, and bandwidth is wasted.

4.4.3 Microsoft

Microsoft's Silverlight player achieves a nice balance between bandwidth utilization and stability. Compared to Apple's player, it uses more of the available qualities (figure 4.4), while still achieving a fairly stable viewing experience (figure 4.5), setting it apart from Adobe's player. We observe that Microsoft fills its buffers when the bandwidth is high, progressively increasing quality, instead of instantly jumping to the quality level whose bitrate is closest to the current bandwidth. However, we observe that the quality increases just as quickly when the bandwidth is poor, indicating that filling its buffers is not a priority.

Microsoft's biggest problem is that the buffer tends to be very small (as seen in figure 4.9 when comparing occurrences of buffer underruns with the Netview results), especially considering that it is designed for PCs with plenty of memory. This limitation makes it unnecessarily vulnerable to buffer underruns. Average quality is better than Apple's, but not quite as high as Adobe's numbers. Microsoft's player would have achieved better results with a better buffering strategy, as it often wastes bandwidth by idling, even when its buffers are almost empty. On the other hand, the bandwidth utilization is quite stable at about 65 % and the Silverlight media player is intended to run as a browser plugin, so it might be a design goal to leave some bandwidth available for other services.

4.4.4 Summary of Results

We found large performance differences between the various systems, even though none of them have advantages over each other in terms of what information is available to make quality adaptation decisions. Apple and Adobe's players represent two opposites in that Apple sacrifices high average quality for stable quality, whereas Adobe does the opposite. Microsoft's Silverlight player falls in between, but without compromising too much on either parameter.

From our experiments, we conclude that the quality scheduler (which decides which quality to use for every media segment downloaded) has a large impact on the QoE in adaptive HTTP solutions and that several products on the market have a definitive potential for improvement when streaming in mobile networks. The remainder of this dissertation will focus on development of new quality schedulers and other technologies for improving performance in mobile streaming scenarios.

4.5 An Improved Quality Scheduler for Mobile Video Streaming

The quality schedulers in current commercial products from Apple, Adobe and Microsoft show that there is a potential for improvement when streaming over wireless

links to mobile receivers. In this section, we present a new quality scheduler that performs better than all of them under the following conditions:

- Non-live content (pre-fetching content is necessary for full effect of our algorithm).
- Wireless networks.
- Mobile receivers.

Our new quality scheduler also works well for live streams, but as we are limited in the amount of buffering we can do with live streams, the improvment with respect to buffer underrun protection is reduced. However, even with live streams, we are still able to reduce unnecessary quality fluctuations, making the playout more smooth than current commercial products.

4.5.1 Quality Scheduling Parameters

A quality scheduler's performance is determined by its buffer size and how it decides which quality level to use. We will in the following present an exhaustive list of the parameters that constitute our proposed quality scheduling algorithm. For each graphical illustration of a given parameter's effect, we have chosen a bandwidth curve based on how well it illustrates the effect of the parameter. In other words, the bandwidth curves are those that, for each parameter, have problems for which the parameter provides solutions.

Buffer size

Because outages can last for minutes in mobile streaming, a large buffer is more important than in fixed network streaming. How large the buffer must actually be depends on which bitrates are used and the durations of the network outages. One must simply choose a buffer size which is long enough to cover most outages, but not too large for most devices capable of mobile video streaming. A smaller buffer simply means that the media player potentially runs out of data sooner should the connection go down. Thus, the optimal solution for a media player would be to avoid setting an artifical limit, and let the player buffer as much as it wants for as long as memory is available. However, most media players have to share resources with other programs running on the same device, and most programs do not handle out-of-memory situations gracefully. Because our tests were conducted on laptops with several gigabytes of memory, we set an artifical limit of 200 MB, because most modern hand-held devices can spare this much memory, and it is sufficient for most outages: Even in the highest bitrates (rarely higher than 5 Mbit/s for adaptive HTTP streams), 200 MB is over five minutes of video. As shown in figure 4.3, this is longer than most outages in our experiments with urban environments (the

only exception in downtown Oslo is the underground subway system, where there is almost no connection). Finally, we add that our algorithm was never able to accumulate more than 40 MB in its buffers, because it also wants to play video in high quality, not just pick a low quality all the time to maintain a high buffer fill level. Thus, we could have reduced our buffer limit from 200 MB to 40 MB without affecting our results.

Scaled buffer thresholds for quality levels

The reactive algorithm upgrades the quality once the buffer duration reaches certain chosen thresholds. However, a problem is that, to make a difference in quality in the higher quality levels, the bitrate must often increase dramatically. As shown in figure 4.1, the difference between quality levels 5 and 6 is 1500 kbit/s, while the difference between levels 1 and 2 is 250 kbit/s. To avoid wasting resources prematurely on very high quality levels, the buffer fill level thresholds for jumping between quality levels should take the bandwidth difference between the levels into account. To achieve this, we suggest setting the buffer fill level thresholds using the following simple rules: When the buffer is empty, the lowest quality (level 1) is selected; in the general case for quality levels higher than 1, the following bitrate scaling equation applies:

$$T_N = B \cdot \frac{R_N - R_1}{R_2 - R_1},$$

where T_N is the buffer requirement in seconds for quality level N, and R_N is the bitrate of quality level N. Thus, when the buffer has B seconds of video, level 2 is used; requirements for higher levels depend on their relative increase in bitrate.

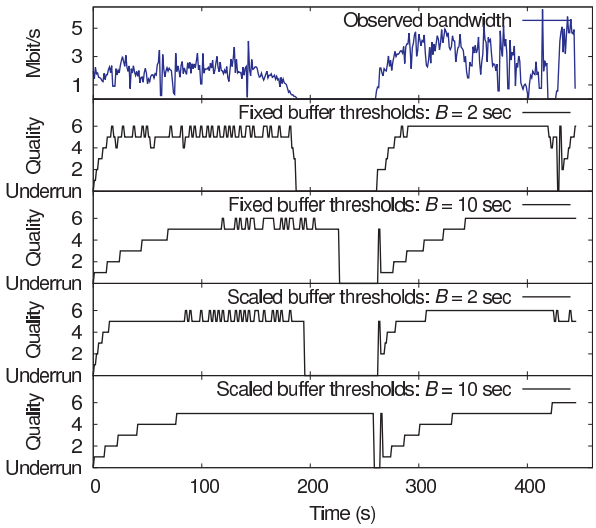


Figure 4.6: Altering the buffer thresholds.

Figure 4.6 compares four different settings. The first two use a fixed step size B of 2 and 10 seconds between all quality levels, i.e., no bitrate scaling. The figure shows that a low step size leads to rapid buffer drainage caused by frequent jumps into quality level 6, leading to several buffer underruns. Increasing the step size to 10 seconds helps, but quality level 6 is still chosen too frequently because the algorithm is not aware of the cost of this level compared to the lower qualities.

The last two settings in the figure set the buffer thresholds to T_N, defined by the bitrate-scaling formula described above, with two different base step values ($B = 2$ and $B = 10$). This eliminates the frequent attempts at quality level 6, which not only causes flickering quality but also prevents the buffer from growing.

Figure 4.6 shows a large improvement when enabling bitrate scaling and the larger base step size. Using these settings, we get a much better viewing experience, almost removing the buffer underruns and having a more stable video quality.

Different thresholds when going up and down in quality

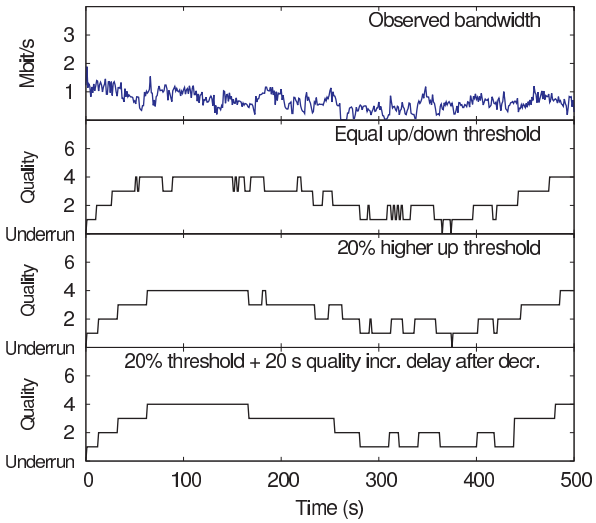


Figure 4.7: Removing rapid oscillations

Another goal when improving the viewing experience is to avoid rapid oscillations in quality, as they cause a flickering effect that greatly reduces perceived quality [169, 124]. One way such oscillations can occur is if the buffer fill level is floating near a quality level threshold. An easy way to limit this is by having slightly different buffer fill level thresholds when going *up* in quality than when going *down*. Figure 4.7 shows how requiring 20 % more in the buffer when going up than down can avoid having the quality level oscillate between two quality levels (we have also tried percentages both smaller and larger than 20, but the differences are small, and the current value of 20 % seems to be a good tradeoff). Since we only require

20 % more video in the buffer, it has a negligible effect on how fast the quality increases. Still, it is sufficent to achieve our desired effect. The important thing is simply that we make the edges between quality levels wider, so that it is unlikely for the client to shift back and forth between two levels when the buffer fill level happens to be near a quality level threshold.

Delayed quality upgrades after a quality drop

Even when having different quality thresholds for going up and down in quality, some oscillations occur when the bandwidth is changing very rapidly. As an additional countermeasure, one extra rule is added: *Do not increase quality before at least T seconds has passed since the last drop in quality*. This further reduces oscillations in quality. This parameter was set to 20 seconds, based on subjective testing of acceptable quality oscillation frequencies. The effect of this parameter is shown in figure 4.7 (the plot at the bottom).

Quality level selection should be limited by estimated download rate

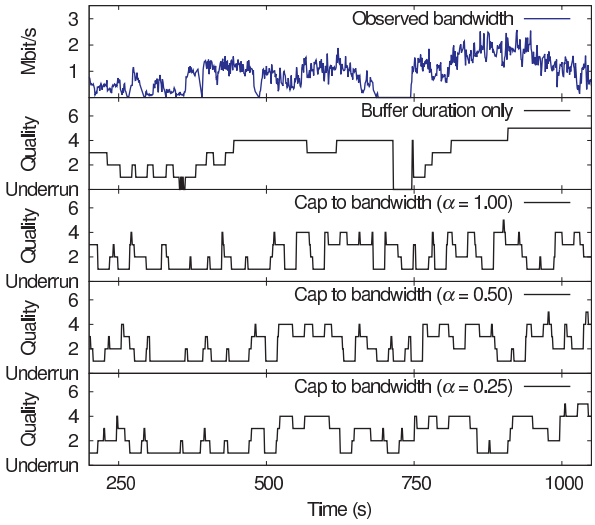


Figure 4.8: Cap quality selection effects

By never allowing the media player to select a quality level whose bitrate exceeds the current download rate, the buffer will rarely drain, and many buffer underruns can be avoided. However, this also leads to rapid fluctuations in quality, because the quality will suddenly drop when a transient dip in download rate occurs. This is especially harmful when the quality drops multiple levels at once, e.g., from level 6 to level 1. One way to reduce this flickering effect is to smoothen out the bandwidth curve using an exponentially-weightened moving average. This means that

we have to choose the smoothing factor α, which represents the weight given to the most recent bandwidth sample. Mathematically, $B_N = \alpha \cdot b_N + (1-\alpha) \cdot B_{N-1}$, where B_N is the moving average bandwidth sample N, and b_N is the raw, unweighted, bandwidth sample N. Here, $0 < \alpha \leq 1$, and the closer α is to 1, the more weight is given to the most recent bandwidth observation, and the more the quality will fluctuate. If α is made too small, the current download rate estimates will be misleading, and can lead to either buffer underruns (if the estimated download rate was too high) or too low average video quality (if it was too low). After experimenting with different α values in several field trials, we decided to use $\alpha = 0.25$, as it was found to be a good trafe-off between stability and underrun protection (see figure 4.8). However, we *completely disable the bandwidth cap on quality selection* when the current quality $Q > 3$. This prevents sudden deep drops in quality, which would negatively impact the QoE. With the buffer threshold rules described above and the quality level configuration shown in figure 4.1, the media player will have at least 30 seconds in its buffers when $Q > 3$ (due to the buffer fill level requirements discussed previously in this section), so we can afford to drain the buffers to achieve more stable quality.

4.5.2 Evaluation of the New Reactive Quality Scheduler

In the previous subsection, we described the parameters that constitute our quality scheduling algorithm, and the rationale behind them and the values we have used. To summarize:

- Our buffer limit is set very high so that data can potentially be available for longer outages, and so that we can utilize more of the bandwidth (extra bandwidth in addition to the playout rate can be used to grow the buffer).
- The buffer fill level thresholds for switching between quality levels are scaled according to the relative bitrate between the levels. This is done so that the quality scheduler can take into account that some quality levels may be very expensive to use.
- The buffer thresholds mentioned in the above point are set slightly different depending on whether the quality switch is towards lower or higher quality. The reason is that this reduces the number of rapid fluctuations in quality that can occur if the buffer fill level is floating right around a threshold limit.
- After a drop in quality, the quality scheduler prohibits a switch to higher quality until some period of time has elapsed. This further reduces the number of wasteful and annoying fluctuations in quality.
- Before the buffer is sufficiently full, the quality scheduler limits itself based on the estimated available bandwidth. In effect, when the buffer fill level is

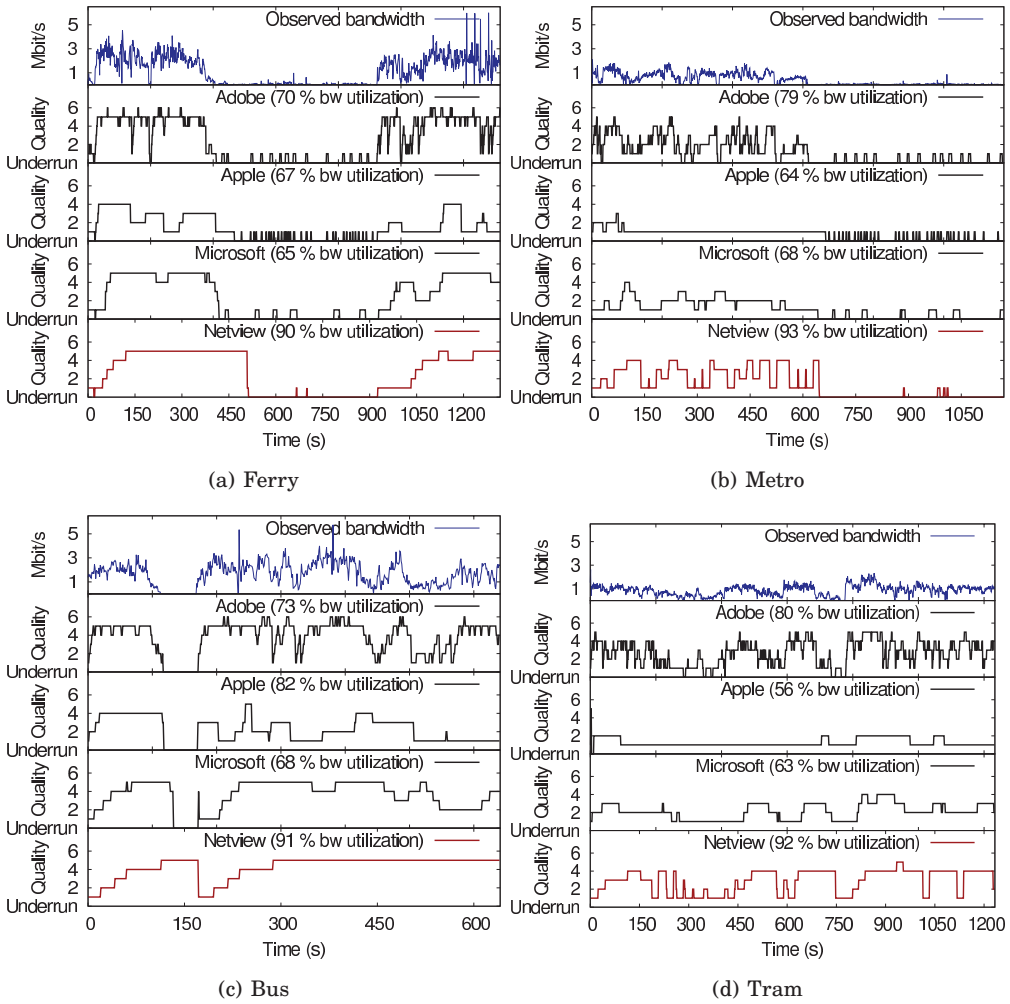


Figure 4.9: Comparison of quality scheduling for the four commute routes. The first graph from the top in each subfigure shows the bandwidth log that was chosen for the respective routes, and the graphs under it show the quality that was achieved. Quality is plotted at playout time. This implies that effects of network outages are reflected with a delay when buffers run dry.

low, the quality scheduler tries to avoid draining it by only using quality levels whose bitrate are lower than the potential download rate.

The results are shown in figures 4.9, 4.10, and 4.11. We see that compared to the other commercial systems, this algorithm offers better protection against buffer underruns (note for example figure 4.9(c) where the Netview scheduler is the only one that avoids the underrun), is more stable in most scenarios, and makes better use of the available bandwidth (resulting in higher average quality). However,

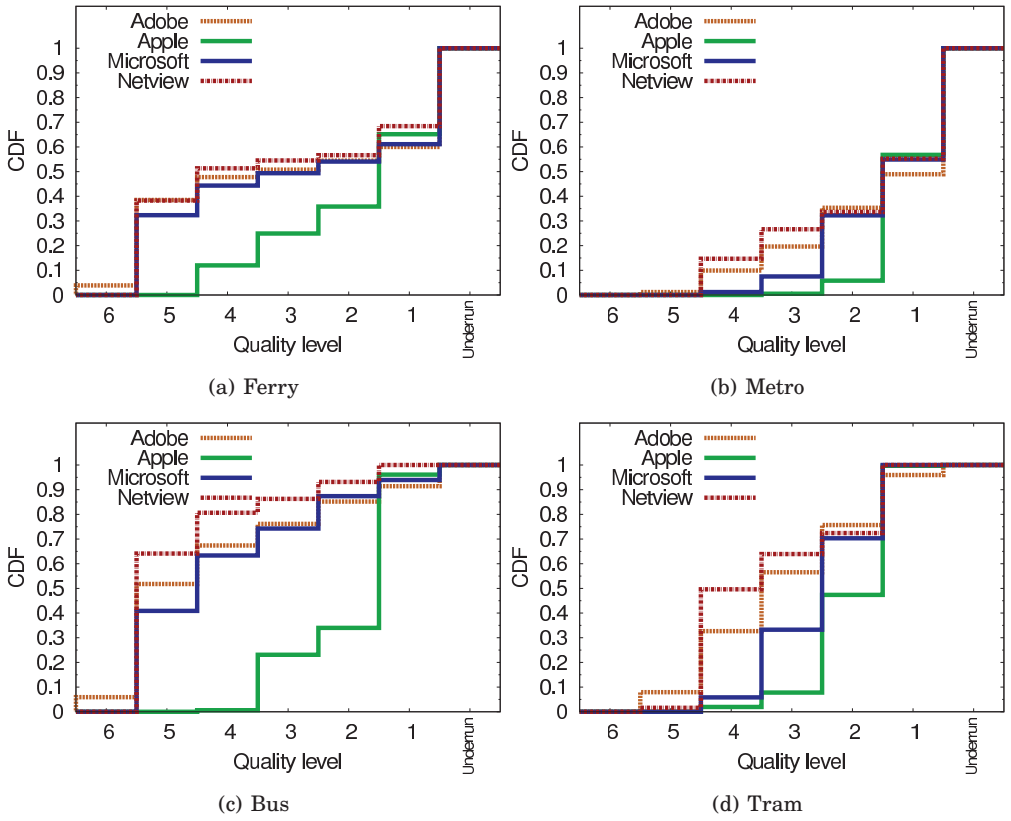


Figure 4.10: Cumulative distribution function of quality. The relative height of each step represents the total amount of time that a video is played out at a given quality level (and the last step shows the cumulative duration of playout interruptions). Higher towards top-left corner is generally better, but the frequency of quality changes (shown in figure 4.9) can also affect perceived quality, and this information is not apparent here.

while building our set of bandwidth logs for all four routes, we observed that the bandwidth numbers for a given position are fairly deterministic. Thus, assuming that one can predict the travel path and speed, one can also predict the bandwidth. Utilizing such a prediction, it should be possible to improve the QoE further. This is discussed further in the next subsection.

4.5.3 A Comparison with the Theoretical Optimum Result

To estimate how much can be gained by utilizing bandwidth prediction, we compare our reactive quality scheduler with the optimal result, i.e., the result of an omniscient quality scheduler that pre-fetches video based on a perfect prediction of future bandwidth and trip duration. This is, of course, impossible in a real-world implementation, but it can be done in a lab environment where we run simulations

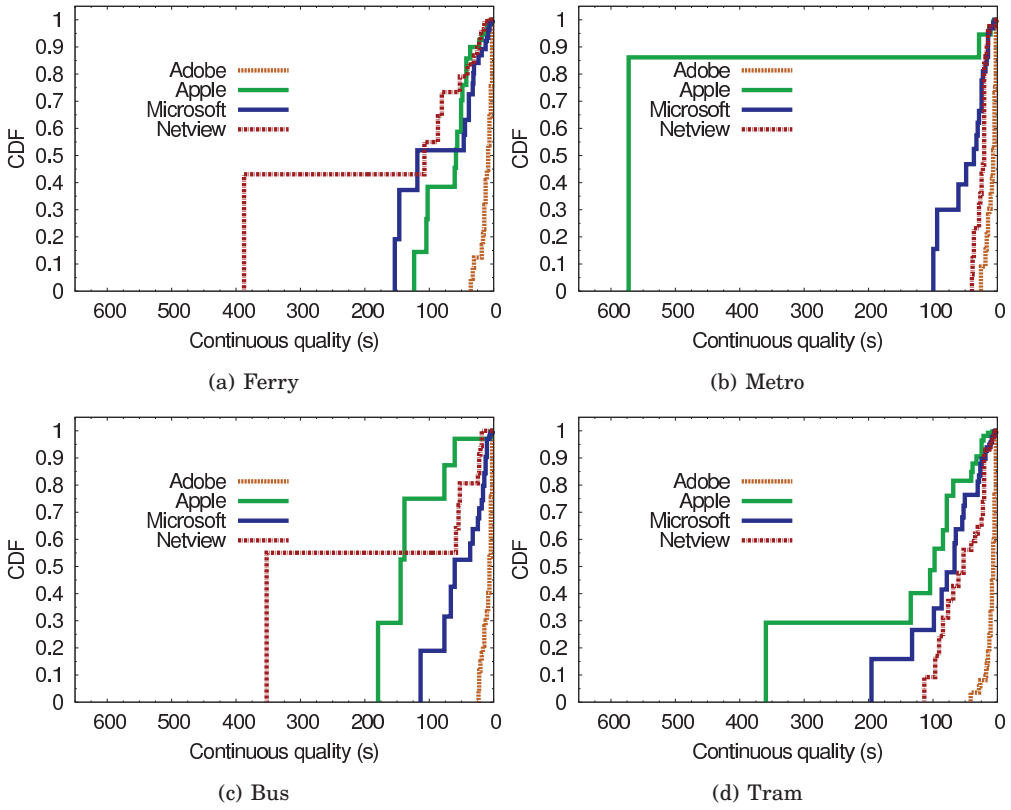


Figure 4.11: Cumulative distribution function of quality stability on the four commute routes. A point in the plot represents how much of the playout time (y-axis) has been played out with stable quality intervals larger than a given length (x-axis). Buffer underruns are not counted. *Note:* This plot considers only stability but ignores the quality of the playout interval (the long intervals of the Apple player in figures 4.11(b) and 4.11(d) are playouts at quality level 1 as seen in figures 4.9(b) and 4.9(d)).

on previously collected bandwidth traces. Thus, the only purpose of the omniscient scheduler is to serve as a benchmark for our real quality scheduler, to see how it compares to the optimal result.

The omniscient scheduling algorithm is simple to describe: As any other quality scheduler, it makes a quality decision for every segment in the stream. In contrast to a real quality scheduler, it *knows* the remaining duration of the streaming session, the available bandwidth for every remaining second, and the exact number of bytes in every segment in the stream. Thus, it can flawlessly simulate a streaming session from the current time and to the end of the streaming session. It simulates one such streaming session for every quality level in the stream, and the *highest* quality level that completes with the *fewest* buffer underruns is the chosen quality.

It follows logically from this algorithm that the quality can *only increase* throughout the streaming session: If a drop in quality would have been required to avoid

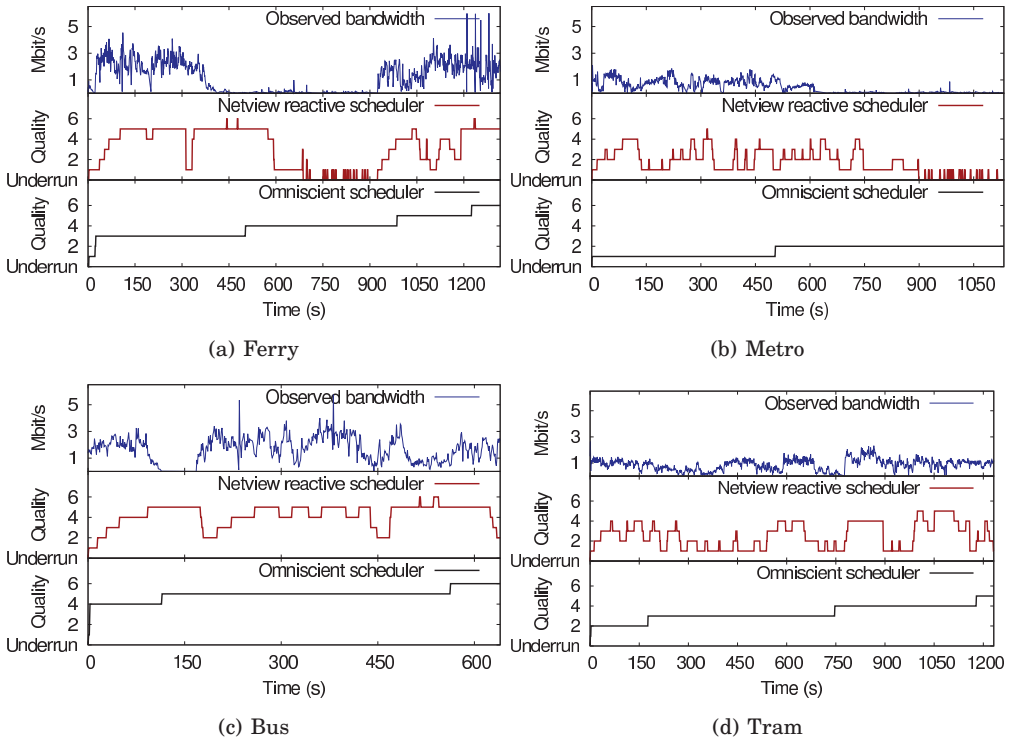


Figure 4.12: Comparison of our reactive quality scheduler against the optimal result with an omniscient quality scheduler that pre-fetches video based on perfect predictions.

a buffer underrun, the scheduler would not have picked that quality level to begin with (since every quality level picked has been confirmed to be free of underruns for the remainder of the streaming session). The reason the quality can increase throughout the session is that when a quality level choice is made at time t_{now}, there will be times t_{low} between t_{now} and t_{end} where the buffer is almost empty. Had a higher quality level been chosen at t_{now}, an underrun would have occured at t_{low}. However, once we are past these low-buffer points, the next simulated streaming session may find that a higher quality level can now safely be used, and the quality increases. Because the omniscient scheduler has perfect knowledge, it always consumes its buffers completely.

The results of the omniscient scheduler for all four routes are shown next to our best reactive algorithm in figure 4.12. The results clearly show that our best reactive implementation, although superior in most respects to quality schedulers used in other commercial systems, is greatly outperformed by the omniscient scheduler. In particular, the omniscient scheduler's playout is completely free of buffer underruns, and much more stable (since drops in quality will never occur). We conclude from this that a quality scheduler with an accurate bandwidth prediction can greatly improve the perceived video quality compared to a purely reactive scheduler.

4.6 Summary

According to our metrics, Microsoft's Smooth Streaming client has the best performance in mobile streaming among previously available commercial streaming clients (that we tested). Adobe's implementation seems to make no effort to protect against buffer underruns and quality fluctuations, while Apple's algorithm is too cautious, resulting in low average quality. We then presented a new quality scheduling algorithm with results that are similar to Microsoft's client, but with better protection against buffer underruns and better bandwidth utilization. However, when compared to the theoretical optimum result with an omniscient quality scheduler that knows everything that is going to happen, we see that we are still far from the optimal result. Our TCP throughput traces in section 3.3 show that the variance in throughput between different runs (along the same path) is often small, indicating that bandwidth predictions can be possible. The next chapter describes a new approach to quality scheduling that utilizes the geographically deterministic bandwidth figures to make predictions, in an effort to close the gap between the optimal result and a real-world implementation of a quality scheduler.

Chapter 5

Video Streaming Using a Location-Based Bandwidth-Lookup Service for Bitrate Planning

The previous chapters briefly described the evolution of video streaming from early UDP-based protocols to the adaptive HTTP streaming systems that are most popular today, and presented a comparison of commercially available adaptive HTTP streaming players. The focus of this comparison was the quality adaptation behavior which, of all the differences between the various players, is the one that has the greatest impact on perceived video quality. Next, a new algorithm for quality scheduling was presented, that offered more stable quality, fewer buffer underruns and better bandwidth utilization than the currently available commercial systems. Finally, we showed that there is still much room for improvement, if a quality scheduler could be designed to make bandwidth predictions and plan ahead based on historic bandwidth traces. In this chapter we develop a predictive quality scheduler that goes a long way to close the gap between the reactive scheduler (presented in section 4.5) and the optimal result achieved with omniscient prediction (described in subsection 4.5.3).

When using mobile Internet devices, it is a common occurrence to have the connection go down for significant periods of time. For example, one could enter a tunnel, or when travelling by sea, the distance to the nearest base station could be large. Such losses of connectivity will at best result in low quality video presentations with frequent jumps between quality levels, but having the video playout stop completely due to buffer underruns is also common. Either way, the effect is harmful to the users' perceived quality. These scenarios would greatly benefit from capacity planning which could be used to pre-buffer and smoothen the video quality

over a longer period. However, if nothing is known about the likelihood of bandwidth fluctuations, the only safe option is to use a low video bitrate and download as fast as possible until the buffer is large enough to last through most normal outages. For this conservative approach to work, the chosen quality usually has to be much lower than the average quality achieved by the omniscient scheduler described in subsection 4.5.3.

Ideally, the receiver would know in advance precisely how the network behaves over the course of the streaming session, regardless of how the user moves around with the streaming device. We will show how video receivers equipped with GPS devices can be used to automatically build a database that enables bandwidth prediction through geographical location and historic bandwidth measurements, and that the prediction can be sufficient even with just a few samples per position. Using such a location-based bandwidth-lookup service, the application can predict future fluctuations in bandwidth for a trip along a predicted route, and present a smoother, higher quality video stream to the users[1].

By the end of this chapter, we present an adaptive bandwidth prediction algorithm that combines the reactive buffer-based algorithm described in section 4.5 with a prediction model. Using the bandwidth data presented in section 3.3, we first simulated video streaming using several algorithms, and compared the results with both the ideal case (the omniscient algorithm with complete a priori knowledge) and the classical case (no knowledge of the receiver's geographical location, using only currently buffered data and observed bandwidth when scheduling quality). We implemented the best performing algorithm in Netview's adaptive media player [7], and did real video streaming sessions along all four commute routes to confirm that our method works just as well in the real world.

The real-world tests show that our bandwidth prediction system gives significant improvements in perceived video quality. Compared to the reactive algorithm, the predictive algorithm has less unused video in the buffer at the end of the trip, it experiences fewer playout interruptions and fewer changes in video quality.

5.1 Related Work

Video streaming has been a hot research topic for a couple of decades, and many video services are available today. A remaining challenge is to adapt video streaming to the unpredictable behavior of wireless networks like General Packet Radio Service (GPRS) and High-Speed Packet Access (HSPA). With mobile receivers in such networks, fluctuating network bandwidths strongly influence the video streaming service performance [137, 74], raising a need for bitrate adaptation to cope with

[1]Non-GPS clients can also use this information for planning trips using public transportation, although they cannot adapt to unexpected situations. This kind of use is beyond the scope of this dissertation.

temporary connection loss, high packet error rate and insufficient channel capacity, etc. In order to reduce the effects of fluctuations in download rate and temporary link interruptions caused by, for example, changes in the channel conditions or the wireless interface, one important strategy has been to adapt video quality, and thus resource consumption, to resource availability. Currently, several commercial systems like Apple's HLS [127] and Microsoft's Smooth Streaming [167] monitor the download speed of video segments and dynamically adapt to resource availability changes by switching between video segments coded in different qualities and bitrates. This is a highly popular approach, but it can be a challenge to avoid too frequent jumps in quality, which may annoy the users [169, 124]. As such, these systems use a traditional preloading approach, smoothening out the quality changes over a larger window.

To adapt to varying network conditions, several similar preload approaches use (hierarchical, layered) scalable video codecs such as scalable MPEG (SPEG) [94, 110], Multiple Description Coding (MDC) [87], and the Scalable Video Coding (SVC) extension to H.264 [146]. For example, priority progress streaming [94] makes a prefetch window which is first filled with the scalable codec's base layer. If there is more time (and bandwidth) before the window should be played out and the next window started to be filled, quality enhancing layers are downloaded. Schierl et al. propose [143] a priority-based media delivery for wireless 3GPP using SVC. A transmission scheduling algorithm prioritizes media data according to its importance in both RTP and HTTP-based streaming scenarios. This system prebuffers low-quality (but high priority) data of a given window length and increases the quality if the buffer is filled. Another codec based solution is the SP/SI-frame extension [104] to the H.264 video coding format. (This extension, MDC, SVC, and SPEG are all described briefly in section 2.2.)

A cross-layer design to improve the H.264 video stream over IEEE wireless networks is proposed by Mai, Huang and Wei [119]. This system discovers the importance of packets containing Network Abstraction Layer (NAL)-units according to their influence on picture quality in terms of PSNR and maps the video packets to appropriate access categories with shortest expected waiting time. The most important packets are sent first and late (low priority) packets can be dropped on the sender side.

The video playback may also be adapted to local resources like remaining battery power on mobile devices. A QoS adaptation model [152] for mobile devices can be used to automatically adjust the video quality (and processing requirements) from the remaining battery power, desirable playback duration and the user's preference. A similar approach is presented by Hsu and Hefeeda [93], where a quality-power adaptation framework controlling the perceived video quality and the length of viewing time for both live and on-demand broadcast scenarios on battery-powered video receivers. The framework predicts the remaining battery life-time and adjusts the amount of SVC-coded video data to receive and process.

There exist several approaches for delivering video streams to moving vehicles where the network is partitioned and there are temporal channel variations. For example, V3 [89] streams live video from traffic incidents by constructing a mobile ad-hoc network using a vehicle-to-vehicle approach, using the different cars to forward video data in a peer-to-peer manner. The cars are assumed equipped with GPS devices which enable the vehicles to track their location and trajectory.

There has also been performed experiments collecting network characteristics in such mobile scenarios. In [112], Lee, Navarro, Chong, Lee, and Gerla captured traces of signal-to-noise ratios along with GPS coordinates from vehicles to later objectively evaluate and compare different rate adaptation schemes in order to have the same test environment for all tests. Thus, these measurements were not used for predicting geographical bandwidth availability.

General QoS-prediction services and QoS-information systems have been suggested several times. Sun, Sauvola, and Riekki describe [151] an idea of using network resource awareness on the application level, but there is no QoS-prediction service, users only make predictions based on their own history. Wac, van Halteren, and Konstantas suggest [155] a general QoS-prediction service containing real end-to-end performance metrics that is both generated by and available to other users, and the feasibility of using such predictions based on geographical location, time and historical data was proved for a mobile health monitoring application [155]. Similarly, Horsmanheimo, Jormakka, and Lähteenmäki performed a trial that demonstrates [92] the usefulness of location-aided planning and adaptive coverage systems for network planning and resource management of mobile networks. Their trial does not deal with adaptive media streaming, focusing instead on the accuracy of various location methods and evaluating the accuracy of signal level prediction as a function of the location. Furthermore, the data collected is intented for the network operators, not the users. Liva et al. propose [114] a gap filler to provide bi-directional connectivity to train-based terminals. A propagation analysis using a wave-guide theory model derives a path loss law that can be used in a link budget calculation, i.e., calculating how the signal behaves to predict gaps based on parameters like propagating mode, the form of the tunnel, attenuation due to roughness of the walls and antenna insertion loss.

The listed techniques in this section have all shown the ability to adapt to oscillations in resource availability and to some degree deal with network disconnections. Some of the approaches, like adapting the video streaming/segment download and playout to local device resources (e.g., remaining battery power), are completely orthogonal to our approach. With respect to distributed resources, the challenge is to know how the availability fluctuates to increase the average video quality and reduce the number of buffer underruns. Existing systems often monitor the observed throughput, but foreseeing future degradations or even connectivity loss is hard. To better enable the streaming application to predict future bandwidth

availability and then adapt the video quality, regardless of whether adaptation is performed using a scalable codec [94, 143], transcoding [61] or multi-quality segments [127, 167, 51, 121], we propose to use a GPS-based bandwidth lookup service with a predictive video quality selection algorithm, i.e., similar to have been done by Wac et al. [155], but in a completely different application scenario.

Such an idea of a geo-predictive media delivery system has recently been developed in parallel to our work [69]. The authors predict network outages and prebuffer video. However, they provide only simulation results, do not use adaptation, and do not consider route planning. In contrast, our results are based on a running prototype with months of real-life experiments (complemented by simulations) that stream video to a mobile device whose user travels along fixed commuter routes. We adapt video quality to available bandwidth and present video bitrate planning algorithms that make use of the bandwidth lookup service.

5.2 Predicting Network Conditions Based on Geographical Location

Geographical location has a great influence on network conditions experienced by mobile devices. For example, a receiver can be located on open ground near a base station, in which case the signal should be excellent, or in a tunnel or mountainous area far from a base station, where there is no connection to the network. In a mobile streaming scenario, it would be beneficial to predict future network conditions based on the current and future geographical locations. This seems especially feasible for users that view video while commuting. When commuting, it is reasonable to assume a predictable sequence of locations, and reliable public transport enables fairly good predictions of the times when those locations are visited.

The basic idea with the location-based bandwidth lookup service is to monitor a streaming session's download rate and geographical location, and to upload this data to a central service. This service enables all users to query for predicted available bandwidth for given locations based on previous observations, even without requiring that uploaders' identities are stored alongside the position information they provided. Today, many new mobile phones and other mobile devices are capable of this. A receiver with a GPS can log network conditions along with GPS data while viewing video. By uploading them periodically, every viewer helps to build a database that also other viewers can access. While moving along a particular path such as a popular commute route, the streaming application queries the bandwidth database for average bandwidths for every position along this path. These numbers are then used to predict and prepare for connectivity gaps, and to smoothen out the video quality along a route with highly fluctuating bandwidth. In a public transport scenario, a full prediction and a corresponding schedule can be calculated a priori

using the GPS-based lookup service to verify the timely progress and validate the prediction.

Making efficient use of such a data set is possible because reception quality in mobile networks is often explained by geographical location. Combining this with the fact that many travel routes are highly predictable (especially when considering commuters using public transportation), the value of such a database becomes clear. It greatly empowers algorithms for pre-buffering and early quality adaptation. The main motivation for using these algorithms is to provide a more stable video viewing experience: planned quality adaptation can lead to more gentle quality switching, and bandwidth drops can be canceled out by pre-buffering during high-bandwidth parts of the trip.

5.3 Video Streaming with a GPS-based Bandwidth-Lookup Service

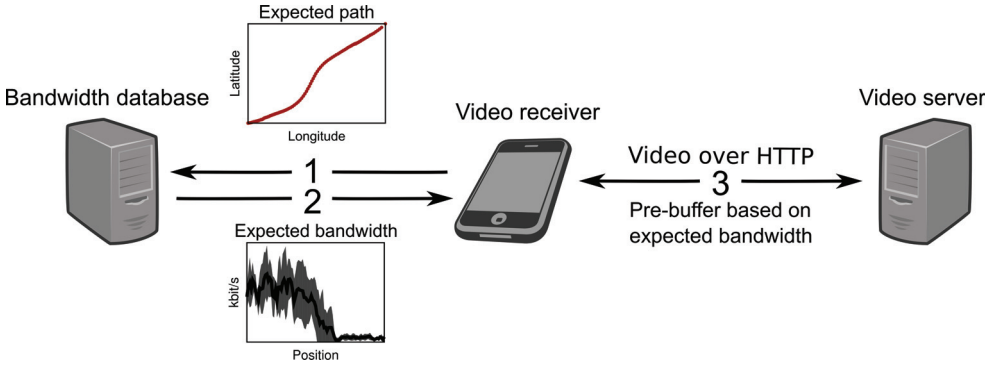


Figure 5.1: System architecture.

The architecture of our GPS-based bandwidth-lookup service is illustrated in figure 5.1, and during a streaming session, the following three steps are performed:

1. The receiver chooses a destination and an appropriate route. It sends the route to the bandwidth database, providing the latitude and longitude value for every 100 meters along the path. Using 32 bits for each value (more precision than required by the GPS' accuracy), the path description requires only eight bytes per 100 meters, so it is straight-forward to send the entire path. In an actual product, each receiver would record its own set of travel routes, and path predictions would be based on this personal data set. In other words, privacy sensitive travel logs need not be shared with other people.

2. Once the bandwidth database receives the path description, it returns a sequence of bandwidth samples for each point listed in the path description.

Each bandwidth sample represents the average bandwidth measured at that point. It is sufficient to use 16-bit integers, so the historic bandwidth data requires only two bytes per 100 meters.

3. During operation, the receiver uses previously recorded position logs and bandwidth measurements to calculate the estimated number of bytes that it can download over the remaining time of the (predicted) trip. Equipped with bandwidth prediction, path and speed prediction, and knowing the average bitrate of each quality level, the receiver can plan which quality levels to use, and start downloading. For every downloaded segment, the receiver repeats this calculation, taking the number of bytes already in the buffer into account. The goal is to have an empty buffer at the end of the trip, while only going up[2] in quality, while at the same time minimizing rapid fluctuations in quality and avoiding buffer underrun. When the receiver detects that it deviates from the predicted route or timeline, it makes a new path prediction, and fetches a new set of bandwidth predictions.

Using this approach, it is possible to predict the amount of data that can be downloaded in the future, and thus, an outage in the network signal does not need to cause an interruption: The effects of short losses of network connection as well as accurately predicted long connection losses can be "smoothed" out over time.

5.3.1 Test System

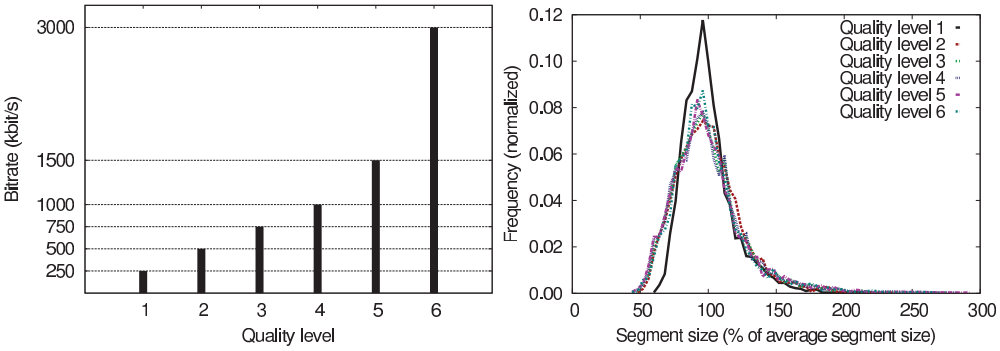
Testing of the proposed solution in an adaptive segmented HTTP streaming scenario was done using an enhanced version of Netview's proprietary media player and media downloader. This system operates in a similar way to the other modern HTTP streaming systems from Adobe, Apple and Microsoft, and it supports several adaptive streaming formats. For this test, we chose to use a modified version of Apple's HLS format. We used the same playlist format, but to save space for the lower bitrate quality levels, we used Netview's own multipexing container [138] instead of the significantly less efficient MPEG-2 Transport Stream format typically used with Apple HLS.

For content, we used European football (soccer) matches encoded in six quality levels (see figure 5.2(a)), all using a fixed segment length of two seconds and 25 frames per second. Quality levels range from low-resolution/low-bitrate video suitable for old mobile phones to HD quality for laptops with big displays. We plan long-term and do not need to adapt quickly. Thus, we do not have need for H.264/SVC [146], which is not available on most mobile devices today anyway. Akamai recommends [52] that video for 3G networks should be encoded at 250 kbit/s for

[2]We aim for an increasing quality, because 1) users strongly prefer this to starting high and lowering the quality towards the end [169], 2) starting low reduces the startup time [101] and 3) being careful in the beginning helps build a buffer faster, making the session less prone to buffer underruns.

low quality and 450 kbit/s for high quality, i.e., very close to our chosen quality levels (the reason for also having bitrates far above 450 kbit/s is that our configuration does not target *just* small portable devices, but also laptops and tablet computers that benefit from HD resolutions).

Figure 5.2(b) shows the variations in the segment sizes for all the six quality levels. It is clear that the bitrate for each quality level is not constant across the entire stream, but the deviation from the average is small, so using the average bitrate for predictions is not a problem.



(a) Quality levels and corresponding bitrates.

(b) Deviation from the average bitrate.

Figure 5.2: Quality levels used in the test streams.

Note that when proving the concept of our lookup service in our initial tests, and when demonstrating the effect of bandwidth prediction, we did not implement path prediction. We performed this operation by hand, manually identifying which of our chosen commute paths to use before starting the streaming session. The remaining duration of the trip was estimated based on the current position (our algorithm assumes that the travel speed at every point of the remaining part will equal the average of previously recorded trips). Developing on-the-fly path prediction and management of general, more random movement (e.g., a car with passengers that stream video) are outside the scope of this dissertation.

5.3.2 Algorithms

The described location-based bandwidth lookup service calculates and predicts the available bandwidth along a path. Obviously, reported entries in the database do not have identical position and bandwidth measurements, and recording time as well as traffic patterns vary as well. Considering this, it is up to the application to determine how the predicted bandwidth should be used. In our study, we have tested and compared different algorithms for quality adaptation. Later we also include in the comparison an omniscient algorithm that shows the “optimal” result.

One of our algorithms represents current systems (such as those compared in chapter 4) with a reactive approach (no prediction). The predictive algorithm uses our database with GPS-tagged historic bandwidth measurements to improve the quality scheduling.

5.3.3 A Buffer-Based Reactive Algorithm

The basic idea of the buffer-based reactive algorithm is to select the video bitrate (and thus quality) based on the number of seconds of video that are preloaded in the buffer. Thus, when the buffer reaches a certain size for a given level, the system is allowed to increase the quality. Similarly, when draining the buffer, the selected quality is reduced if the buffer shrinks below the threshold. See section 4.5 for a full description of how our buffer-based reactive algorithm works.

Reactive quality schedulers perform fine in many cases, but the quality varies according to the bandwidth availability, and during network outages, buffer underruns and playout interruptions are often unavoidable. The reactive algorithm we use therefore trades average quality for better protection against playout interruptions and large and frequent jumps in quality. We limit the quality level according to the resource availability, i.e., avoid selecting a higher quality level than supported by the current available bandwidth. Additionally, we scale the buffer fullness thresholds for changing quality according to the bitrate differences between quality layers, and to further avoid frequent oscillations in quality, the threshold for increasing the quality level is higher than when lowering the quality level (all of these settings and more are described in subsection 4.5.1).

5.3.4 A History-Based Prediction Algorithm

To make use of the location-based bandwidth lookup service, we use a history-based prediction algorithm that predicts future bandwidth using previously recorded bandwidth numbers. In this method, a path prediction is created where, for every 100 meters along the path, the system queries for a predicted bandwidth. For example, a very simple prediction is to simply use the average of recently logged (e.g., less than four weeks ago) bandwidth numbers within 100 meters of this position. Each time a new video segment is to be downloaded, the algorithm takes the path, travel time and bandwidth predictions, current position and buffer fill level, and calculates the highest quality level that could be used for the rest of the trip without getting a buffer underrun anywhere. This is the same algorithm as used in the omniscient scheduler, described in subsection 4.5.3. The only difference is that here, we do not know the duration of the streaming session and the available bandwidth for every part of it, we must instead try to predict it based on historical observations.

In many cases, we have observed that even our simplest predictive algorithm (that only uses the average bandwidth as its prediction) performs well. However, our

experiments show that there are several factors that may influence the results in a real environment, such as accuracy of the GPS signal, competition for bandwidth, base station failure, unpredicted outages and traveling delays. In other words, there are situations where the prediction fails and buffer underruns still occur. Simply picking the average bandwidth for all predictions and trusting that blindly often leads to catastrophic failures as seen in figure 5.3. This figure shows what happens when the actual bandwidth is lower than the predicted bandwidth, and the quality scheduler does not act when it notices that the prediction is wrong. Since it picks higher qualities than the bandwidth can support, buffer underruns are inevitable.

To solve this issue, we tried three different ways of making the quality scheduler more dynamic and resistant to unforseen events, all described below.

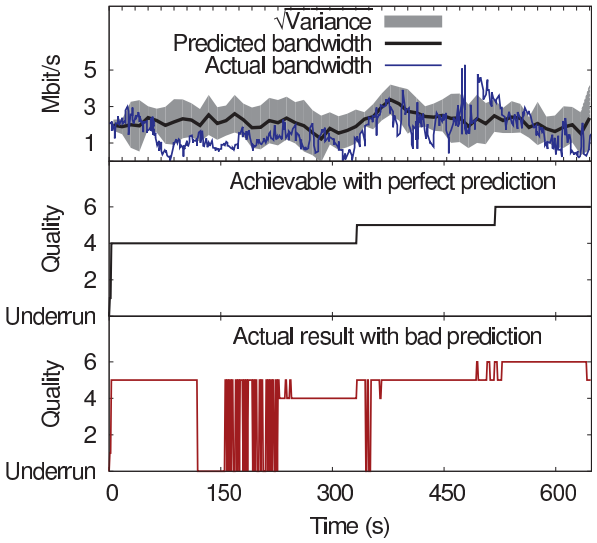


Figure 5.3: Result of a bad prediction when using a purely predictive quality scheduler.

Variance Scheduler

The *Variance scheduler* is a purely predictive history-based quality scheduling algorithm that is almost identical to our initial approach of only having a fixed prediction where, for every second along the trip, the bandwidth prediction is the average measured bandwidth at the location predicted at that time. The only difference is that the variances of the bandwidth measurements are also fetched from the bandwidth database, because they tells us how likely it is that the observed bandwidth will deviate from the average. When making a prediction, our average bandwidth values are adjusted as follows:

$$\text{PredictedBandwidth}(\textit{position}) = \langle \text{Bandwidth}(\textit{position}) \rangle + p \cdot \sqrt{\text{Variance}(\textit{position})}$$

Here, the optimism parameter p can be negative to get a more pessimistic prediction, and positive for an optimistic estimate. Letting the adjustment value depend

on the variance in bandwidth has the benefit that the correction value is greater in the bandwidth samples that are most uncertain. As we progress in our streaming session, we can adjust the optimism parameter to make our predictions more in line with the observations seen so far, hoping that they also indicate this run's performance for the remainder of the trip.

Monte-Carlo Scheduler

A purely predictive history-based quality scheduling algorithm which we call the *Monte-Carlo scheduler*, predicts bandwidth using a Monte-Carlo method based on recorded values. This method takes a trip prediction (includes both positions and expected arrival times at said positions) and generates N different hypothetical runs ($N = 2000$ in our simulations) along our predicted trip. For every second t of the predicted trip, we calculate the expected position (x, y) and use our bandwidth lookup service to fetch every bandwidth sample within a 100 meter radius of (x, y). From this set of bandwidth samples, we *randomly* pick a single sample, and use it as the bandwidth value for time t in the hypothetical run.

Once this is done for all N hypothetical runs, the Monte-Carlo scheduler simulates a streaming session for all N runs, and sorts them by the average quality they could support without underrun, and only one of them is picked as our prediction. Which one is picked depends on our optimism parameter p, where $0 \leq p \leq 1$, and $p = 0$ picks the most poorly performing run for a *very* pessimistic prediction, $p = 0.5$ picks the median, and $p = 1$ picks the best performing run. Because every bandwidth sample that constitute a hypothetical run was picked randomly amongst a set of real runs, the optimism parameter in the Monte-Carlo scheduler will, like in the Variance Scheduler, also most strongly affect the locations where the bandwidth is most unpredictable.

The main problem with the Monte-Carlo prediction is that it is computationally expensive. Every time we ask the quality scheduler for a suggested quality level, it has to generate N hypothetical runs starting from the current location, simulate a streaming session for each of them to evaluate their performance, and then sort them according to performance. Especially the simulation part takes time, making this an extremely CPU-intensive quality scheduler.

Hybrid Reactive/Predictive Scheduler

This approach combines the simple predictive scheduler that uses the average bandwidth at every 100 meters as the prediction with a reactive quality scheduler very similar to the one described in section 4.5. When quality scheduler is asked for a suggested quality level, it generates a suggestion using both the reactive and the predictive algorithms. The lowest of those two is then used, effectively making the reactive scheduler a "safety net" to protect against failed predictions.

The reactive scheduler used here is almost the same as the one presented in section 4.5, but with one important difference: We do not limit quality levels to the currently observed bandwidth, because it interferes with the long term planning required to smooth out variations in bandwidth.

Omniscient Scheduler

To compare our results to an optimal scenario, we also included the omniscient prediction algorithm from subsection 4.5.3. Unlike the above methods, this a posteriori algorithm has all measurements for a particular test run available. It chooses bandwidths in the same way as described in the predictive algorithm above: For every segment, the maximum quality level that could be supported for the rest of the trip is used. The difference is that the omniscient algorithm always has a perfect prediction, so it never needs to reduce quality, and consumes its buffers completely.

Note that, while ending with empty buffers is the optimal result, we reduced the importance of this goal for the other non-omniscient predictive schedulers to take variable-duration trips into account. An omniscient algorithm knows *exactly* how long a trip takes and *exactly* the amount of available bandwidth at all times, and can safely end the streaming session with empty buffers and maximum average quality, but this is impossible in reality. Thus, when nearing the end of the trip, the non-omniscient schedulers should not increase the quality further simply for the sake of achieving empty buffers at the exact end of the trip. Instead, use the extra bandwidth to fill the buffer, thus widening the safety margins. This is important because a real implementation that tries to end with empty buffers takes a great

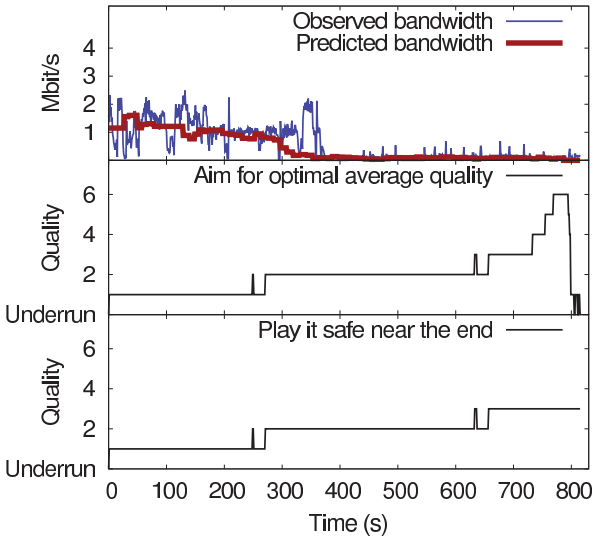


Figure 5.4: Limit aggressiveness at the end of the trip.

risk, as the margins are very small near the end, and any deviation (delay) from the predicted trip can cause a buffer underrun. Our metro test case demonstrates this, as shown in figure 5.4. What happens near the predicted end of the trip is that the algorithm ramps up quality dramatically to drain its buffers, in an effort to maximize average quality. When a delay occurs inside the tunnel (where the available bandwidth is almost non-existent) the buffers run dry before the trip's end, and we are unable to recover due to the lack of bandwidth. This demonstrates an unfortunate special case that had to be addressed. As a countermeasure, we relax the buffer drain requirements near the end of the trip. When most of the trip is done, there is little point in maxing out the quality just for the sake of optimization, so we limit the quality to the maximum achieved at 85 % of the trip's estimated duration. The result is that our buffers contain more unused data at the trip's end, but we are better protected from forced bandwidth drain at bandwidth-poor locations near the end.

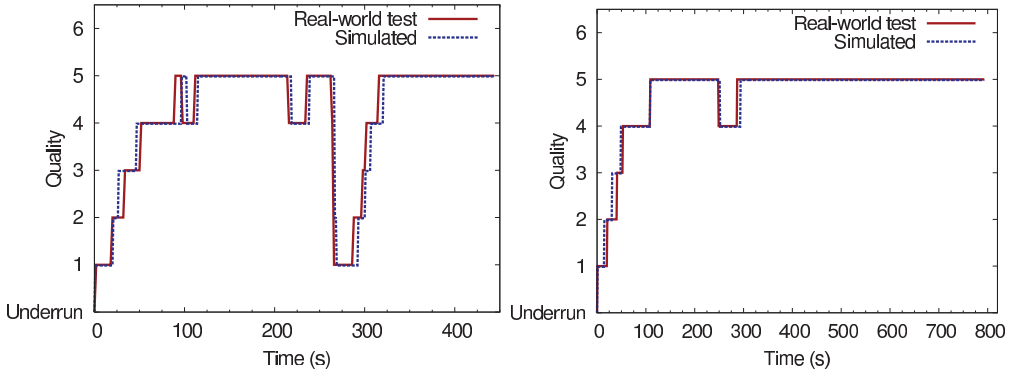
5.3.5 Comparing Quality Schedulers Through Simulation

Performing real streaming experiments in a live, mobile environment to test all possible parameters of various different algorithms would be extremely time consuming, so we started by creating a network and streaming simulator. This simulator would take as input a bandwidth trace acquired from a real-world streaming session, and simulate streaming under those conditions with a different quality scheduling algorithm. The simulation also took into account the bitrate of the video stream (configured as shown in figure 5.2), even including variations over time (i.e., not just the average bitrate). Figure 5.5 compares the real-world quality scheduler result with the simulated result. It demonstrates that our simulator is remarkably accurate, its result overlapping almost exactly with the real-world result.

Not only did the simulator make it possible to accurately test in seconds what would have taken hours in real-world tests, but it also enabled us to compare different quality selection algorithms on the same bandwidth trace and compare the results directly. This would have been impossible in real-world tests because one will never get exactly the same bandwidth curves on two different runs.

5.3.6 A Comparison of Different Predictive Schedulers

Both the Monte-Carlo and the Variance schedulers are pure predictive schedulers that adapt to failed predictions through the use of an "optimism" parameter p. Because they are similar in this respect, it makes sense to exclude one of them early on. Figure 5.6 shows that the results are almost identical when using comparable optimism values. Since the Monte-Carlo scheduler is extremely expensive computationally and provides no advantages over the Variance scheduler, it will not be discussed any further.



(a) Reactive algorithm (b) Predictive algorithm

Figure 5.5: Testing the accuracy of the simulations by comparing real-world tests of algorithms with the simulation (using the bandwidth measured in the real test to make them comparable)

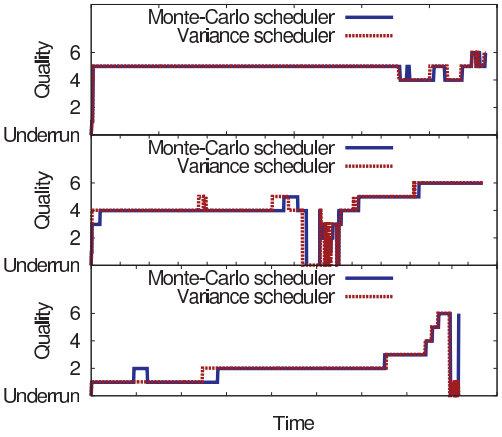


Figure 5.6: The Monte-Carlo and Variance schedulers give almost identical results when both are set to medium optimism, but Monte-Carlo is far more expensive computationally.

The purpose of adding the optimism parameter to the history-based prediction scheduler was to make it auto-adjust when the observed bandwidth deviates from the prediction. In particular, we want to avoid the repeated buffer underruns that usually occur in such cases (illustrated in figure 5.3). This turned out to be almost impossible in the general case. To avoid buffer underruns when the prediction failed by a large degree, the optimism parameter had to drop extremely fast. This means that the achieved video quality was much lower than the network could actually support. Attempting to restore the optimism value after a prediction failure was almost without exception a mistake, as buffer-underruns would occur soon after the optimism was restored. Thus we concluded that it was impossible to conjure up a working bandwidth prediction after the original prediction turned out to be

false, and thus we should not be using a predictive algorithm in such scenarios. We found our Hybrid Reactive/Predictive scheduler to be vastly superior to the Variance scheduler for unpredictable runs, while identical in performance when the prediction is good. Thus, for the rest of the dissertation, the term *predictive scheduler* will refer to the Hybrid Reactive/Predictive scheduler described in subsection 5.3.4.

5.3.7 Evaluation of the Predictive Scheduler

We present in figure 5.7 comparisons of the reactive buffer algorithm, the predictive algorithm and the omniscient algorithm for four different bandwidth logs (one for each of our chosen commute routes). The first observation is that the traditional re-

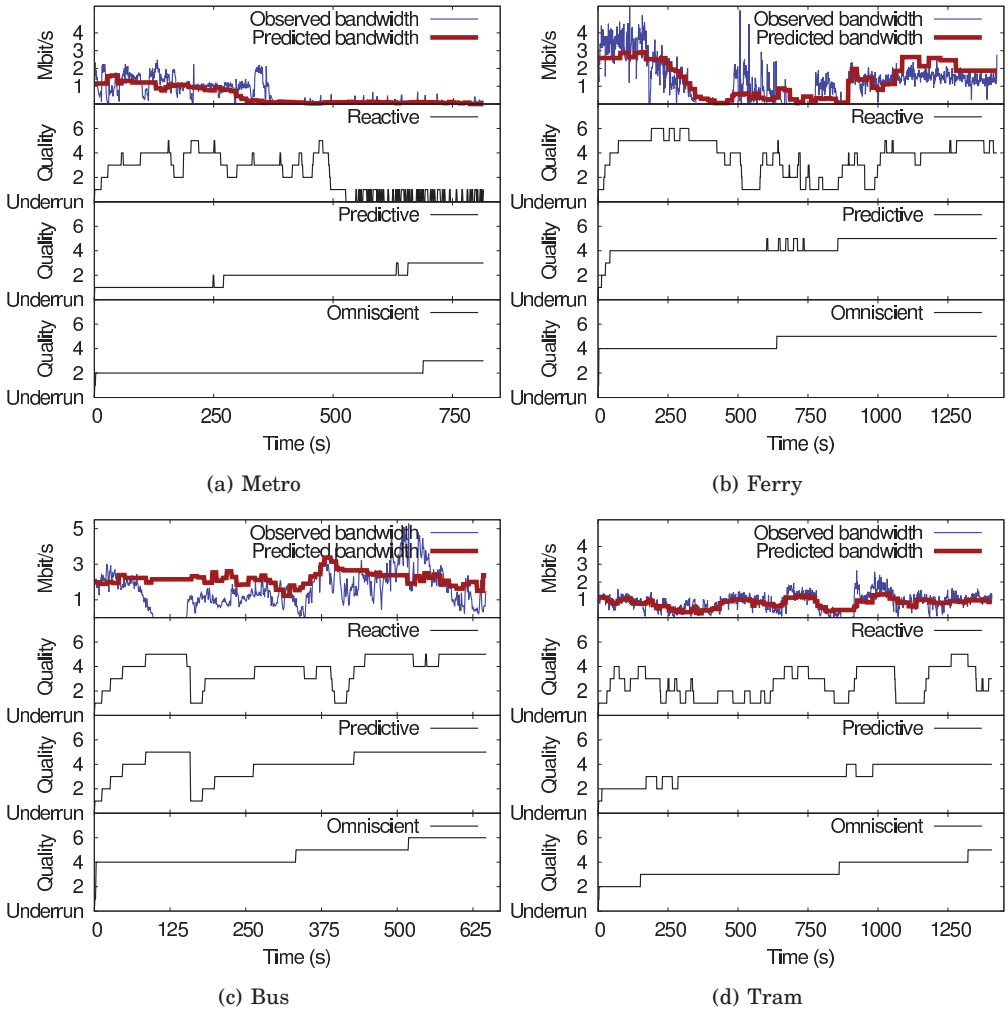


Figure 5.7: Video quality levels for the reactive, prediction and omniscient algorithms.

active algorithm follows the available bandwidth fairly well – the video quality rises and falls according to the available bandwidth (but is delayed in playout because of buffering).

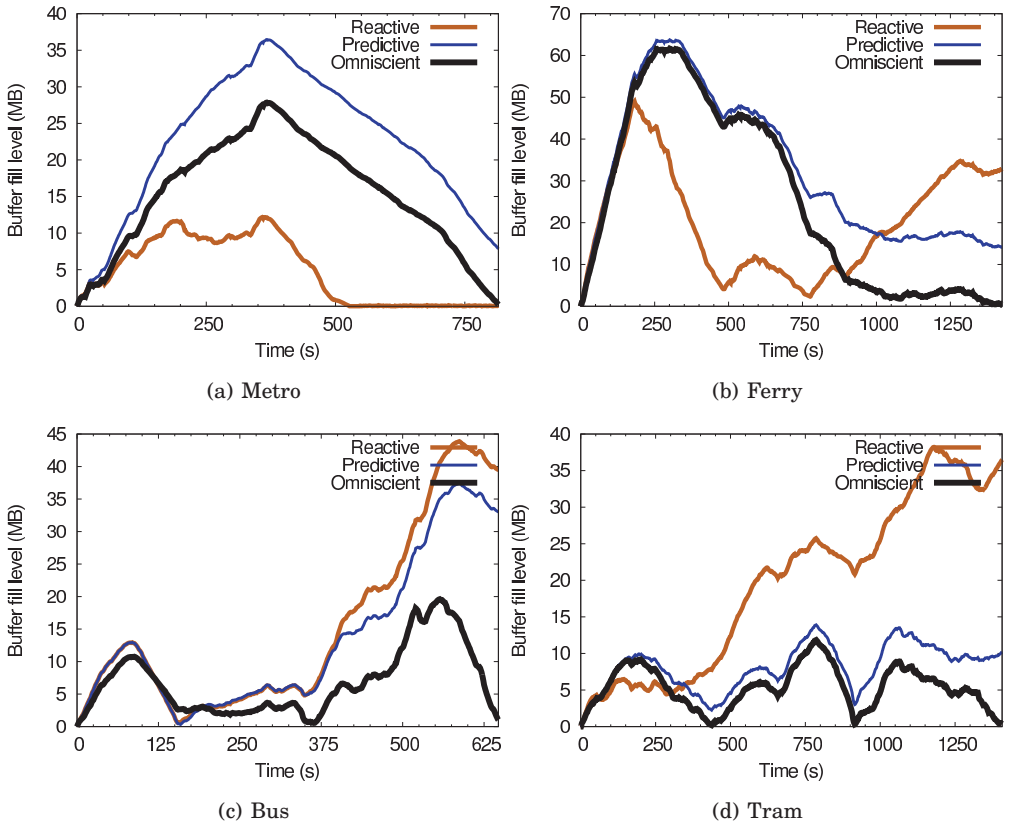
Additionally, the results show that, while the reactive scheduler's bandwidth limiting, step size scaling and different up/down thresholds are able to deal with small network outages, the quality is still often unstable. For longer periods without network connection, as demonstrated in the metro scenario, the reactive algorithm fails to provide continuous video playback. We see that the predictive algorithm is better at handling network variations and outages. The video quality is smoothed out as the algorithm can use the predicted bandwidth in the calculations, and even large outages (caused, for example, by tunnels or simply being far from the closest base station) can be foreseen and managed. For example, in the metro tunnel, the reactive algorithm drains the buffer quickly, and the video playout stops approximately 300 seconds before arrival. The predictive algorithm calculates the average possible quality in advance, and by sacrificing quality at the beginning for prefetching bandwidth, the video playout lasts for the whole 820 seconds of the trip (half of which is in a tunnel). From the results, we observe that the predictive scheduler performs close to the omniscient scheduler when the prediction is good, and always outperforms the reactive scheduler.

Furthermore, for these tests, we present the buffer fullness as a function of time in figure 5.8. In the plots, we can see that the omniscient algorithm always ends with an empty buffer due to its perfect prediction, while the less precise real-world algorithms have more in the buffer at the trip's end. They do not stop downloading video until the player is shut down, and the playout position may be far behind the download position when the bandwidth prediction is below the actual bandwidth, i.e., the under-prediction is used to download video data beyond the estimated arrival time. Nevertheless, we observe again that the predictive algorithm follows the omniscient algorithm better than the reactive algorithm.

To evaluate the accuracy of our simulations, we first ran a real streaming session and then a simulation on the bandwidth log recorded in the real session. The results with respect to the quality level as a function of time are shown in figure 5.5. We can observe that the real and the simulated systems make almost exactly the same decisions.

5.3.8 Real World Video Streaming Tests

In the previous subsection, we used simulations to show the effectiveness of the prediction model. In this section, we show the results from real streaming sessions in the real environment for our tested commute paths using the bus, the metro, the ferry and the tram as shown in figures 3.7(a), 3.8(a), 3.10(a) and 3.11(a). In figure 5.10, we present the results where the graphs on the left show the result using a traditional reactive buffer based algorithm, while the graphs on the right



(a) Metro

(b) Ferry

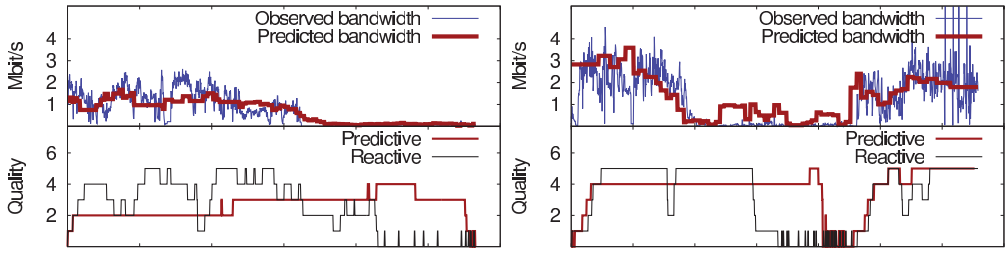
(c) Bus

(d) Tram

Figure 5.8: Buffer-fullness for the reactive, prediction and omniscient algorithms.

show the results using the GPS-based bandwidth lookup service with the predictive algorithm. Additionally, in each figure, we have also plotted the results from a simulation over the same bandwidth trace captured during the real test using the other algorithm with which we want to compare. In other words, the results should be directly comparable since we showed in the previous section that the real-world results directly map to the simulated results.

The plots in figure 5.10 show that the real-world tests confirm the simulation results. Using the bandwidth lookup service with the predictive algorithm, a user will experience far fewer buffer underruns and more stable quality. However, we also experienced that there are conditions that prevent error-free predictions. For example, we experienced buffer underruns shown in figure 5.9(a), because the metro was delayed (more than the 15 %-remaining time function could handle) in an underground station near the end of the trip. Because such delays occur very rarely, we failed to predict the duration of the trip, causing a buffer underrun without any way to recover. Another example is shown in figure 5.9(b), where the signal was almost completely gone for several minutes, at a location where previous mea-



(a) Metro: Delay in area with no bandwidth

(b) Ferry: Long, unpredicted network outage

Figure 5.9: Examples of failed predictions.

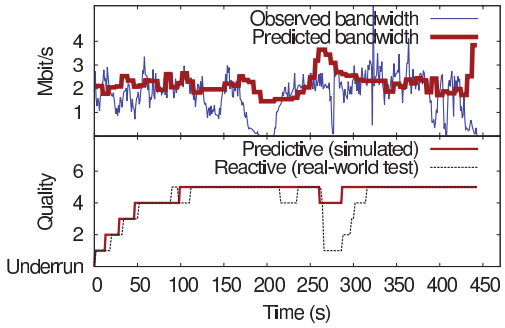
surements found acceptable network conditions (note the differences between the observed and the predicted bandwidth). Nevertheless, the real-world experiments show that the proposed system is well suited for video streaming in bandwidth fluctuating vehicular mobility scenarios.

5.4 Summary

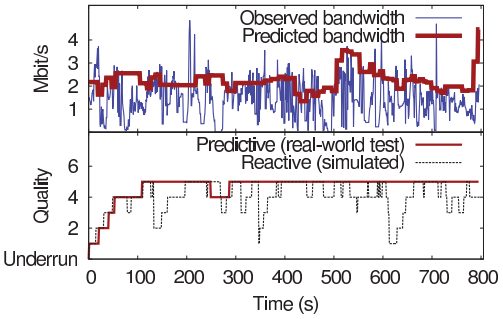
In this chapter, we have presented a GPS-based bandwidth-lookup service that is used for bitrate planning in a video streaming scenario. During a streaming session, the receiver's bandwidth is monitored and reported back along with the associated GPS positional data, i.e., the users themselves build up the database required for the service. Then, using such a service, the streaming application can predict future network outages and bandwidth fluctuations based on the previously measured bandwidths.

As a proof of concept, we implemented a prototype that was tested on popular commute routes using public transportation in Oslo, Norway. Along these routes, we measured the bandwidth and used the measurements to build a database for the lookup service. Then, we modified our existing adaptive HTTP streaming prototype [101] to perform bitrate prediction based on the information returned by the lookup service. Our experiments, both simulations and real-world tests, show that the experienced video quality can be smoothened and that the severe network outages can be handled. Using prediction, we were also able to get much closer to the performance of the omniscient scheduler than we got using our best reactive scheduler.

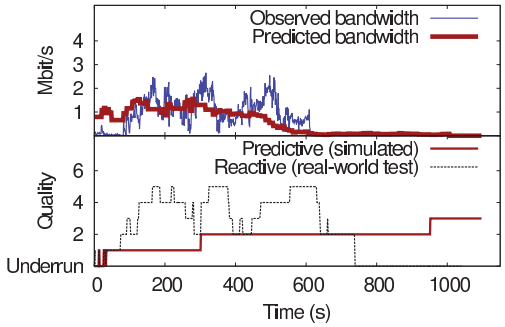
However, while performing measurements of the 3G/HSDPA network in Oslo, we often observed that other wireless networks were available. The next chapter will attempt to extend the work presented here with support for multi-link streaming in an effort to maximize video quality through the use of all available resources.



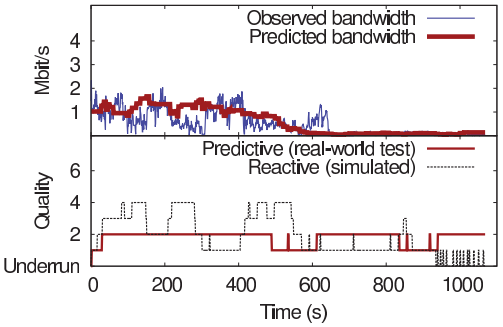
(a) Bus: Reactive (real) vs. Predictive (sim)



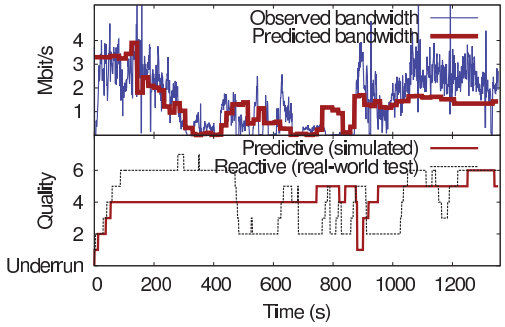
(b) Bus: Reactive (sim) vs. Predictive (real)



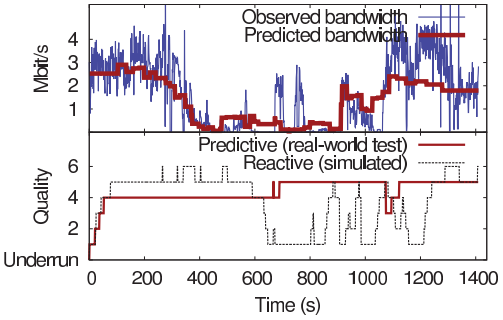
(c) Metro: Reactive (real) vs. Predictive (sim)



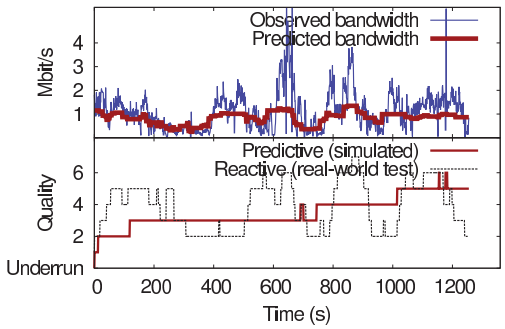
(d) Metro: Reactive (sim) vs. Predictive (real)



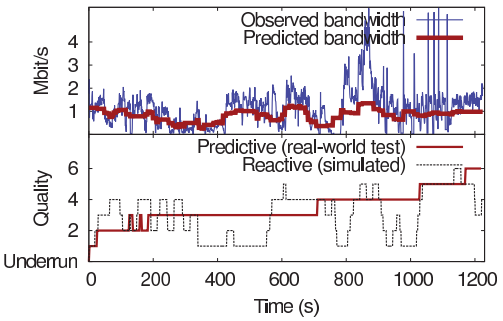
(e) Ferry: Reactive (real) vs. Predictive (sim)



(f) Ferry: Reactive (sim) vs. Predictive (real)



(g) Tram: Reactive (real) vs. Predictive (sim)



(h) Tram: Reactive (sim) vs. Predictive (real)

Figure 5.10: Real-world experiments: The Reactive vs. the Predictive scheduler.

Chapter 6

Increasing Available Bandwidth through Multi-Link Streaming

In chapter 5, we showed that adaptive video streaming in mobile wireless networks can benefit greatly from bandwidth prediction, giving fewer buffer underruns and playout interruptions, more stable quality, and high utilization of available bandwidth. However, we used only one type of wireless network while observing that others were available. This indicates that there may still be much to gain by switching between or aggregating multiple wireless networks.

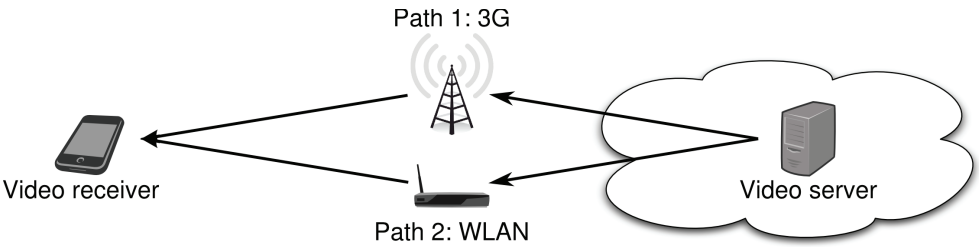


Figure 6.1: An example of multi-link streaming. In this example, we have a multihomed video receiver with two network interfaces, 3G and WLAN (i.e., the same as we use in our experiments in this chapter). Each interface has its own IP address, and thus have different routing paths. The multi-link streaming depicted here is *not transparent* to the video receiver, because it has separate connections for the two interfaces.

This chapter briefly describes how multi-link streaming (illustrated in figure 6.1) can be used together with our prediction-based adaptive streaming system from chapter 5 to further improve the quality in mobile streaming scenarios. We used a technique for transparent network roaming and connection handover [78], without requiring modifications to the operating system, or introducing new network protocols. *Transparent* in this context means that standard network applications (in our case a streaming media player) running on a multihomed device can utilize

different types of networks simultaneously, without requiring the application or the media streaming server to know about it.

Our HTTP-based streaming video system presented in chapter 5 was used to evaluate the performance of the solution described in this chapter. We present experimental results from a tram commute route in Oslo (Norway), where the multi-link framework seamlessly switched between WLAN and 3G networks. To further improve the accuracy of our bandwidth prediction system, we used Ruter's real-time information service for public transport in Oslo [35] to get information about tram arrival times and estimated travel time.

Section 6.1 lists related work, and section 6.2 describes the location-based transparent handover solution. The results from our experiments are presented and discussed in section 6.3, and we conclude in section 6.4.

6.1 Related Work

The reason video streaming to mobile devices is still an active research topic is that unpredictable bandwidth fluctuations in mobile wireless networks remains a problem. We have shown in previous chapters that bandwidth adaptation is an essential part of the solution to this problem, and there is no shortage of media streaming products and standards that incorporate this technique, or published research papers that build on bitrate adaptation. Lookup services for QoS information and prediction have also been suggested before, and several examples of related work on this topic is listed in section 5.1. However, combining bitrate adaptation with multi-link streaming is still in the early stages.

Most current operating systems support single-linked multihoming (that is, being connected to multiple networks, but never using more than one link at the same time). Multi-link streaming can be enabled using add-on software, but this unfortunately also requires changes to the applications and servers [107], or even to network infrastructure such as WLAN access points [88].

Multi-link streaming on the overlay level is a popular research topic. In [158], Wang et al. develop an algorithm for selecting paths and controlling transmission rates in the context of an overlay network where a server can send data over multiple overlay paths to its destination. An overlay network is created between two end-points, consisting of one TCP connection for each available interface. The capacity of each connection is initally based on an educated guess, and then a more accurate estimate is determined through probing. Their packet scheduler performs link selection and send-rate adjustments on each iteration in order to maximize the throughput and distribute the traffic fairly amongst the links. Wang, Wei, Guo, and Towsley present in [157] a similar solution for live video streaming applications, wherein the client opens multiple TCP connections to the server. Hence, it is not a transparent multi-link solution, but requires special logic in both the client

and the server. They also show that good video streaming quality can be expected if the aggregated throughput is roughly 1.6 times higher than the bandwidth requirement of the video. Evensen et al. also describe [78] and demonstrate [80] an overlay-level solution for video streaming, where an adaptive HTTP streaming client is extended to stream from multiple links at the same time (the same technology that is used in this chapter). The same authors have also presented [79] a similar approach for UDP-based streaming. In contrast to the systems presented by Wang et al. [157, 158], these are fully transparent multi-link solutions that work with standard client-side systems and streaming servers. Another example of an application-specific multi-link solution is given by Qureshi, Carlisle, and Guttag [133], where they introduce "Tavarua", a multimedia streaming system that uses network-striping to deliver high-bitrate video over mobile wireless networks.

A lot of research has been done on roaming clients, and some of this is related to our work. The IETF-standardized Mobile IP protocol [128, 102] enables mobile devices to move between different networks while maintaining a fixed IP address. Mähönen, Petrova, Riihijärvi, and Wellens envision a *Cognitive Resource Manager* [118] that can function as a connection manager that, among other things, decides which type of network to use. Di Caro et al. present in [65] a handover solution that is semi-transparent on the application level. However, the solution requires knowledge of the port numbers used by the application, and it relies on active probing to determine if links are available and requires user interaction to switch between links. Another example is "Wiffler" [58], a system where the main idea is to reduce monetary costs of data transfers by delaying them until a cheaper network becomes available. As an example, 3G bandwidth costs more than WLAN bandwidth, so it can make sense to wait for a WLAN connection before starting large transfers on a 3G network. Applications that send data specify their delay threshold, and a proxy process detects availability of WLAN networks and shifts as much traffic onto those as possible.

Most existing work in these areas ignore real-world complications such as those demonstrated by Kaspar et al. [105] and by ourselves [136, 139, 137]. We have taken components from various research papers cited above and incorporated all of it into a complete solution for adaptive bitrate multi-link streaming, that is able to improve performance of standard adaptive HTTP formats such as Apple HLS, Microsoft Smooth Streaming, and MPEG DASH, using bandwidth prediction and multi-link streaming. The implementation details of the multi-link system are not important within the scope of this dissertation, because we are mostly interested in the interplay between the prediction-based streaming technology we developed in chapter 5 and multi-link streaming in general. We want to confirm that our system can successfully run on top of such a system, and that the performance (defined as the QoE from the user's perspective) is improved. However, for the sake of completeness, the next section briefly describes the system architecture of our multi-link system.

6.2 System Architecture

The multi-link system that we used can be roughly separated into three main components: (1) the multi-link framework, which we call *MULTI* [81, 77], (2) the adaptive HTTP streaming client, and (3) the network and bandwidth lookup service. The MULTI framework is described in subsection 6.2.1 and the adaptive streaming client in sections 4.5 and 5.3. The network and bandwidth lookup service is described in section 5.3, but some additions for multi-network support are described in section 6.2.2. Figure 6.2 shows how these components interact, and also gives a high-level overview of how transparent roaming was implemented through the MULTI framework.

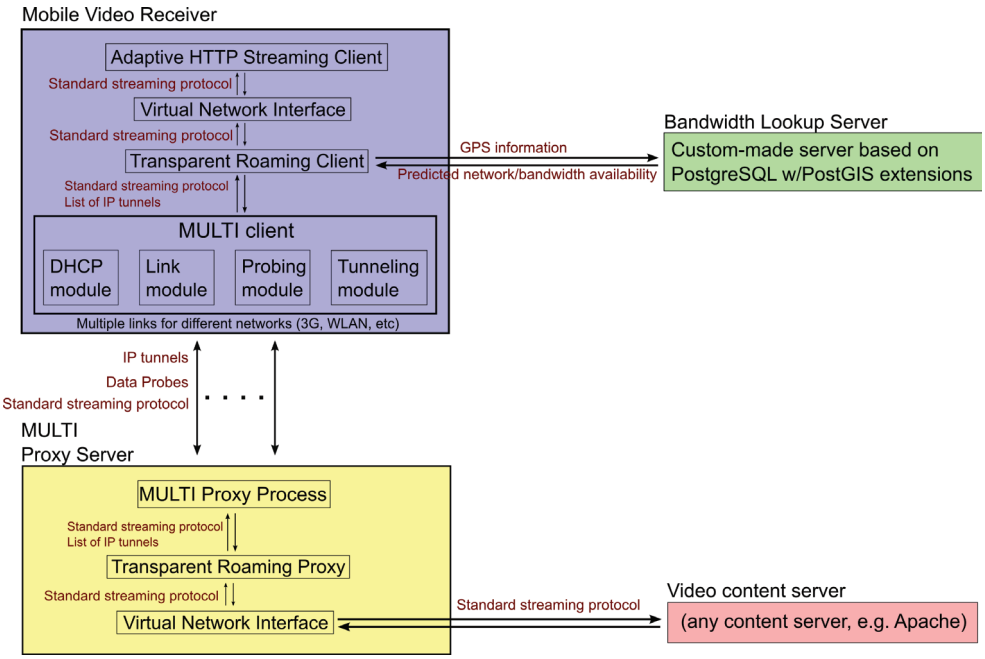


Figure 6.2: Overview of the MULTI framework for implementing transparent multi-link networking. Each colored box can represent a different machine in the network, but it is also possible to run the proxy, lookup and content servers on the same machine.

6.2.1 MULTI: A Transparent Framework for Multi-Link Streaming

The MULTI component simultaneously monitors and configures the network interfaces, updates routing tables, and enables multiple links. Here we only briefly outline the MULTI system, as we are only using MULTI to test the multi-link per-

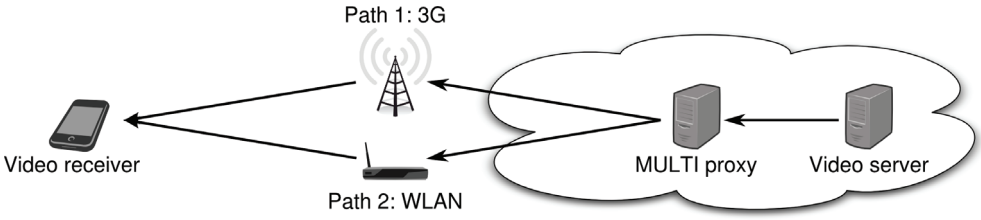


Figure 6.3: An example of multi-link streaming with transparent roaming. This differs from figure 6.1 in that the MULTI proxy (depicted as the yellow box in figure 6.2) shields the video server from having to manage multiple links per receiver. This is necessary to be able to seamlessly migrate connections in the middle of a file download without special logic in the web server.

formance of our streaming system. The implementation of MULTI is outside the scope of this dissertation; for the full description, see [81, 77].

The MULTI framework consists of several submodules, as shown in figure 6.2. Application transparency on the client side is achieved through the use of a local virtual network interface through which all video data is routed. The media player application sees only the virtual network interface, and can thus benefit from multi-link technology without being affected by its complexity.

To achieve transparency on the video server side, a proxy server is needed to hide the complexity of the video receiver's multiple links (see figure 6.3). Thus, the MULTI framework also includes components that should run outside the client, on a separate proxy server. This proxy server will be a middle-man between the mobile video receiver and the video content server. It can run on its own machine, on the same machine that hosts the video content, or be installed on the client itself. The proxy server transforms the receiver's multiple links into a single link with which the content server interacts.

6.2.2 Location-Based Network and Bandwidth Lookup for Multi-Link Streaming

The data points representing the expected throughput at different geographical locations are, as described in section 5.3, collected by the users of the video service, and the network information is stored in a database with standardized Geographic Information System (GIS) extensions for handling location-based calculations. The database used in our first prototype is PostgreSQL [31] using the PostGIS extensions [30].

All the information about the network and the performance from one measurement at a given time is stored as a single record in the database. This record includes network ID, time, GPS coordinates and observed performance metrics like bandwidth, round-trip time and packet loss rate (round-trip time and packet loss

rate is only used by MULTI, not the quality scheduling system described in 5.3). Figure 6.4(b) shows an example bandwidth trace where we differentiate between different networks (3G and WLAN in this example). With such a database, applications can use PostGIS queries that return historical bandwidth measurements based on location, network type, and age of the data points, as shown in the following PostGIS query example:

```
SELECT network_id, AVG(bandwidth)
FROM table_observed_performance
WHERE query_gps = gps AND time < 10-days-old
GROUPBY network_id
```

This returns the predicted average bandwidth for all available networks at a given GPS location based on measurements from the last 10 days. While the above information is sufficient if a user moves arbitrarily around, users often follows a given path, particularly when commuting (our scenario), which we have shown (chapter 5) can be used to perform long-term bandwidth availability prediction. Our database therefore defines a table for known paths, such as well-known commute routes in Oslo, returning a list of GPS coordinates and the respective time spent at given locations (for example within the vicinity of a GPS-coordinate or at a station). Using PostGIS queries, like the one shown above, for multiple points along the expected path, the media downloader can calculate the predicted amount of data that can be downloaded. As in chapter 5, it uses this information to fill gaps in the stream that are caused by expected network outages, and it can do long-term planning of quality adaptation for a more stable video quality.

Information about the network provider of a given network ID is kept in a separate table, and can be used to look up other relevant data such as pricing. Although this parameter is not taken into consideration by our prototype, it could be added to enable users to optimize for metrics like monetary bandwidth cost, rather than performance.

6.2.3 Video Streaming and Quality Scheduling

Our video streaming and quality scheduling component is the same as that described in chapter 5, except that the bandwidth prediction data in the lookup service now contains an extra field that identifies the network that was used for each bandwidth sample.

Connection handover and link selection is currently performed transparently by the roaming client and proxy. It can happen that the client is forced to choose another network than the one with the highest average bandwidth (which the quality scheduler assumed we would use in its prediction). For example, this could happen if the best network is WLAN, but the network password has changed, preventing

the connection from succeeding. If something this happens, it means that the prediction will be too optimistic, and the reactive quality scheduler will have to rescue the predictive scheduler from buffer underrun (see the description of the hybrid reactive/predictive scheduler in subsection 5.3.4), in effect making the video quality fluctuate more rapidly than it should. This state is however corrected rapidly by telling the quality scheduler that we are using a slower network than predicted, and that the prediction must be recalculated with this in mind.

6.3 Experiments and Results

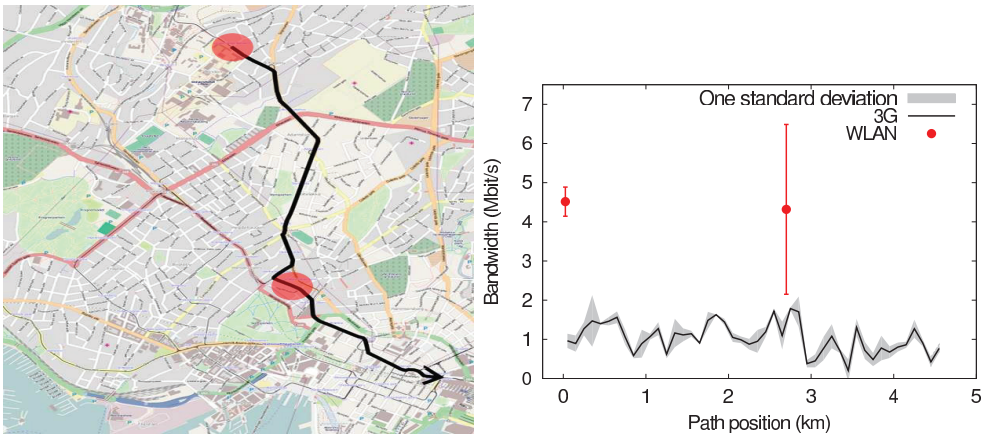
To test our proposed solution where the predictive quality scheduler from chapter 5 is extended with multi-link support, we have again performed real-world experiments on a tram commute route in Oslo (figure 6.4(a)) with different available networks (figure 6.4(b)). The results presented here serve as a proof of concept.

6.3.1 Test Scenario

Our real-world experiments were performed on a tram commute route between the University of Oslo (Blindern) and central Oslo, as shown in figure 6.4(a). Several networks with varying network availability depending on location are available along this route. In our tests, the client was able to connect to Telenor's 3G network at every position along the route, and the *eduroam* WLAN [16] was available at the tram station near Blindern, and outside the Oslo University College closer to downtown Oslo. The predicted available bandwidths as a function of the distance from the start of the route are shown in figure 6.4(b). We observe that the WLAN has high bandwidth (compared to 3G), but it has a very limited coverage, whereas the 3G network has lower bandwidth but is always available. Because WLAN is unsuitable for mobile streaming, we only got a good connection while waiting for the tram. Thus, we started the streaming session while we were still sitting on the terminal waiting. To predict the duration of waiting time (with a good WLAN connection), we used Ruter's online traffic information systems [35] to estimate the time until the arrival of the next tram. As test content, we used the same streams as those described in subsection 5.3.1 and figure 5.2.

6.3.2 Results

We performed three different sets of experiments: 1) 3G only, 2) switching between WLAN and 3G and 3) aggregating the WLAN and 3G bandwidths. For all tests, the performance of both video quality schedulers was evaluated. The first two sets of experiments were performed on the tram. Because any two runs will experience different conditions (for example, available bandwidths differ and the total travel



(a) Tram commute path. The areas highlighted in red indicate WLAN zones that we could use.

(b) Networks and bandwidths.

Figure 6.4: Map and observed resource availability.

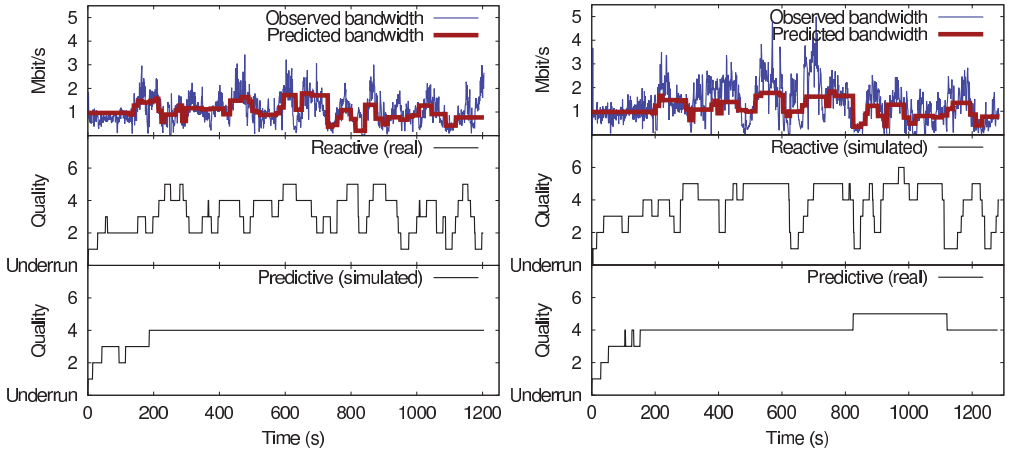
time varies due to traffic), results using the other scheduler were simulated and plotted based on the observed bandwidth curves. Thus, the real-world performance of the reactive scheduler was compared with simulated performance of the predictive scheduler, and vice versa. This was done to get directly comparable results. Our bandwidth aggregation results were obtained through simulations.

3G Only

3G was used as our base case, and an average bandwidth of about 1000 kbps was observed. The 3G network was available for the entire trip, and it provided stable performance at all times of day. Thus, our prediction algorithm was able to improve the video quality significantly compared to the reactive scheduler, as shown in figure 6.5. The reactive scheduler followed the available bandwidth and gave a more unstable video quality than the predictive scheduler. With respect to the achieved quality, the video quality rarely exceeded level 4 at 750 kbps (level 5 requires about 1500 kbps).

Switching Networks (WLAN and 3G)

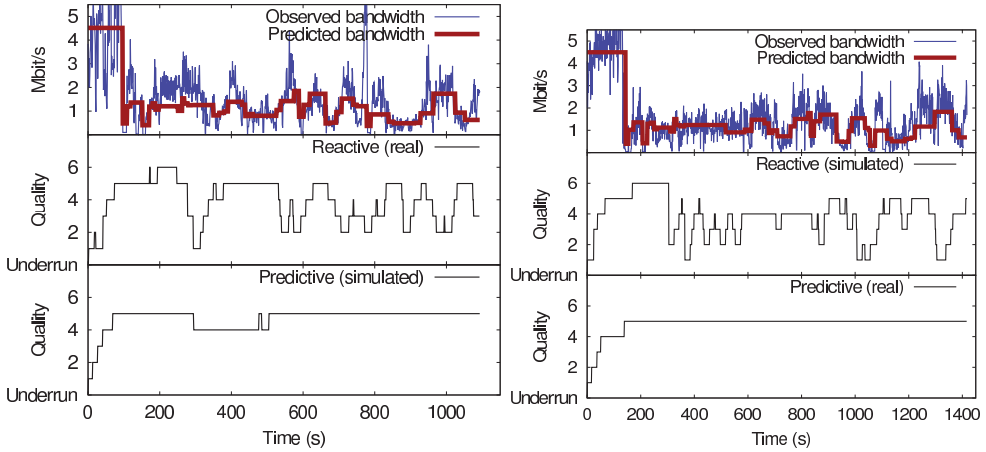
This experiment uses switching between available networks, where MULTI, at any given time, chooses the network with the highest expected bandwidth. Figure 6.6 shows the results using this approach. The eduroam WLAN always outperformed the 3G network, and was chosen whenever available. Both schedulers benefited from the increased capacity of the WLAN. The video quality was significantly higher than with 3G only (figure 6.5). With the predictive scheduler, the media player was able to stream at quality level 5 (1500 kbps) for most of the trip, compared to level 4



(a) Real-world: Reactive.

(b) Real-world: Predictive.

Figure 6.5: Streaming along the tram route while only using 3G.



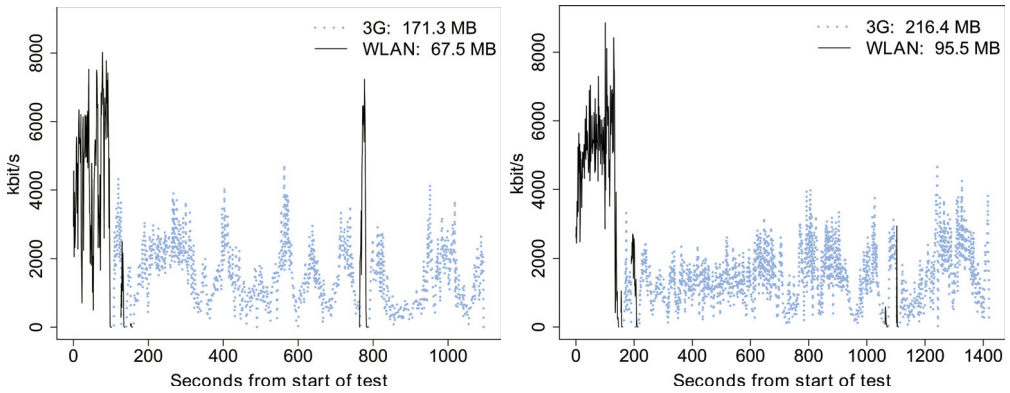
(a) Real-world: Reactive.

(b) Real-world: Predictive.

Figure 6.6: Streaming along the tram route while switching between WLAN and 3G.

(750 kbps) when only 3G was used. The reason is that the higher bandwidth of the WLAN enabled the client to receive more data. Thus, it was able to work up a bigger buffer and could request segments in a higher quality. Also, the predictive scheduler achieved a much more stable video quality than the reactive scheduler.

As we described earlier, the handover was handled transparently, and with respect to handover performance, we have plotted the throughput for the streaming sessions from figure 6.6 in figure 6.7. From the plots, we can observe that the downtime due to handover time (around 20 seconds) is small considering what we gain from switching from a 3G network that averages 1 Mbit/s to a WLAN network that



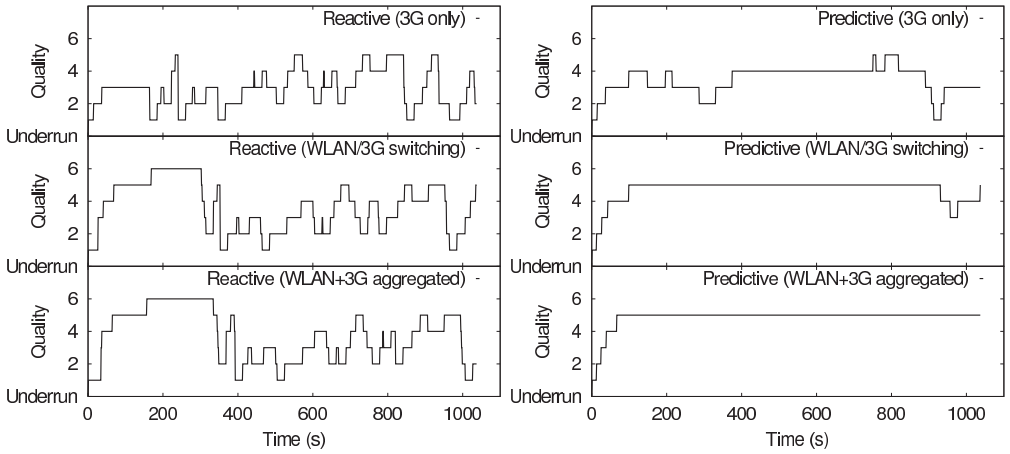
(a) Handover: Reactive.

(b) Handover: Predictive.

Figure 6.7: Achieved throughput and handovers (the megabyte numbers in the plot legend show the total number of bytes downloaded over the session for each connection).

averages 4.5 Mbit/s. However, this switch should only be done when the WLAN availability can be expected to last significantly longer than the handover downtime. This is only expected to happen if the vehicle has a predicted stop in a WLAN hotspot (this is not inconceivable, as trains have to wait around five minutes when arriving in Oslo's central train station, a place where WLAN hotspots could be available).

Aggregating networks (WLAN + 3G)



(a) Reactive

(b) Predictive

Figure 6.8: Perceived video quality: 3G only vs. WLAN/3G switching vs. WLAN+3G aggregated (all results are simulated).

To evaluate the performance of aggregating WLAN and 3G, we simulated several streaming sessions. This was done because we wanted to directly compare the results of 3G only, WLAN/3G switching, and WLAN + 3G aggregation. Thus, we needed to have the exact same conditions in every case, which is only possible in a controlled simulated environment. The simulated bandwidths of WLAN and 3G were based on traces from our real-world experiments. Because WLAN was only available during the first minute (at the tram station near Blindern) and later when passing the Oslo University College, the available bandwidth was most of the time equal to that of 3G.

The results from one representative set of simulations is shown in figure 6.8. As expected, the performance improved when more bandwidth was available. For example, when the client could aggregate WLAN and 3G bandwidths, the predictive scheduler was able to avoid a dip in quality towards the end. Also, the additional bandwidth made the video quality increase more rapidly on startup.

6.4 Summary

In this chapter, we explored if our prediction-based adaptive streaming system from chapter 5 could be further improved through the use of transparent multi-link streaming. The results of our experiments (presented in subsection 6.3.2) clearly show that this is possible on routes where multiple networks are available – even without aggregation, when just switching between networks, there is a clear improvement in performance: Using multi-link streaming in combination with prediction-based adaptive streaming, we get more stable video playouts and on average higher quality than we could achieve using only the 3G network. The results were obtained by performing streaming sessions with a fully working prototype implementation on a popular commute route in Oslo.

Chapter 7

Conclusion

In the following, we summarize earlier chapters, and repeat our contributions with slightly more detail than in section 1.5 in the introductory chapter. There is also a section on future work that discusses remaining problems and ideas that were not explored. Finally, we end with a remark on the impact of our research.

7.1 Summary

The work presented in this dissertation was motivated by the recent explosion in popularity of media streaming to mobile devices. We started by noting that wireless networks will always be bandwidth limited compared to fixed networks due to background noise, limited frequency spectrum, and varying degrees of network coverage and signal strength. Consequently, applications that need to move large amounts of data in a timely manner cannot simply assume that future networks will have sufficient bandwidth at all times. It is therefore important to make the applications themselves able to cope with varying degrees of connectivity.

Next, we performed a large number of field experiments in Telenor's 3G network in Oslo, which at the time had just been deployed. Our experiments determined the characteristics of the network, studying things like packet drop patterns, latency, base station handover mechanisms, TCP throughput, and more. We observed strong fluctuations in available bandwidth, and that packet loss could be reduced by reducing the streaming bitrate (because the radio signal could use a more robust signal modulation at the expense of bandwidth). We concluded from this that adaptive bitrate streaming, which had recently emerged as a commercially viable technology, is the best solution for media streaming to mobile devices.

The next step was to compare and evaluate the adaptive streaming implementations of Adobe, Apple, and Microsoft in several challenging vehicular mobility streaming scenarios, travelling by bus, tram, ferry and metro while trying to stream video. By far the most important difference between the compared systems was the

buffering and bitrate adaptation strategies, and in this regard, we found very different priorities in the compared systems.

The buffering and bitrate adaptation policies have a big impact on QoE from the user's perspective. How much the media players are able to buffer depend on these policies, and determine if the playout is interrupted by buffer underrun or not, and how much of the available bandwidth can be utilized. The bitrate adaptation policy also determines the average quality of the playout, and if the quality is stable or not (rapidly fluctuating quality is not good for the QoE).

We observed that all compared systems showed potential for improvement with regard to QoE, so we developed our own media client with a bitrate adaptation policy especially designed for the mobile streaming use case. Our reactive quality scheduler was designed similarly to current products, using only the current download rate and the buffer fullness to determine which qualities to use. Despite using only these parameters, we were still able to develop a quality scheduler that significantly improved performance compared to existing products. This was confirmed using bandwidth traces from real-world field trials.

While collecting data on network characteristics and bandwidth variations, we found that although there were great fluctuations in connectivity when moving around in a mobile network, the quality of the connection was highly dependent on geographical location, and also quite deterministic. Thus, the next logical step was to determine if this predictability could be used to improve performance of our quality scheduler. We developed a database of historic bandwidth measurements for points along various popular commute routes in Oslo, and a new quality scheduling algorithm (the predictive quality scheduler) that used this bandwidth lookup database to predict the bandwidth at every point for the rest of the streaming session. Based on simulations, we found that the prediction-based quality scheduler was able to provide a much more stable and high-quality viewing experience. We then confirmed our simulated results in the real world using a prototype implementation with which we successfully streamed video along the same commute routes, experiencing far fewer buffer underruns, higher quality and more stable playout than our best reactive scheduler.

Finally, we extended our media client with support for multi-link streaming, and added additional network information in our bandwidth lookup database. With multi-link support enabled, we could use the extra information to switch between or aggregate different wireless networks, thus increasing the average bandwidth experienced along the trip. Again, we confirmed our results in the field using our prototype implementation.

In summary, we developed a working implementation of an adaptive bitrate media streaming client for the mobile streaming use case. It is able to perform bandwidth prediction and multi-link streaming to deliver a vastly improved viewing experience compared to present state of the art media streaming products.

7.2 Contributions

The work presented in this dissertation addresses several issues in the field of mobile video streaming. Mobile receivers are troubled by fluctuating bandwidth, making it difficult to achieve satisfactory QoE in video streaming applications. We have presented several innovative solutions to the problem, where we extend existing adaptive bitrate streaming technologies with new algorithms for quality adaptation, bandwidth prediction and multiple network utilization. A fully functional prototype implementation was developed, proving the efficiency of our suggested solutions. The following list summarizes briefly our contributions to the problems stated in section 1.2:

1. *Gathering of network data from a real 3G network.* We spent a considerable amount of time collecting data on network characteristics in a real-world 3G network. We used custom made analysis tools, measuring both on the packet-level using UDP, and application-level performance using TCP. We showed that the performance of 3G networks is very sensitive to congestion and is strongly affected by geographical location. In non-congested areas, we showed that the available bandwidth can be predicted with fairly high accuracy based on geographical location. The collected data on network characteristics was successfully used to develop improved technologies for streaming under such conditions, and has been made available [140] to other researchers performing similar work.

2. *An in-depth comparison of existing commercial products.* To evaluate the performance of existing commercial products in adaptive video streaming under challenging network conditions, we performed a comprehensive set of tests using the data set collected in the above point. A bandwidth throttling module for the Apache web server was used to reproduce the bandwidth fluctutations that we observed in field testing. This ensured that all products were given a fair and realistic basis for comparison, and helped expose several weaknesses in current technologies.

3. *A better quality adaptation scheme for mobile receivers.* Knowing the weaknesses of existing adaptive video streaming products made it possible to develop a better quality adaptation scheme while still only basing quality adaptation decisions on download rate and buffer fill level, like other systems that are available on the market today. Even though no extra information was available to make better quality adaptation decisions, we were able to provide a significantly improved performance in mobile streaming scenarios, resulting in fewer buffer underruns and more stable quality.

4. *Demonstrating that deterministic bandwidth can be used to improve performance in video streaming.* Equipped with a custom-made bandwidth prediction service based on the data set collected in the 3G network measurement phase of the project, we were able to extend our quality adaptation algorithm mentioned above with information about future network conditions. This information made it possible to compensate for variations in the network, averaging out the quality over time and thus greatly improving QoE for the viewer. The results were compared to the optimal result (a perfectly predicted bandwidth), and found to be surprisingly close in many cases.

5. *Demonstrating that multi-link streaming is a feasible way to improve performance in an adaptive video streaming client.* We showed that using multiple different wireless networks at the same time could further improve QoE in an adaptive bitrate media client by increasing the average network capacity available for video streaming. Our experimental media client was combined with a software module that provides application-transparent multi-link streaming, and the bandwidth adaptation scheme was the same as before (our best predictive scheduler), but we extended the bandwidth database with information about different networks. In effect, what was achieved was increasing the available bandwidth, thus increasing the quality in a streaming session.

In summary, our contributions include field experiments that exposed the behavior (packet loss patterns, congestion handling, TCP throughput, predictability of bandwidth based on location, etc.) of 3G/HSDPA networks, a comparison of commercial streaming systems in this network, and a new and greatly improved buffering and bitrate adaptation policy that takes advantage of everything we learned from doing field experiments. The technology was implemented in a fully functional prototype implementation that has been successfully used to stream video in real mobile networks.

7.3 Future Work

While we have developed a fully functional prototype that demonstrates that bandwidth prediction and multi-link streaming can greatly improve performance of adaptive bitrate media clients in mobile streaming scenarios, there are still many parts of the system that need further improvements and open issues that must be solved before such a service is made available to the public.

For example, guaranteeing the consistency of the database of measurements is difficult. As shown in section 3.3, signals may be lost, positions may be inaccurate and the measurements may be corrupted. In our prototype, we have not filtered measurements, but the lookup service should do this.

Moreover, a related issue is that the current lookup prototype returns the predicted bandwidth, but the current competition for bandwidth (number of concurrent users) is not taken into account. Thus, the available bandwidth predicted by the lookup service might be shared transparently. This situation will, however, be improved by allowing the system to return different statistics according to day-of-week and time-of-day as we describe above, where such concurrent use over time is automatically reflected in the predicted bandwidth.

Another problem that is not yet addressed is automatic path prediction and path predictions that fail. In this dissertation, we manually selected the path when starting a streaming session, hence only the travel time was uncertain. In a real product, each client should maintain a set of travel paths. When starting a streaming session, it should monitor how the user moves around and compare it to previously recorded routes. Time of day and day of week can be important factors to correctly predict the route (e.g., commute routes are more likely on workdays than during weekends). Partially overlapping paths that fork can be a common cause of path prediction failures, but this is easily detected, and a new path prediction can be generated in these events.

As the users move around both randomly and along predefined paths, at different speeds, etc., a question that arises is how far into the future we should try to predict. In our experiments, we have assumed public transportation where the route and the approximate speed (timing) are known. However, the idea can be applied perfectly well to more individual means of transportation such as cars, whose direction and speed are unpredictable. Such scenarios are outside the scope of this dissertation, but they raise several challenges that should be addressed.

We have chosen to demonstrate predictive quality scheduling using adaptive HTTP streaming with multiple H.264/AVC versions of a segment as shown in figure 5.2(a). The reason is of course the current popularity of this streaming solution, the availability of Netview's media player, and the DAVVI system [101] to perform real-world streaming tests. However, this also means that when a decision is made with respect to the video quality of a segment, one must either stick with the chosen quality for the duration of the segment or re-download the complete segment in another quality, possibly wasting bandwidth. This makes quality scheduling more vulnerable to "unpredicted" variations. Nevertheless, there are no restrictions on the lookup service that prevents it from being applied with other adaptive streaming solutions. For example, using a scalable codec like SVC and building up playout windows enhancing the video quality layer by layer until the resources are consumed (like PALS [134] or priority progress streaming [94]) would, with respect to the quality scheduling, give a simpler and possibly a more efficient system (the inherent overhead of scalable codecs notwithstanding).

Finally, we also note that while GPS devices were used to monitor the receiver's location in our prototype, the level of precision delivered by GPS devices is not re-

quired for the bandwidth prediction to work. The disadvantage of using GPS is increased power consumption, a big issue for mobile devices. A less power hungry localization method that should be explored is multilateration of radio signals between several radio towers [63].

7.4 Final Remark

We have, in the work done in the context of this dissertation, tried to strike the right balance between academic research and the development of commercially viable technology. We believe that we have succeeded with this goal: Our data sets and prototypes have been frequently requested by other researchers to test their own ideas and take our work in new directions, and our media client prototype has already evolved into a mature, deployed, product. End-users are viewing video on embedded devices using this product, and the advanced quality adaptation policies are an important part of it.

Bibliography

[1] Microsoft Case Study: Microsoft Silverlight – TV 2. http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000006269, Dec. 2009.

[2] Conviva Helps Power 2010 FIFA World Cup Online Video Fan Experience. http://www.conviva.com/conviva-helps-power-2010-fifa-world-cup-online-video-fan-experience/, July 2010.

[3] Microsoft Case Study: Microsoft Silverlight – NBC Universal. http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000007258, Nov. 2010.

[4] Microsoft Case Study: Microsoft Silverlight 3 – CTV. http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=4000007347, May 2010.

[5] Envivio 4Caster C4. http://www.envivio.com/files/products/4caster-c4-datasheet.pdf, 2011.

[6] HTTP Live Streaming Overview: Using HTTP Live Streaming. http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/UsingHTTPLiveStreaming/UsingHTTPLiveStreaming.html, Apr. 2011.

[7] Netview Media Client. http://www.netview.no/index.php?page=downloader, 2011.

[8] Network time protocol (NTP). http://ntp.org/, 2011.

[9] Anevia ViaMotion. http://www.anevia.com/products/viamotion-ott/, 2012.

[10] The Apache HTTP server project. http://httpd.apache.org/, 2012.

[11] Apple – iTunes – everything you need to be entertained. http://www.apple.com/itunes/, 2012.

[12] BBC iPlayer - iPlayer TV Home. http://www.bbc.co.uk/iplayer/tv, 2012.

[13] Cisco visual networking index: Forecast and methodology, 2011-2016. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html, 2012.

[14] CodeShop Unified Streaming Platform. http://www.unified-streaming.com/, 2012.

[15] Comoyo. http://comoyo.com/, 2012.

[16] eduroam: A secure world-wide roaming access service for the international research and education community. http://eduroam.org/, 2012.

[17] ESPN Player: Watch live and on demand sports video online. http://www.espnplayer.com/espnplayer/console, 2012.

[18] ESPN Sports Search. http://search.espn.go.com/, 2012.

[19] FIFA World Cup. http://www.cbc.ca/sports/soccer/fifaworldcup/, 2012.

[20] FXPAL: TalkMiner. http://talkminer.com/, 2012.

[21] Global internet phenomena report: 1h 2012. http://www.sandvine.com/news/global_broadband_trends.asp, Apr. 2012.

[22] Hulu - Watch TV. Watch Movies. Online. Free. http://hulu.com/, 2012.

[23] Latest Rugby News – RTÉ Sport. http://www.rte.ie/sport/rugby, 2012.

[24] Media Source Extensions. http://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html, 2012.

[25] NBC Olympics. http://www.nbcolympics.com/, 2012.

[26] NBC.com – TV network for primetime, daytime and late night television shows. http://nbc.com/, 2012.

[27] Netflix – watch TV programmes online, watch films online. http://netflix.com/, 2012.

[28] ns-3 - A discrete-event network simulator for Internet systems. http://www.nsnam.org/, 2012.

[29] Olympics breaks BBC online records, with 106m video requests. http://www.televisual.com/news-detail/Olympics-breaks-BBC-online-records-with-106m-video-requests_nid-1913.html, Aug. 2012.

[30] PostGIS: Support for geographic objects in the PostgreSQL relational database. http://postgis.refractions.net/, 2012.

[31] PostgreSQL: The world's most advanced open source database. http://www.postgresql.org/, 2012.

[32] Quavlive Adaptive Streaming. http://www.quavlive.com/adaptive-streaming, 2012.

[33] RealNetworks Helix Universal Server. http://www.realnetworks.com/helix/streaming-media-server/, 2012.

[34] Record viewing figures for BBC iPlayer in 2011. http://www.bbc.co.uk/mediacentre/latestnews/2012/iplayer.html, Jan. 2012.

[35] Ruter realtime traffic information. http://ruter.no/, 2012.

[36] Sandvine report: Mobile networks teeming with streaming. http://www.sandvine.com/news/pr_detail.asp?ID=366, Apr. 2012.

[37] StreamOcean Streaming Technology. http://www.streamocean.com/en/article/Streaming_Technology, 2012.

[38] Super Bowl. http://nbcsports.msnbc.com/id/46096311/ns/sports-nfl/, 2012.

[39] tcpdump. http://www.tcpdump.org/, 2012.

[40] The GPAC Open Source multimedia framework. http://gpac.wp.mines-telecom.fr/home/, 2012.

[41] TV 2 Sumo. http://webtv.tv2.no/webtv/, 2012.

[42] TV 2 Sumo – Sporten. http://webtv.tv2.no/webtv/sumo/?treeId=2, 2012.

[43] VG Nett. http://vg.no/, 2012.

[44] VG Nett Live. http://vglive.no/, 2012.

[45] VideoLAN – Official page of the VLC media player. http://www.videolan.org/vlc/index.html, 2012.

[46] YouTube - Broadcast Yourself. http://youtube.com/, 2012.

[47] YouTube statistics. http://www.youtube.com/t/press_statistics, Nov. 2012.

[48] J. Adams and G.-M. Muntean. Power save adaptation algorithm for multimedia streaming to mobile devices. In *Proc. of the International Conference on Portable Information Devices (IEEE PORTABLE)*, pages 1–5, May 2007.

[49] M. Adams. Will HTTP adaptive streaming become the dominant mode of video delivery in cable networks? In *2012 Spring Technical Forum, National Cable and Telecommunications Assocation*, May 2012.

[50] Adobe Systems, Inc. Real-time messaging protocol (RTMP) specification. `http://www.adobe.com/devnet/rtmp.html`, Apr. 2009.

[51] Adobe Systems, Inc. HTTP dynamic streaming on the Adobe Flash platform. `http://www.adobe.com/products/httpdynamicstreaming/pdfs/httpdynamicstreaming_wp_ue.pdf`, 2010.

[52] Akamai Technologies. Akamai HD for iPhone encoding best practices. `http://www.akamai.com/dl/whitepapers/Akamai_HDNetwork_Encoding_BP_iPhone_iPad.pdf`, 2011.

[53] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. of the Annual Conference on Multimedia Systems (ACM MMSys)*, pages 157–168, Feb. 2011.

[54] J. J. Alcaraz and F. Cerdan. Performance evaluation of AQM schemes in rate-varying 3G links. In *Personal Wireless Communications*, volume 4217 of *Lecture Notes in Computer Science*, pages 310–321. 2006.

[55] Altus, Inc. Easy enterprise video tools from Altus. `http://www.altuslearning.com/altus_vsearch.php`, 2012.

[56] M. Asefi. *Quality-Driven Cross-Layer Protocols for Video Streaming over Vehicular Ad-Hoc Networks*. PhD thesis, University of Waterloo, Aug. 2011.

[57] A. Auge and J. Aspas. TCP/IP over wireless links: performance evaluation. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, volume 3, pages 1755–1759, May 1998.

[58] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3G using WiFi. In *Proc. of the International Conference on Mobile Systems, Applications, and Services (ACM MobiSys)*, pages 209–222, 2010.

[59] E. Birkedal, C. Griwodz, and P. Halvorsen. Implementation and evaluation of Late Data Choice for TCP in Linux. In *Proc. of the International Symposium on Multimedia (IEEE ISM)*, pages 221–228, Dec. 2007.

[60] A. Botta, A. Pescape, G. Ventre, E. Biersack, and S. Rugel. Performance footprints of heavy-users in 3G networks via empirical measurement. In *Proc. of the International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pages 330–335, June 2010.

[61] J. Brandt and L. Wolf. Adaptive video streaming for mobile clients. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 113–114, 2008.

[62] P. Bucciol, E. Masala, N. Kawaguchi, K. Takeda, and J. De Martin. Performance evaluation of H.264 video streaming over inter-vehicular 802.11 ad hoc networks. In *Proc. of the International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, volume 3, pages 1936–1940, Sept. 2005.

[63] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, 7(5):28–34, Oct. 2000.

[64] S. Carmel, T. Daboosh, E. Reifman, N. Shani, Z. Eliraz, D. Ginsberg, and E. Ayal. Network media streaming. *US Patent Number 6,389,473*, May 2002. Filed: Mar. 1999. Assignee: Geo Interactive Media Group Ltd.

[65] G. A. D. Caro, S. Giordano, M. Kulig, D. Lenzarini, A. Puiatti, F. Schwitter, and S. Vanini. Deployable application layer solution for seamless mobility across heterogeneous networks. *Ad Hoc & Sensor Wireless Networks*, 4(1-2):1–42, 2007.

[66] P. Chang, M. Han, and Y. Gong. Extract highlights from baseball game video with hidden Markov models. In *Proc. of the International Conference on Image Processing (ICIP)*, volume 1, pages 609–612, 2002.

[67] Y. chi Lee, J. Kim, Y. Altunbasak, and R. M. Mersereau. Performance comparisons of layered and multiple description coded video streaming over error-prone networks. In *Proc. of the International Conference on Communications (IEEE ICC)*, volume 1, pages 35–39, May 2003.

[68] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner, and P. R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, Jan. 1989.

[69] I. D. Curcio, V. K. M. Vadakital, and M. M. Hannuksela. Geo-predictive real-time media delivery in mobile environment. In *Proc. of the Annual Workshop on Mobile Video Delivery (ACM MoViD)*, pages 3–8, Oct. 2010.

[70] R. Dahyot, A. Kokaram, N. Rea, and H. Denman. Joint audio visual retrieval for tennis broadcasts. In *Proc. of the International Conference on Acoustics, Speech, and Signal Processing (IEEE ICASSP)*, volume 3, pages 561–564, Apr. 2003.

[71] L. De Cicco and S. Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *Proc. of the conference on HCI in Work and Learning, Life and Leisure: Workgroup Human-Computer Interaction and Usability Engineering (USAB)*, pages 447–464, 2010.

[72] J. Derksen, R. Jansen, M. Maijala, and E. Westerberg. HSDPA performance and evolution. *Ericsson Review No. 3*, pages 117–120, 2006.

[73] P. Deshpande, X. Hou, and S. R. Das. Performance comparison of 3G and metro-scale WiFi for vehicular network access. In *Proc. of the SIGCOMM conference on Internet measurement (ACM IMC)*, pages 301–307, 2010.

[74] A. Diaz, P. Merino, L. Panizo, and A. Recio. Evaluating video streaming over GPRS/UMTS networks: A practical case. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 624–628, Apr. 2007.

[75] R. Dicker. Super Bowl online broadcast wins more than 2.1 million e-viewers. `http://www.huffingtonpost.com/2012/02/07/super-bowl-online-broadcast_n_1260955.html`, July 2012.

[76] A. Ekin, A. Tekalp, and R. Mehrotra. Automatic soccer video analysis and summarization. *IEEE Transactions on Image Processing*, 12(7):796–807, July 2003.

[77] K. Evensen. *Aggregating the Bandwidth of Multiple Network Interfaces to Increase the Performance of Networked Applications*. PhD thesis, Simula/ University of Oslo, Jan. 2012.

[78] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, and P. Engelstad. Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks. In *Proc. of the Annual Conference on Multimedia Systems (ACM MMSys)*, pages 57–68, 2011.

[79] K. Evensen, D. Kaspar, A. F. Hansen, C. Griwodz, and P. Halvorsen. Using multiple links to increase the performance of bandwidth-intensive UDP-based applications. In *Proc. of the Symposium on Computers and Communications (IEEE ISCC)*, pages 1117–1122, 2011.

[80] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen. Demo: quality-adaptive video streaming with dynamic band-

width aggregation on roaming, multi-homed clients. In *Proc. of the International Conference on Mobile Systems, Applications, and Services (ACM MobiSys)*, pages 355–356, 2011.

[81] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen. Mobile video streaming using location-based network prediction and transparent handover. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 21–26, June 2011.

[82] F. H. P. Fitzek, B. Can, R. Prasad, and M. Katz. Overhead and quality measurements for Multiple Description Coding for video services. In *Proc. of Wireless Personal Multimedia Communications (WPMC)*, pages 524–528, 2004.

[83] M. M. J. French. A mobile phone Faraday cage. *Phys. Educ.*, 46(3):290, 2011.

[84] B. Gao, J. Jansen, P. Cesar, and D. C. Bulterman. Accurate and low-delay seeking within and across mash-ups of highly-compressed videos. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 105–110, 2011.

[85] A. Goel, C. Krasic, K. Li, and J. Walpole. Supporting low-latency TCP-based media streams. In *Proc. of the International Workshop on Quality of Service (IEEE IWQoS)*, pages 193–203, May 2002.

[86] A. Goel, C. Krasic, and J. Walpole. Low-latency adaptive streaming over TCP. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(3):1–20, Sept. 2008.

[87] V. Goyal. Multiple description coding: compression meets the network. *Signal Processing Magazine, IEEE*, 18(5):74–93, Sept. 2001.

[88] F. Guo and T.-C. Chiueh. Device-transparent network-layer handoff for micromobility. In *Proc. of the International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (IEEE MASCOTS)*, pages 1–10, Sept. 2009.

[89] M. Guo, M. H. Ammar, and E. W. Zegura. V3: A vehicle-to-vehicle live video streaming architecture. *Pervasive Mob. Comput.*, 1(4):404–424, 2005.

[90] P. Halvorsen, D. Johansen, B. Olstad, T. Kupka, and S. Tennøe. vESP: enriching enterprise document search results with aligned video summarization. In *Proc. of the International Conference on Multimedia (ACM MM)*, pages 1603–1604, 2010.

[91] H. Holma and J. Reunanen. 3GPP release 5 HSDPA measurements. In *Proc. of the International Symposium on Personal, Indoor and Mobile Radio Communications (IEEE PIMRC)*, pages 1–5, Sept. 2006.

[92] S. Horsmanheimo, H. Jormakka, and J. Lähteenmäki. Location-aided planning in mobile network — trial results. *Wirel. Pers. Commun.*, 30(2-4):207–216, Sept. 2004.

[93] C.-H. Hsu and M. Hefeeda. Achieving viewing time scalability in mobile video streaming using scalable video coding. In *Proc. of the Annual Conference on Multimedia Systems (ACM MMSys)*, pages 111–122, 2010.

[94] J. Huang, C. Krasic, J. Walpole, and W. chi Feng. Adaptive live video streaming by priority drop. In *Proc. of the Conference on Advanced Video and Signal Based Surveillance (IEEE AVSS)*, pages 342–347, July 2003.

[95] ISO/IEC 13818-1:2007. *Generic coding of moving pictures and associated audio information: Systems*.

[96] ISO/IEC 14496-10:2012. *Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding*.

[97] ISO/IEC 14496-12:2004. *Coding of audio-visual objects – Part 12: ISO base media file format*.

[98] ISO/IEC 23009-1:2012. *Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*.

[99] T. Isotalo and J. Lempiäinen. HSDPA measurements for indoor DAS. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 1127–1130, Apr. 2007.

[100] D. Johansen, P. Halvorsen, H. Johansen, H. Riiser, C. Gurrin, B. Olstad, C. Griwodz, Å. Kvalnes, J. Hurley, and T. Kupka. Search-based composition, streaming and playback of video archive content. *Multimedia Tools and Applications*, 61:419–445, 2012. 10.1007/s11042-011-0847-5.

[101] D. Johansen, H. Johansen, T. Aarflot, J. Hurley, A. Kvalnes, C. Gurrin, S. Zav, B. Olstad, E. Aaberg, T. Endestad, H. Riiser, C. Griwodz, and P. Halvorsen. DAVVI: a prototype for the next generation multimedia entertainment platform. In *Proc. of the International Conference on Multimedia (ACM MM)*, pages 989–990, Oct. 2009.

[102] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Proposed Standard), June 2004.

[103] M. Jurvansuu, J. Prokkola, M. Hanski, and P. H. J. Perälä. HSDPA performance in live networks. In *Proc. of International Conference on Communications (IEEE ICC)*, pages 467–471, June 2007.

[104] M. Karczewicz and R. Kurceren. The SP- and SI-frames design for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):637–644, July 2003.

[105] D. Kaspar, K. Evensen, A. Hansen, P. Engelstad, P. Halvorsen, and C. Griwodz. An analysis of the heterogeneity and IP packet reordering over multiple wireless networks. In *Proc. of the Symposium on Computers and Communications (IEEE ISCC)*, pages 637–642, July 2009.

[106] N. Kennedy. The VideoPress OSMF Player. `http://videopress.com/2010/06/30/osmf-player/`, June 2010.

[107] K.-H. Kim and K. G. Shin. PRISM: Improving the performance of inverse-multiplexed TCP in wireless networks. *IEEE Transactions on Mobile Computing*, 6(12):1297–1312, Dec. 2007.

[108] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). `http://tools.ietf.org/html/rfc4340`, Mar. 2006.

[109] S. D. Kramer. ESPN's live-streamed NBA Finals averaged 330k viewers. `http://paidcontent.org/2012/06/23/espns-live-streamed-nba-finals-averaged-330k-viewers/`, June 2012.

[110] C. Krasic, J. Walpole, and W.-c. Feng. Quality-adaptive media streaming by priority drop. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 112–121, 2003.

[111] O. Layaida and D. Hagimont. Designing self-adaptive multimedia applications through hierarchical reconfiguration. In *Proc. of the International Conference on Distributed Applications and Interoperable Systems (IFIP DAIS)*, volume 3543 of *Lecture Notes in Computer Science*, pages 95–107, 2005.

[112] K. Lee, J. Navarro, T. Chong, U. Lee, and M. Gerla. Trace-based evaluation of rate adaptation schemes in vehicular environments. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 1–5, May 2010.

[113] B. Li and M. I. Sezan. Event detection and summarization in American football broadcast video. 4676:202–213, 2001.

[114] G. Liva, N. Diaz, S. Scalise, B. Matuz, C. Niebla, J.-G. Ryu, M.-S. Shin, and H.-J. Lee. Gap filler architectures for seamless DVB-S2/RCS provision in the railway environment. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 2996–3000, May 2008.

[115] J. Loomis and M. Wasson. VC-1 technical overview. http://www.microsoft.com/windows/windowsmedia/howto/articles/vc1techoverview.aspx, Oct. 2007.

[116] C. Lura. Sumo-stillhet i TV 2. http://www.bt.no/nyheter/okonomi/Sumo-stillhet-i-TV-2-2479385.html, Mar. 2011.

[117] K. Ma, R. Bartoš, S. Bhatia, and R. Nair. Mobile video delivery with HTTP. *IEEE Communications Magazine*, 49:166–175, Apr. 2011.

[118] P. Mähönen, M. Petrova, J. Riihijärvi, and M. Wellens. Cognitive wireless networks: Your network just became a teenager (poster). In *Proc. of IEEE INFOCOM*, 2006.

[119] C.-H. Mai, Y.-C. Huang, and H.-Y. Wei. Cross-layer adaptive H.264/AVC streaming over IEEE 802.11e experimental testbed. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 1–5, May 2010.

[120] Microsoft Corporation. Microsoft media server (MMS) protocol specification. http://msdn.microsoft.com/en-us/library/cc234711%28v=prot.10%29.aspx, Oct. 2012.

[121] Move Networks. Internet television: Challenges and opportunities. Technical report, Move Networks, Inc, Nov. 2008.

[122] C. Müller, S. Lederer, and C. Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proc. of the Workshop on Mobile Video (ACM MoViD)*, pages 37–42, Feb. 2012.

[123] Netflix, Inc. Q1 2012 shareholder letter. http://files.shareholder.com/downloads/NFLX/1925190471x0x562104/9ebb887b-6b9b-4c86-aeff-107c1fb85ca5/Investor%20Letter%20Q1%202012.pdf, Apr. 2012.

[124] P. Ni, A. Eichhorn, C. Griwodz, and P. Halvorsen. Fine-grained scalable streaming from coarse-grained videos. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 103–108, June 2009.

[125] O. Oyman and S. Singh. Quality of experience for HTTP adaptive streaming services. *IEEE Communications Magazine*, 50(4):20–27, Apr. 2012.

[126] J. Ozer. Streaming Learning Center – What's the right keyframe interval? http://www.streaminglearningcenter.com/blogs/whats-the-right-keyframe-interval-.html, Nov. 2010.

[127] R. Pantos and W. May. HTTP live streaming. http://tools.ietf.org/html/draft-pantos-http-live-streaming-10, Oct. 2012.

[128] C. Perkins. IP Mobility Support for IPv4. http://www.ietf.org/rfc/rfc3344.txt, Aug. 2002.

[129] C. Perkins. RTP and the Datagram Congestion Control Protocol (DCCP). http://tools.ietf.org/html/rfc5762, Apr. 2010.

[130] J. Postel. User datagram protocol (UDP). http://tools.ietf.org/html/rfc768, Aug. 1980.

[131] J. Postel. Transmission control protocol (TCP). http://tools.ietf.org/html/rfc793, Sept. 1981.

[132] J. Prokkola, P. Perala, M. Hanski, and E. Piri. 3G/HSPA performance in live networks from the end user perspective. In *Proc. of the International Conference on Communications (IEEE ICC)*, pages 1–6, June 2009.

[133] A. Qureshi, J. Carlisle, and J. Guttag. Tavarua: video streaming with WWAN striping. In *Proc. of the International Conference on Multimedia (ACM MM)*, pages 327–336, 2006.

[134] R. Rejaie and A. Ortega. PALS: peer-to-peer adaptive layered streaming. In *Proc. of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (ACM NOSSDAV)*, pages 153–161, 2003.

[135] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network. In *Proc. of the Workshop on Mobile Video (ACM MoVid)*, pages 25–30, 2012.

[136] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen. Video streaming using a location-based bandwidth-lookup service for bitrate planning. *ACM Trans. Multimedia Comput. Commun. Appl.*, 8(3):24:1–24:19, Aug. 2012.

[137] H. Riiser, P. Halvorsen, C. Griwodz, and B. Hestnes. Performance measurements and evaluation of video streaming in HSDPA networks with 16QAM modulation. In *Proc. of the International Conference on Multimedia and Expo (IEEE ICME)*, pages 489–492, June 2008.

[138] H. Riiser, P. Halvorsen, C. Griwodz, and D. Johansen. Low overhead container format for adaptive streaming. In *Proc. of the Annual Conference on Multimedia Systems (ACM MMSys)*, pages 193–198, Feb. 2010.

[139] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Bitrate and video quality planning for mobile streaming scenarios using a GPS-based bandwidth lookup service. In *Proc. of the International Conference on Multimedia and Expo (IEEE ICME)*, pages 1–6, July 2011.

[140] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. DATASET: HSDPA-bandwidth logs for mobile HTTP streaming scenarios. http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/, Oct. 2012.

[141] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. http://www.ietf.org/rfc/rfc3261.txt, June 2002.

[142] D. Sadlier and N. O'Connor. Event detection in field sports video using audio-visual features and a support vector machine. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(10):1225–1233, 2005.

[143] T. Schierl, Y. Sanchez De La Fuente, R. Globisch, C. Hellge, and T. Wiegand. Priority-based media delivery using SVC with RTP and HTTP streaming. *Multimedia Tools Appl.*, 55(2):227–246, Nov. 2011.

[144] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Real-time transport protocol (RTP). http://tools.ietf.org/html/rfc3550, July 2003.

[145] H. Schulzrinne, A. Rao, and R. Lanphier. Real time streaming protocol (RTSP). http://www.ietf.org/rfc/rfc2326.txt, Apr. 1998.

[146] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the scalable video coding extension of the H.264/AVC standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, Sept. 2007.

[147] E. Setton and B. Girod. Video streaming with SP and SI frames. In *Proc. of the conference on Visual Communications and Image Processing (VCIP)*, 2005.

[148] C. E. Shannon. A mathematical theory of communication. *Bell Syst. Tech. Journal*, 27:379–423 and 623–656, 1948.

[149] R. Stewart. Stream Control Transmission Protocol (SCTP). http://tools.ietf.org/html/rfc4960, Sept. 2007.

[150] T. Stockhammer. Dynamic adaptive streaming over HTTP – standards and design principles. In *Proc. of the Annual Conference on Multimedia Systems (ACM MMSys)*, pages 133–144, Feb. 2011.

[151] J.-Z. Sun, J. Sauvola, and J. Riekki. Application of connectivity information for context interpretation and derivation. In *Proc. of the International Conference on Telecommunications (ConTEL)*, volume 1, pages 303–310, 15-17, 2005.

[152] M. Tamai, T. Sun, K. Yasumoto, N. Shibata, and M. Ito. Energy-aware QoS adaptation for streaming video based on MPEG-7. In *Proc. of the International Conference on Multimedia and Expo (IEEE ICME)*, volume 1, pages 189–192, June 2004.

[153] O. Utsumi, K. Miura, I. Ide, S. Sakai, and H. Tanaka. An object detection method for describing soccer games from video. In *Proc. of the International Conference on Multimedia and Expo (IEEE ICME)*, volume 1, pages 45–48, 2002.

[154] M. Vojkan, M. Petkovic, V. Mihajlovic, W. Jonker, and S. Djordjevic-kajan. Multi-modal extraction of highlights from TV Formula 1 programs. In *Proc. of the International Conference on Multimedia and Expo (IEEE ICME)*, volume 1, pages 817–820, 2002.

[155] K. Wac, A. van Halteren, and D. Konstantas. QoS-predictions service: infrastructural support for proactive QoS- and context-aware mobile services. In *Proc. of the International Workshop on Context-Aware Mobile Systems (CAMS)*, volume 4278 of *Lecture Notes in Computer Science*, pages 1924–1933, Oct. 2006.

[156] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via TCP: An analytic performance study. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(2):16:1–16:22, May 2008.

[157] B. Wang, W. Wei, Z. Guo, and D. Towsley. Multipath live streaming via TCP: Scheme, performance and benefits. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):25:1–25:23, Aug. 2009.

[158] B. Wang, W. Wei, J. Kurose, D. Towsley, K. R. Pattipati, Z. Guo, and Z. Peng. Application-layer multipath data transfer via TCP: Schemes and performance tradeoffs. *Perform. Eval.*, 64(9-12):965–977, Oct. 2007.

[159] R. Weber, M. Guerra, S. Sawhney, L. Golovanevsky, and M. Kang. Measurement and analysis of video streaming performance in live UMTS networks. In *Proc. of the International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–5, Sept. 2006.

[160] T. Wiegand, G. 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, July 2003.

[161] M. Wien, R. Cazoulat, A. Graffunder, A. Hutter, and P. Amon. Real-time system for adaptive video streaming based on SVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1227–1237, Sept. 2007.

[162] F. Xie, K. Hua, W. Wang, and Y. Ho. Performance study of live video streaming over highway vehicular ad hoc networks. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, pages 2121–2125, Oct. 2007.

[163] Q. Xu, A. Gerber, Z. M. Mao, and J. Pang. AccuLoc: practical localization of performance measurements in 3G networks. In *Proc. of the International Conference on Mobile Systems, Applications, and Services (ACM MobiSys)*, pages 183–196, 2011.

[164] J. Yang, N. Tin, and A. K. Khandani. Adaptive modulation and coding in 3G wireless systems. In *Proc. of the Vehicular Technology Conference (IEEE VTC)*, volume 1, pages 544–548, 2002.

[165] J. Yao, S. S. Kanhere, I. Hossain, and M. Hassan. Empirical evaluation of HTTP adaptive streaming under vehicular mobility. In *Proc. of the International Conference on Networking - Volume Part I (IFIP NETWORKING)*, pages 92–105, 2011.

[166] D. Yow, B.-L. Yeo, M. Yeung, and B. Liu. Analysis and presentation of soccer highlights from digital video. In *Proc. of the Asian Conference on Computer Vision*, pages 499–503, 1995.

[167] A. Zambelli. Smooth Streaming technical overview. `http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/`, Mar. 2009.

[168] W. Zhou, A. Vellaikal, and C. C. J. Kuo. Rule-based video classification system for basketball video indexing. In *Proc. of the International Conference on Multimedia (ACM MM)*, pages 213–216, 2000.

[169] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proc. of the International Conference on Quality of Service (IEEE IWQoS)*, pages 137–154, 2003.