Modeling and Optimization of Network Assisted Video Streaming

Printed by Ipskamp Printing, The Netherlands.

ISBN: 978-94-6402-503-3

VRIJE UNIVERSITEIT

Modeling and Optimization of Network Assisted Video Streaming

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor
aan de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. V. Subramaniam,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Bètawetenschappen
op woensdag 28 oktober 2020 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Jan Willem Martin Kleinrouweler

geboren te Nieuwkoop

promotoren:	prof.dr. D.C.A. Bulterman
	prof.dr. R.D. van der Mei
copromotor:	dr. P.S. Cesar

Dankwoord

Dit proefschrift zou niet tot stand zijn gekomen zonder de hulp en het advies van velen. Al diegenen wil ik hierbij bedanken voor alle inspirerende ontmoetingen, de discussies en de feedback die hebben bijgedragen aan de resultaten in dit proefschrift.

Graag zou ik een aantal mensen in het bijzonder bedanken.

Allereerst mijn begeleiders en promotoren Pablo Cesar, Rob van der Mei en Dick Bulterman. Ik wil jullie heel erg bedanken voor de kans die jullie mij geboden hebben met de onderzoeksplek bij het CWI.

Beste Pablo, ik wil jou bedanken voor de vrijheid en het enorme vertrouwen dat je mij gegeven hebt. Ik heb veel van je geleerd; je hebt mij geholpen me te ontwikkelen als onderzoeker. Bedankt voor jouw begeleiding, scherpe blik, het geduld en het vertrouwen bij het positioneren van het onderzoek. Het was voor mij een fijne tijd bij de DIS groep.

Rob, jouw enthousiasme en passie voor het vakgebied zijn aanstekelijk. Ik kijk met een dankbaar gevoel terug op onze overleggen (die gerust een hele middag in beslag namen), de privé-colleges waarin je mij de theorie bijbracht en de mooie resultaten die we bereikt hebben. Dank voor deze ervaringen en ik hoop dat onze paden in de toekomst nog eens zullen kruisen.

Dick, jij hebt mij de ogen geopend en laten zien hoe leuk het is om onderzoek te doen. Bedankt voor jouw uitnodiging om bij het CWI te komen en dat je mij deelgenoot gemaakt hebt van een bonte verzameling van onderzoekers. Je hebt mijn onderzoek (letterlijk) van grote afstand moeten volgen, maar wanneer wij elkaar spraken, was jouw nuchtere- en relativerende kijk op het werk en op het leven altijd zeer verhelderend.

De leden van de promotiecommissie: prof. dr. S. Bhulai, dr. ing. T. Kielmann, prof. dr. S.J. Mullender en dr. ing. G. Hoekstra, mag ik u bedanken voor het plaatsnemen

in de commissie en beoordelen van het manuscript. Prof. dr. K. Nahrstedt, thank you for reading and assessing the manuscript.

Graag wil ik ook Hans van den Berg bedanken voor onze mooie samenwerking aan de laatste paper. Ik vind het fijn dat wij onze samenwerking, nu als collega's bij TNO, voortzetten en ik kijk uit naar de interessante onderzoeken die ons nog te wachten staan.

Sergio Cabrero, I would like to thank you for your support and your help with the architecture designs and the experiments. I enjoyed our discussions and working together. You sparked my enthusiasm when you introduced me to Software Defined Networking, which opened up many new possibilities for our system.

Joost Bosman, ik wil je bedanken voor onze leuke sessies voor het whiteboard. Ik heb veel van je geleerd en het was fijn dat ook wij samen hebben kunnen werken aan een mooie paper.

Gerard Hoekstra, jouw presentaties aan het begin van mijn onderzoekstraject hebben mij geïnspireerd tot het maken van modellen voor videostreaming. Onze gesprekken hebben dan ook bijgedragen aan het vormgeven van het onderzoek. Dank hiervoor.

Bedankt Marwin, Fons, Jack, Simon, Steven, Tom, Ivan, Chen, Kees, Rufael, Mario en de vele andere collega's van de Distributed Interactive Systems groep voor de fijne tijd die ik bij het CWI heb gehad.

Mama, papa, Emily, Margot en Jasper, bedankt voor alles wat jullie voor mij hebben gedaan, jullie onvoorwaardelijke steun en liefde.

Tot slot, lieve Britta, wat ben ik jou enorm dankbaar. Niet alleen voor het mooie onderzoek dat we hebben gedaan, jouw kritische blik op het testbed en de resultaten, het doorlezen en corrigeren van de talloze conceptversies van dit proefschrift, maar ook omdat jij altijd voor mij klaar stond. Ondanks een periode waarin het echt even tegenzat, heb jij mij de ondersteuning, de moed en het vertrouwen gegeven om dit traject succesvol af te ronden. Ik verheug me op een mooie toekomst samen. Ik hou van je!

Contents

1 | Introduction

Computer networks are often organized following a mesh- or tree network topology. Where mesh topologies are typically found in the Internet backbone, tree network topologies are common towards the network edges. For example, tree network topologies are found in home- and corporate networks, large Ethernet based networks in data centers, and as part of the access network of Internet Service Providers (ISPs). Switches and (gateway) routers combine the traffic from multiple devices into one network link. Sharing this network link by multiple devices essentially means sharing the capacity of this network link. Ideally, the division of network resources should reflect the users' needs and wishes. In practice, however, the performance of networked applications is heavily dependent on the application level and network level protocols.

Online video streaming is a networking application where the gap between the ideal resource sharing and the actual accomplished sharing becomes clear. Figure 1.1 shows how four video streaming players share a network link. The lines indicate the video bitrate (i.e., the video quality) over time, showing the inability of video players to evenly share the resources, resulting in continuous over- and under

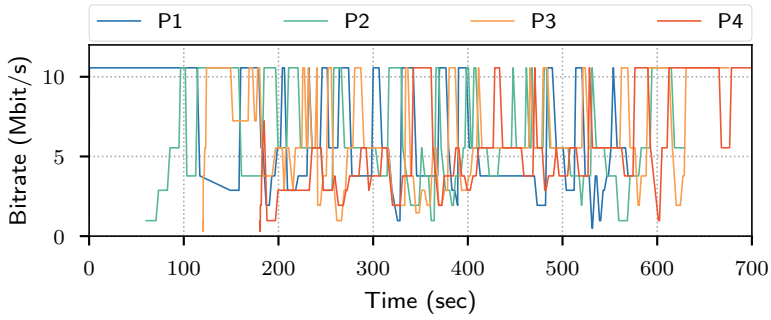


Figure 1.1: Video bitrates for four competing video players in a shared network.

shooting the video bitrate. The lack of control and coordination, combined with the typical selfish implementations of video players trying to always obtain the highest video bitrate, creates an unstable streaming environment and leaves the end user with a poor streaming experience.

The technology driving the video players in the example is HTTP Adaptive Streaming (HAS). HAS has become the dominant streaming technology and is used by large content providers, such as YouTube and Netflix [79]. As such, HAS accounts for a significant part of today's global Internet traffic [117]. Compared to traditional streaming technologies which use dedicated streaming protocols, for example the Real-time Transport Protocol/Real-time Streaming Protocol (RTP/RTSP) suite, HAS uses the HyperText Transfer Protocol (HTTP) for transferring the media content. Using HTTP has the advantage that it traverses firewalls and proxies without manual intervention. It also offers Content Delivery Network (CDN) providers an easier and economical deployment alternative. Horizontal scalability is easier achieved by re-using existing CDN technology and the same infrastructure can be used for live, time-shifted, and Video on Demand (VoD) streaming. Users are not tied to managed networks and dedicated hardware, such as the common use of set top boxes for IPTV, because HAS is supported on a variety of Internet-enabled devices, including smartphones, tablets, computers, and Smart TVs.

Despite its advantages, HAS suffers from performance problems when used on shared network connections, as is demonstrated in Figure 1.1 and further described by Huang et al. [60] and Akhshabi et al. [7, 9]. The decision on video bitrate is made by the HAS players, which rely on bandwidth estimations. Bandwidth estimations are difficult to perform and often unreliable. Additionally, HAS technology is unable to obtain a constant high bandwidth in crowded network environments. As a result, the users' Quality of Experience (QoE) is negatively affected [115, 35, 57]. Users may experience video freezes, low video quality, frequent changes in video quality, and unfair bandwidth sharing. A suboptimal video quality distracts the user, is annoying, and may eventually lead to the abandonment of the stream [43, 124].

This thesis aims at optimizing the streaming performance in shared networks. We propose an optimization platform that is designed to reuse existing HAS protocols, while it reduces the effect of those protocols on the video streaming performance. The proposed optimization in this thesis is a Control Element that works from within the network. The network-based Control Element closely interacts with video players to establish optimal bandwidth sharing, as shown in an overview in Figure 1.2. The information exchange between HAS players and the network starts with HAS players sharing details about the video stream and the playback device with the Control Element. With this information, the Control Element can allocate network resources. The Control Element divides the available bandwidth among

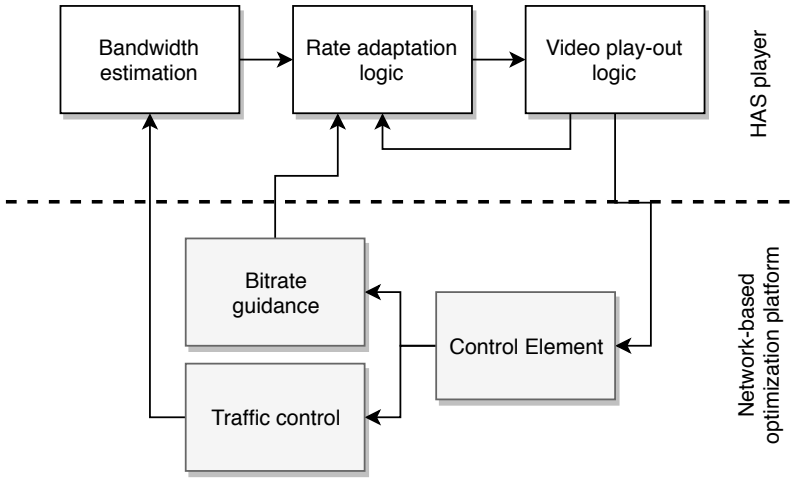


Figure 1.2: Information exchange between HAS players and network elements (adapted from Cofano et al. in [32]).

the video players and informs players about this division through a mechanism called bitrate guidance. In addition to bitrate guidance, traffic control mechanisms are set up accordingly to ensure correct bandwidth sharing. HAS players combine the information from the Control Element with their own bandwidth estimations to decide on the video quality.

The study of optimizing video streaming performance using our Control Element is divided into two parts: (1) control mechanisms and (2) bandwidth sharing policies. The first part of this thesis focuses on the design, implementation, and evaluation of the Control Element. We provide a network setup with suitable mechanisms for Quality of Service (QoS) management for video streaming. QoS management with the Control Element goes beyond prioritizing specific flows, as it, for example, is done in DiffServ architectures [21, 25]. The Control Element in this thesis is aware of HAS and assists HAS players in selecting a video bitrate, basing its decision on the active HAS players, available video representations, and capacity of the network link. As such, the Control Element supports HAS players in optimally using the resources of a bottleneck link.

Having effective mechanisms for exchanging information with HAS video players and establishing traffic control is the first step in network optimization for video streaming. Understanding the impact of bandwidth sharing policies on the overall streaming performance is also required, because tailoring a network to the requirements of the user is done by selecting an adequate sharing policy. Computing a bandwidth sharing policy instance, providing the available network capacity and

information about the active users as input, determines how the network capacity is divided among HAS players. Assessment of the overall streaming performance (e.g., video quality, changes in video quality, and fairness) is then based on changes in the network. Therefore, in the second part of this thesis, we present novel methods in the form of performance models that can accurately determine the expected video streaming performance in a network. These models generate fast results and deep insights on video quality and changes in quality, which will allow the construction of optimal bandwidth sharing policies for different network environments.

This thesis is unique in the sense that it studies video streaming optimization from both an emperical and theoretical angle. As such, we have created a cross-over between two research fields that usually remain separated: multimedia systems and stochastics. The contributions that form the scientific base of this thesis are published in journals and conference proceedings from both fields. This thesis shows that bridging the two fields can be of great value. The cross-over approach has generated valuable insights into tailoring networks for video streaming, showing how combining bandwidth sharing policies with mechanisms for controlling the bitrate of video streams provide the desired overall streaming performance in shared networks.

1.1 Challenges

The principle of HAS is that the control over the stream is part of the video player. Video players adapt the video quality to the network conditions, as shown in Figure 1.3. This process is supported by the preparation of the source material [128]. Video files are encoded in multiple bitrates and split into small segments with a typical duration between one and ten seconds. A HAS player downloads the segments from one or more HTTP servers, changing the video bitrate by selecting segments of a higher or lower video quality. The location and characteristics (e.g., bitrate, resolution, or codec) of the segments are described in a manifest file, which also serves as the entry point for a video stream. The approach of using HTTP, splitting the video into segments, and having the intelligence in the client, enables flexibility in deployment, and it scales well. The downside of the HAS approach is a decrease in streaming performance [60]. Inaccurate bandwidth estimations and high sensitivity to cross-traffic on a shared network connection cause performance problems. HAS players rely on their bandwidth estimations to determine the best fit between available bandwidth and video quality. When selecting a too high bitrate, players cause an overload on the system. Network overloads may result in video freezes for the player in question or other players in the network. Selecting a too low bitrate means that a player could have streamed in higher video quality, missing out on quality for the user.

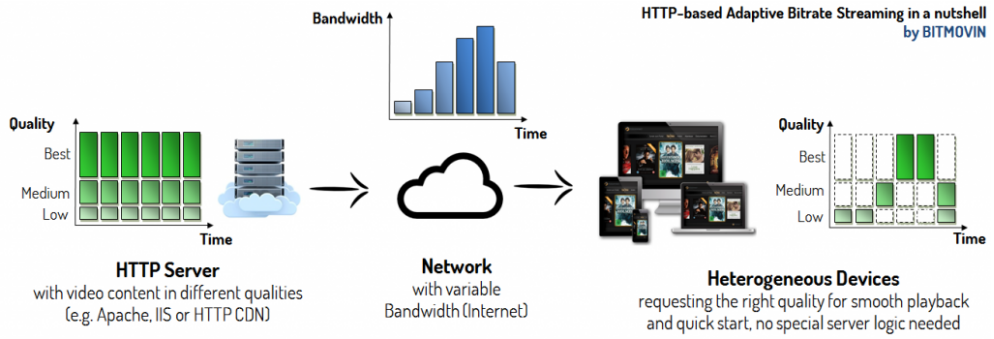


Figure 1.3: HAS players adapt the video bitrate to the available bandwidth [30].

Players continuously have to estimate how much bandwidth is available. During the past years, there has been a large focus on improving adaptation algorithms in HAS players. However, HAS players make bandwidth estimations only from their perspective. This perspective is too limited and does not provide HAS players with a good overview of other concurrent traffic. When video players share a network connection, they are not aware of each other. Instead of coordinating on how to share the available network resources, HAS players selfishly try to use the highest bitrates. Sharing a network thus becomes a competition for bandwidth. When some players increase the video bitrate, it has the inevitable effect that other players have to decrease their bitrate.

In addition to the bandwidth competition among HAS players, there is also a bandwidth competition with other network traffic. HAS downloads video segments in bursts and it pauses between segment downloads. These ON/OFF traffic patterns, as described by Akhshabi et al. [9], prevent high throughputs for HAS streams. The Transmission Control Protocol (TCP), which is used in HAS, requires large window sizes for high throughput. Pausing downloads causes the TCP window to reset. As a result, HAS traffic gets a relatively small portion of network resources.

The discussion about bandwidth competition raises the following question: are best-effort shared computer networks sufficient for creating high-quality streaming experiences, considering the characteristics of HAS? Video players that are not aware of each other and other non time-sensitive traffic taking over the bandwidth from video streams may create a mismatch between network performance and the users' needs. It seems that HAS based video streaming can benefit from better coordination of network resource sharing. This leads to the core hypothesis of this thesis:

A network-based Control Element, that guides HTTP adaptive video streaming clients, improves the video quality and fairness in shared networks.

1.1.1 Integrating a Control Element into the HAS Architecture

In this thesis, we study a Control Element that resides in the network and assists HAS players. Optimizing HAS using the Control Element means that the location of the Control Element and the interaction between the Control Element and HAS players has to be considered. The Control Element needs an overview of the network for which it will manage the resources. We consider bottleneck network links on the last mile as candidates to be manged by the Control Element. As a result, the Control Element manages resources for network links close to the HAS player.

For the interaction between the Control Element and HAS players on the managed network link, it is important to realize that the protocols used by HAS should not change. The reasoning for not changing the HAS protocols is as follows. Video streaming has traditionally been server-based. Specialized streaming servers have been used to send video chunks to clients using dedicated streaming protocols, such as the Real-Time Messaging Protocol (RTMP) [3] or the Real-time Streaming Protocol (RTSP) stack[1] [119]. The User Datagram Protocol (UDP) was often used for efficient and low-latency delivery of media streams. However, server-based streaming faces limitations regarding deployment and scalability. Firewalls and Network Address Translation (NAT) block incoming streams and user intervention (e.g., setting up port forwarding rules) is required. In addition, server-based streaming does not scale, because the specialized streaming servers are stateful and require many computing cycles to re-encode video content for different clients.

Client-based streaming technologies, such as HAS, gained attraction because they overcame the limitations in deployability and scalability. By using the ubiquitous HTTP protocol instead of specialized streaming protocols, complications with firewalls and NATs are resolved. HAS improved on scalability by moving the intelligence to the client and using common HTTP servers in CDNs.

HAS has contributed to the success of online video streaming. Given the importance of deployability and scalability for large content providers, our Control Element should not require changing servers and protocols for HAS delivery. Not changing the servers and protocols means that the Control Element has to integrate seamlessly with existing HAS delivery infrastructures. Obtaining information about network use, dividing network resources, and disseminating information from the Control Element to HAS players has to be done complementary to HAS, not by changing HAS. This leads to our first research question:

Research question 1: *How can a Control Element be designed to seamlessly integrate into existing HAS streaming infrastructures?*

[1] RTSP is used together with the Real-Time Transport Protocol (RTP) [118] for transporting media content and the RTP Control Protocol (RTCP) [64] for monitoring and QoS control.

1.1.2 Delivering a High Quality of Experience

The role of the Control Element is to divide network resources among HAS players. The assumption is that a Control Element that manages network resources will increase streaming performance. A better streaming performance should translate into a better QoE. Therefore, the Control Element should be evaluated in terms of QoE indicators. Even though several holistic QoE models have been proposed in the literature [56, 130, 135], there does not seem to be a consensus. However, the following indicators are frequently used to evaluate HAS streaming performance:

- **Video freezes:** A video freeze is a playback stop caused by a buffer underrun. If the video bitrate is higher than the available bandwidth, the video buffers will eventually run empty. Qi and Dai [107] show that video freezes decrease the QoE. A longer freeze is preferred over several short freezes. Singh et al. compare video quality to freezes, and find that users are more sensitive to video freezes than an increase in video quality [122]. Seufert et al. summarize these findings as "video freezes should be avoided whenever possible, as already little freezes severely degrade the perceived quality" [120].
- **Video quality:** Because of the impact of video freezes on QoE, it is the primary goal of adaptive streaming to avoid video freezes. Nevertheless, the video quality should be a high as possible. Hoßfeld et al. find that the time on the high layer largely affect the QoE [57], a finding that has been confirmed in [130].
- **Quality switches:** Quality switches are the result of HAS players selecting video segments in a different bitrate, either to avoid a video freeze or to increase the video quality. Infrequent quality switches are acceptable if they would allow to spend more time on higher video qualities [57]. However, when the adaptation frequency becomes too high, the QoE is degraded. High amplitude quality switches, especially switches towards lower bitrates, are negatively perceived [130].
- **Fairness:** Fairness refers to sharing networking resources such that users will equally perceive the video quality. When it is preferred, for instance in the case of premium users, certain streams may be of higher bitrate. Unfairness affects the streaming experience negatively for users that get a lower video quality. Users that receive a higher bitrate may be positively impacted. However, one of the cornerstones of the Control Element that is proposed in this thesis is to serve all users in a network, and aim for an overall optimal sharing of network resources.

The Control Element in this thesis has two mechanisms to influence the streaming performance: bitrate guidance and traffic control. Depending on which mechanism is used, the Control Element may be more or less effective in improving the streaming experience. This leads to the following question:

Research question 2: *(How) do the mechanisms in the Control Element improve video streaming performance in shared networks in terms of video freezes, video quality, quality switches, and fairness?*

1.1.3 Tailoring Bandwidth Sharing Policies

The Control Element has an overview of active HAS players in a shared network. It divides the available bandwidth between the active players with the goal to have the optimal bitrate for each player given the current network state. When dividing the bandwidth, the Control Element internally follows a sharing policy. From a network perspective, changing the sharing policy affects the utilization of the network. From a user perspective, the sharing policy affects fairness, video quality, and changes in video quality. In networks with only a few players, simple reasoning is sufficient to determine the expected streaming performance given a sharing policy. In a larger scale network, with tens or hundreds of active HAS players and high dynamics of starting and stopping players, analyzing the expected streaming performance is a challenge.

The empirical approach of building testbeds and evaluating streaming performance based on experiments is a costly and time-consuming process. Simulations or experimental runs require long setup- and run times. Furthermore, the process is error-prone. Because analyzing and adapting sharing policies includes frequent changes to the policies, new and faster methods for evaluating the performance of HAS bandwidth sharing policies are necessary.

This thesis investigates the potential of performance modeling for analyzing sharing policies. We study models that describe the process of starting and stopping players in a shared network, similar to a birth-death process. This process can model how many HAS players are expected to be active at the same time. As such, it provides an excellent base for modeling sharing policies in the Control Element. Nevertheless, the models have to be tailored to express (different) sharing policies. The Control Element should be able to distinguish between players, for instance, based on the type of user, the type of player, or the characteristics of streams. Furthermore, models need to take into account HAS characteristics: limited sets of available bitrates, HAS players stream a video by consecutively downloading video segments, adapting the video bitrate is done by requesting segments from a different representation, and HAS players maintain a video buffer to ensure

continuous playback. Only when these requirements are satisfied, models can make accurate predictions of streaming experience given a policy. This discussion leads to the following research question:

Research question 3: *How can the behavior of the Control Element be stochastically modeled and evaluated?*

1.1.4 Bandwidth Sharing in Mobile Networks

The total capacity of a network connection constrains the available bandwidth that the Control Element divides. In wired networks, the capacity is constant. Up to a large extent, this also holds for Wi-Fi networks, especially when the Wi-Fi network is not the bottleneck network link. In contrast to those "fixed" networks, the network capacity of mobile (cellular) networks is strongly dependent on the quality of the radio signal. When the channel quality of the mobile signal gets lower, the effective throughput decreases, because a mobile base station will use a modulation scheme with more redundancy. The channel quality in mobile networks heavily fluctuates, potentially causing high variations in throughput.

Because of the variations in the effective throughput, assigning a fixed share of network resources – so-called resource blocks – does not guarantee a specific throughput, nor that the throughput remains constant over time. These variations in throughput imply that the Control Element cannot guarantee the desired video quality and reduce switches in the same way as it would do for fixed networks. To be able to divide network resources using the semantic level of the desired video quality, the cost (in resource blocks) for a given video quality first has to be estimated.

Taking a naive approach for allocating resources to match the target bitrate at all times is costly and inefficient. When a mobile client is in a low channel quality zone, the costs of maintaining a constant video bitrate are excessively high. The total resources required for delivering a video stream can be lowered when most of the stream is downloaded in good channel quality zones, and the HAS player relies on the video buffer in low channel quality zones. To ensure video playback while lowering the costs in the network, predictions on channel quality are needed, and the HAS player has to manage its video buffer accordingly. This discussion on the efficiency of video stream delivery leads to the following research question:

Research question 4: *How can modeling be used in the Control Element to achieve a better streaming experience and more efficient delivery of video streams in mobile networks?*

1.2 Methodology

As it becomes apparent in the research challenges, optimizing networks for video streaming requires having the right mechanisms and using them in the best possible way. The study of each of the two parts requires a different approach. Work on the mechanisms requires low-level network information and the specification of protocols between network elements and video players. We use an empirical approach for this part of the optimization problem. Through experimentation with an implementation of our Control Element and by using real video players, we obtained the best performing configuration. This methodology is the norm in the field of multimedia systems. In contrast, work on the policies is more theoretical. Policies are about dividing network resources, assuming that the mechanisms to enforce the decisions are already in place. Therefore, we used methods that are common in the field of stochastics when studying bandwidth sharing policies. To estimate streaming performance given a particular policy, we formulated performance models (i.e., Markov models).

Having two related areas of research using different methodologies does not mean that they stay separated. In this thesis, we deliberately created a cross-over between the work on mechanisms and policies. We assumed that genuinely understanding each of the parts and using knowledge from one field in the other field, would lead to better network architectures and more accurate performance models. Over the course of this thesis, we learned that this assumption is valid. Depending on which aspect of the optimization problem we were working on, different performance assessors were used: from abstract models, over simulations, to experiments with a real implementation. To ensure that findings in one type of evaluation were in line with the others, we frequently compared the results. The solution for optimizing networks for video streaming progressed as a whole. This specific approach allowed us to validate the performance models against a real implementation of our network element, as well as the inclusion of the right key performance parameters in our models.

1.3 Contributions and Thesis Outline

Chapter 2 lays the foundations of this thesis. The first part of Chapter 2 describes the design of the Control Element. This part covers the rationale behind the design, how the Control Element integrates with the existing HAS delivery infrastructure, and the mechanisms for enforcing bandwidth sharing policies. Two manifestations of the Control Element are presented: a proxy server and a Software Defined Networking (SDN) implementation. The second part of Chapter 2 provides an overview of the related work, including client-based, server-based, and network-

based HAS optimization approaches. The contributions in this chapter, which directly address *Research question 1*, include:

- The design of the network-based Control Element that optimizes the delivery of HAS in shared networks. The Control Element is designed to fit into the existing HAS delivery infrastructure.
- The implementation of the Control Element in the form of a proxy server. This implementation works by correcting HAS players when they request too high video bitrates. This implementation can be deployed transparently next to off-the-shelf HAS implementations.
- The implementation of a lightweight and scalable version of the Control Element based on SDN technologies. This implementation requires minor changes to the HAS players, but increases performance and privacy.

This chapter is based on the following publications:

> **Jan Willem Kleinrouweler**, Sergio Cabrero, and Pablo Cesar. 2017. *An SDN Architecture for Privacy-Friendly Network-Assisted DASH*. In ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM). ACM, Volume 14, Edition 3s, Article 44, 22 pages.
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, and Pablo Cesar. 2016. *Delivering stable high-quality video: an SDN architecture with DASH assisting network elements*. In Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16). ACM, New York, NY, USA, Article 4, 10 pages. **[Excellence in DASH Best Paper Award, sponsored by DASH-IF]**
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2015. *Modeling stability and bitrate of network-assisted HTTP adaptive streaming players*. In Proceedings of the 27th International Teletraffic Congress (ITC 27). ITC, Ghent, Belgium, pages 177-184.

Chapter 3 presents the evaluation of the Control Element. The chapter starts with an evaluation of the proxy server implementation in a wired network. The evaluations continue with a performance assessment of the SDN-based implementation in both Wi-Fi and wired networks. The chapter focusses on the impact of the different mechanisms for optimizing HAS in shared networks. The contributions in Chapter 3 that address *Research question 2*, include:

- The performance evaluation of the proxy server implementation of the Control Element. The focus in this evaluation is on reducing quality switches and

improving fairness among HAS players. The evaluation is performed in an emulated wired network with up to 25 HAS players.

- The performance evaluation of the SDN-based implementation of the Control Element in a Wi-Fi network. The focus in this evaluation is on the effect of the mechanisms of bitrate signaling and traffic control on the streaming experience.
- A demonstration of versatility and scalability of the SDN-based implementation of the Control Element. Scalability is assessed in a wired network with up to 600 HAS players.

This chapter is based on the following publications:

> **Jan Willem Kleinrouweler**, Britta Meixner, and Pablo Cesar. 2017. *Improving Video Quality in Crowded Networks Using a DANE*. In Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2017). ACM, New York, NY, USA, pages 73-68. **[2nd place Excellence in DASH Best Paper Award, sponsored by DASH-IF]**
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, and Pablo Cesar. 2017. *An SDN Architecture for Privacy-Friendly Network-Assisted DASH*. In ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM). ACM, Volume 14, Edition 3s, Article 44, 22 pages.
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, and Pablo Cesar. 2016. *Delivering stable high-quality video: an SDN architecture with DASH assisting network elements*. In Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16). ACM, New York, NY, USA, Article 4, 10 pages. **[Excellence in DASH Best Paper Award, sponsored by DASH-IF]**
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2015. *Modeling stability and bitrate of network-assisted HTTP adaptive streaming players*. In Proceedings of the 27th International Teletraffic Congress (ITC 27). ITC, Ghent, Belgium, pages 177-184.

Chapter 4 moves the attention from the mechanisms in our Control Element to the bandwidth sharing policies. This chapter answers *Research question 3* through the formulation of Markov models for estimating the streaming performance of the Control Element. The models take a potential bandwidth sharing policy and streaming load as input and estimate the video quality and video quality changes. The performance models explain several aspects of video streaming with our Control Element, such as the effect of HAS encoding schemes, video segments, playout buffer, and sharing policies. The contributions in this chapter include:

- A Markov model that predicts the expected streaming performance of different sharing policies in the Control Element. The model describes the performance in streaming experience indicators: video quality, quality switches, and fairness.
- The specification of sharing policies that include device characteristics and premium users. Using the model, the specialized sharing policies are compared to the bitrate fair policy.
- An extensive validation covering the accuracy and sensitivity of the performance model against experiments with the proxy server implementation.

This chapter is based on the following publications:

> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2016. *A Markov model for evaluating resource sharing policies for DASH assisting network elements*. In Proceedings of the 28th International Teletraffic Congress (ITC 28). ITC, Würzburg, Germany, pages 112-120.
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2016. *A model for evaluating sharing policies for network-assisted HTTP adaptive Streaming*. In Computer Networks. Elsevier, Volume 19, pages 234-245.
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2015. *Modeling the Effect of Sharing Policies for Network-assisted HTTP Adaptive Video Streaming*. In SIGMETRICS Performance Evaluation Review. ACM, Volume 43, Number 2, pages 26-27.
>
> **Jan Willem Kleinrouweler**, Sergio Cabrero, Rob van der Mei, and Pablo Cesar. 2015. *Modeling stability and bitrate of network-assisted HTTP adaptive streaming players*. In Proceedings of the 27th International Teletraffic Congress (ITC 27). ITC, Ghent, Belgium, pages 177-184.

Chapter 5 covers the use of the Control Element in mobile networks. This chapter investigates the use of buffer level information from a HAS player when scheduling resources in a mobile network, in order to improve effeciency of delivering video content. In mobile networks, the channel quality from the base station to the client determines the transmission efficiency. Chapter 5 describes a strategy that grows the video buffer when network transmission is efficient and shrinks the video buffer when network efficiency is low. Simulations show that this buffering strategy reduces the load on the mobile network, while keeping the video quality constant. The contributions in Chapter 5 that answer *Research question 4*, include:

- The collection of an extensive data set of LTE channel quality traces.
- A Markov model for predicting future channel quality in LTE networks.
- A method for more efficient delivery of HAS streams in mobile networks. Through a buffering strategy that is based on coordination between HAS clients and the Control Element, the load on the mobile network can be lowered while keeping the quality of the video almost constant. Optimal parameters for the buffering strategy are obtained by solving a Markov Decision Process (MDP).

This chapter is based on the following publications:

> **Jan Willem Kleinrouweler**, Britta Meixner, Joost Bosman, Hans van den Berg, Rob van der Mei, and Pablo Cesar. 2018. *Improving Mobile Video Quality Through Predictive Channel Quality Based Buffering*. In Proceedings of the 30th International Teletraffic Congress (ITC 30). ITC, Vienna, Austria, pages 236-244.
>
> Britta Meixner, **Jan Willem Kleinrouweler**, Pablo Cesar. 2018. *4G/LTE Channel Quality Reference Signal Trace Data Set*. In Proceedings of the 9th ACM Multimedia Systems Conference (MMSys'18). ACM, New York, pages 387-392.

Chapter 6 revisits the research questions and concludes this thesis. The chapter reflects on the novel methodology that looks into network optimization from both an empirical and a theoretical perspective. The influence of factors from both research areas and how they contributed to a better understanding of video streaming aware policies is discussed. Chapter 6 closes the thesis with an outlook on future work in the area of network optimization for media streaming.

2 | Architecture and Related Work

Bandwidth sharing in a network with multiple video players is a challenge. The bandwidth has to be divided fairly between the players, and the bandwidth used for streaming should be in balance with other traffic in the network. For video streaming, the common approach is that the video player is in control of the stream and algorithms in the video player select the bitrate (i.e., how much of the network bandwidth to use) based on estimations of the available bandwidth. Hence, each player in a shared network selects the bitrate from a local perspective. This approach has severe limitations. Therefore, in this thesis, we challenge this approach by using a node that resides in the network and supports HAS bandwidth sharing from within the network. We call this node the Control Element. This chapter starts with the description of the architectural design of the Control Element, including obtaining information about players, dividing the available bandwidth, and providing feedback to HAS players. Then, we propose two implementations: a proxy server and an SDN-driven implementation. Both implementations coordinate resource usage. The proxy server approaches this problem from the perspective of correcting video players that do not fairly share the available network bandwidth. We compare this approach to the SDN-driven implementation that informs video players about bandwidth sharing from the start. The end this chapter provides an overview of related work, including client-based, server-based, and network-based optimization techniques.

This chapter is based on publications [K4][K7][K9][K11].

2.1 Introduction

HTTP Adaptive Streaming (standardized as Dynamic Adaptive Streaming over HTTP (DASH) in [66]) is the dominant technology for online video streaming. In HAS, a video stream is encoded in multiple bitrates and resolutions, so-called representations. After encoding, each representation is split into small video segments with a typical duration between one and ten seconds. A manifest file describes the properties of the stream (i.e., the list of available representations) and lists the location of the video segments. All video segments plus the manifest are uploaded to an HTTP server or CDN. The HAS player first downloads the manifest, then it downloads the video segments one by one. Before downloading a video segment, the HAS player selects a representation from the manifest, basing its decision on the current network conditions and the player's state. Bitrate adaptation allows a player to increase the video quality when the available bandwidth in a network increases. When the available bandwidth decreases, switching to a lower video quality representation avoids the player from playback interruptions. The use of video segments and ability to adapt the video bitrate, make HAS a more flexible and robust technology compared to downloading a single video file, for example using progressive download.

HAS has contributed significantly to the success of online video streaming for two reasons: HTTP solved deployment problems caused by firewalls and NATs. Scalability improved by using a client-based streaming approach. Streaming with HAS does not require specialized servers, instead regular HTTP servers and CDNs may be used. Based on internal adaptation algorithms, HAS players select a video bitrate that matches the available bandwidth. The distributed and independent approach of HAS sounds beneficial for both content providers and end-users. However, the approach has two major shortcomings: it is challenging for an individual HAS player to make accurate bandwidth estimations [60] and HAS is submissive to TCP [45, 138].

HAS players make predictions of the future bandwidth when selecting a video bitrate. Common estimation techniques include measuring the download speed of past video segment downloads or observing occupancy of the internal video buffer. Both techniques rely on measurements from the past to determine future bandwidth. Besides that past measurements may not be a good indicator, measurements may be inaccurate because a player can only measure available bandwidth when it is active in the network (i.e., a player has to download a video segment while measuring). HAS players download video segments one by one, creating ON-OFF download patterns as described by Akhshabi et at. in [7]. Not only do ON-OFF patterns create a limited window for throughput measurements, they also provide a false impression of network activity to other HAS players on a shared network connection. HAS

players can only acquire awareness of each other when synchronously downloading video segments. Competing HAS players result in video quality instability and unfair sharing of network resources [9, 7].

The second performance shortcoming of HAS is caused by the effect of TCP on HAS. HAS is submissive to TCP [138]. This means that adaptation algorithms cannot select higher video bitrates than the throughput supplied by TCP. However, the short and bursty nature of downloading video segments over TCP prevents high throughput in networks with cross-traffic [138]. Given low download speeds, HAS players cannot select high-quality video segments and increase the video buffer. This will lead to an overall low video quality and potential video freezes, when TCP throughputs are too low to continue a stream.

This discussion on the design of HAS and its limitations shows a split in objectives. On the one hand, content providers need easy deployment and scalability of their service. Hence, they opt for a simplification of streaming technologies, as shown by the current landscape of video streaming [79]. Users, on the other hand, want a high quality video streaming experience: no freezes, high image quality, stable video quality, and a fair video quality [120]. The limitations of HAS make this difficult to realize in shared networks. This thesis attempts to optimize both objectives: enabling HTTP-based video streaming in shared networks while providing its users a high-quality video streaming experience. In this chapter, we present the design of our network-based optimization platform that challenges the distributed approach of individual adaptation algorithms. By using a Control Element in the network, which has centralized intelligence and awareness of HAS players, each player can be assigned the appropriate network resources. This approach should increase video streaming performance, while easily integrate with existing HAS architectures. As such, the contributions in this chapter answer the first research question in this thesis:

Research question 1: *How can a Control Element be designed to seamlessly integrate into existing HAS streaming infrastructures?*

2.2 Requirements and General Architecture

The core idea behind our optimization platform is to have a centralized node, the Control Element, that manages the resources on a shared network connection. The Control Element obtains information about active HAS players and other traffic to create an overview of the HAS video streams. Given this overview, the Control Element can make an informed (and potentially optimal) decision on how to divide the available bandwidth at each given time. Optimizing bandwidth sharing for HAS

ensures streaming continuity, the highest possible video quality, and high stability. In addition, the bandwidth sharing policy implements user defined fairness. As such, the Control Element addresses these first two requirements:

- *R1: The Control Element should increase the video streaming experience in shared networks.*
  The Control Element is there for the users of a shared network connection. Its main purpose is to eliminate the negative side effects of HAS. The Control Element should supply HAS players with sufficient resources to eliminate video freezes and increase video quality. The Control Element should eliminate the competition for bandwidth between streams and stabilize the video quality.
- *R2: The Control Element should be able to make any division of bandwidth.*
  Sharing network bandwidth is often viewed as equally dividing the resources. In the case of video streaming this means providing each HAS player with the same throughput. However, given the diversity of devices, streams, and users, some streams need more bandwidth than others to yield a fair experience. In these cases, fairness follows a sharing policy that reflects the needs and requirements of the users of a network. The Control Element should be able to execute each policy, allowing uneven division of network bandwidth. The Control Element should prevent HAS players that get lower bitrates assigned from overtaking HAS players that require more bandwidth, even though internal bandwidth estimations indicate possibilities for higher bitrates.

The first two requirements target the improvement in video streaming experience for the end-users. The next two requirements satisfy concerns for content providers. Because the Control Element adds streaming intelligence outside the HAS players, it might seem to break with the principles of HAS: use HTTP and locate all intelligence at the client. These design decisions were taken to ease deployment and solve scalability problems for content providers, and they successfully did so. Given the importance of easy deployment and scalability for content providers, the Control Element should not require a change to the CDN servers or the HAS file formats. The Control Element has to integrate seamlessly into the current delivery infrastructure. The Control Element has to inform itself about active HAS players, divide network resources, and disseminate its decisions in addition to HAS, not by changing HAS. This discussion is reflected in the following two requirements:

- *R3: The Control Element should integrate with the current HAS formats and HTTP servers.*
  Content providers should be able to use current HAS manifest formats and rely on HTTP for transporting manifests and video segments. The addition of the

Control Element to a network should not affect this part of the delivery chain. Content providers should be able to optimize their CDNs independently of the optimization of bandwidth sharing in user networks with the Control Element.

- *R4: HAS players should be able to operate with and without the Control Element.* HAS players can only use the services of the Control Element in the networks where it is active. However, the Control Element will not be deployed in every network. Therefore, video streaming should operate following normal HAS functionality in networks without the Control Element.

Initially, we had a Control Element in mind that would be deployed in a network close to- and owned by its users (e.g., at the gateway of a local area network or a Wi-Fi router). These networks are typically small scale and will support a limited number of HAS players. However, the Control Element does not have to be limited to local networks only. For example, it can also serve at Internet Service Provider (ISP) level and provide support for a HAS-based television service. When the Control Element is used by a (large) third party, it is important that it can handle large amounts of streams. Additionally, the Control Element should not trade privacy for performance. Therefore, we formulated the following three additional requirements when using the Control Element in larger setups:

- *R5: The Control Element should not be invasive to the users' privacy.* The Control Element should not have to inspect network traffic of its users. It should not be able to see what users are watching, in order to protect their privacy.
- *R6: The Control Element should be lightweight and scalable.* The Control Element should be able to handle hundreds of HAS players in a network. This allows for scaling the Control Element to larger networks, found in apartment complexes, large hotels, or universities. Processing and overhead in the Control Element should be minimized, such that it can be deployed on low-powered computing hardware.
- *R7: The Control Element should be able to respond to changes in the client.* In small private networks, the devices used for video streaming might all be known. Users can configure a sharing policy based on the devices they own. In larger networks, the Control Element does not know the users and devices from the start. Therefore, the Control Element should have mechanisms for devices to make themselves, and their characteristics known, to it. Furthermore, properties can change. For instance, when a user changes a video window to full screen. The Control Element should be able to respond to those changes.

The Control Element acts as a central node for managing HAS players. Figure 2.1 shows an overview of how the components in the Control Element interact. Arrows indicate feedback between the components. At the bottom of this figure are the elements that are located in the network. As such, the platform provides an extra layer on top of a network that is dedicated to improving the performance of video streaming applications. The Control Element is the central node in the design. Its task is to obtain information from active HAS players. To divide network resources among HAS players, the Control Element at least needs to know the number of active HAS players and the characteristics of the streams (i.e., the available bitrates and resolutions of video tracks and the bitrate of audio tracks in the manifest). Additional information, such as the type of playback device or user, can be used as well when available. Based on the information gathered by the Control Element, it divides the available bandwidth of a network connection among the HAS players.

Each player is assigned a target bitrate. The target bitrate is the highest available bitrate in a HAS manifest, such that the target bitrate does not exceed the share of resources assigned to a player. To make a HAS player follow the target bitrate, the Control Element provides feedback to the HAS players. We consider two feedback mechanisms: bitrate guidance and traffic control. With bitrate guidance, the Control Element makes sure that HAS players do not select bitrates that are higher than the target bitrate. The role of traffic control is two-fold. First, it ensures that HAS players can stream at the target bitrate by providing guaranteed network resources that match the target bitrate. Second, traffic control reduces the performance artifacts introduced by using HTTP as the underlying protocol.

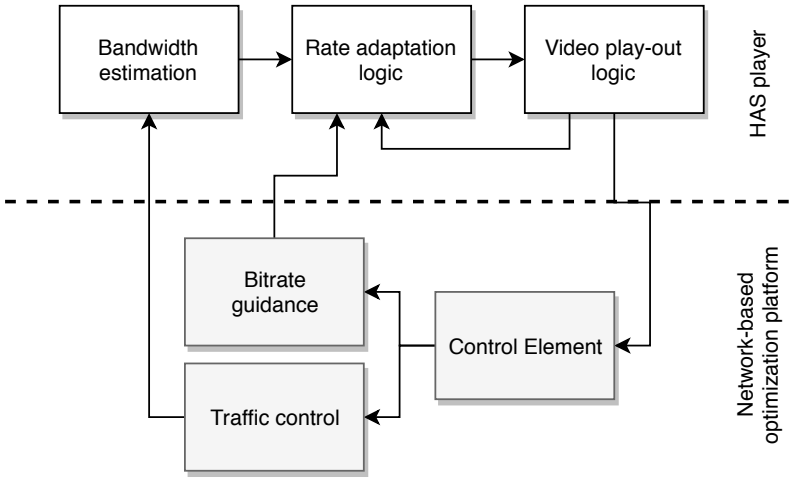


Figure 2.1: Core design of our Control Element for optimizing HAS in shared networks (adapted from Cofano et al. in [32]).

The top of Figure 2.1 shows the elements implemented in the HAS players which are compatible with the Control Element. Although the elements at the top of this figure are similar to a conventional HAS player, the vertical arrows indicate where bitrate adaptation is affected by our optimization. Because of traffic control, the download speeds of video segments are affected. In crowded networks, traffic control can provide an increase in download speed, indicating to the HAS player that sufficient bandwidth is made available. Guaranteeing throughput increases stream continuity. Besides guaranteeing a minimum throughput, traffic control can also cap the throughput. Capping the throughput removes the false impression in HAS players that more bandwidth is available. This process creates download speeds with fewer fluctuations, resulting in more accurate bandwidth estimations and stable streaming quality.

Nevertheless, traffic control is indirect feedback to the HAS player, because HAS players have to measure its effect through bandwidth estimations. In contrast to traffic control, bitrate guidance provides direct feedback. Bitrate guidance means telling a HAS player the target bitrate. The primary purpose of bitrate guidance is to provide HAS players with an upper limit on the video bitrate. Staying under this limit prevents HAS players from putting a too large demand on the network, increasing the continuity and fairness of video streams in the network. When HAS players rely stronger on the target bitrate than on bandwidth estimations, stability is further improved. HAS players can safely adopt the target bitrate when buffer levels are sufficient. Only when buffer levels are low, HAS players should prefer bandwidth estimations over the target bitrate.

The Control Element described in this chapter will be used throughout the thesis. In the next sections, we will discuss two implementations of the Control Element. The first implementation is a proxy server that monitors the traffic on a shared network connection. The proxy server listens for HAS requests and modifies the requests when needed. One can transparently deploy the proxy server in a network, integrating the Control Element without changing current HAS delivery. The second implementation addresses additional concerns like privacy and scalability of the Control Element. This implementation uses SDN technology, and assists HAS players via a separate communication channel between HAS players and the Control Element. With this implementation, we examine the possibilities of having control over the player, while remaining compatible with the existing HAS back end (i.e., servers and protocols).

2.3 Proxy Server Implementation

The first implementation of the Control Element focusses mostly on bitrate guidance. This Control Element implements bitrate guidance by overruling HAS players when they select too high bitrates. Through monitoring each HAS stream and lowering the video bitrates for segment requests with a too high bitrate, the Control Element prevents a network overload. We implement this approach in a proxy server.

The proxy server transparently operates on the gateway of a shared network. As such, all HTTP traffic (i.e., packets with TCP destination port 80) run through the proxy server. The proxy server does not alter regular traffic. It only inspects HAS. By scanning HTTP requests headers for manifests and video segment requests, the proxy server learns about HAS players. Manifests are detected based on the file extension. In our implementation, we scan for HTTP Live Streaming (HLS) manifests with the `.m3u8` file extension. For Dynamic Adaptive Streaming over HTTP (DASH), the `.mdp` extension should be used. When a request for a manifest is detected, the proxy server scans the response from the server. The response contains the manifest which is stored by the proxy server. The proxy server scans the manifest for video segment URLs. For each segment URL, the proxy server configures a trigger that activates when a HAS player requests one of the segments. After processing the manifest, the proxy server divides the available bandwidth in the network among the HAS players. For each player, the proxy server selects a target bitrate. The target bitrate is the highest available bitrate in the HAS manifest that is lower than the assigned share of network resources.

When one of the HAS players requests a video segment, the proxy server checks the bitrate of that segment. When the bitrate is higher than the target bitrate from the proxy server, it has to be lowered to prevent that this specific HAS player takes too many network resources. This action will prevent potential performance problems in other HAS players in the network. The proxy server lowers the bitrate of a video segment by rewriting the HTTP request, before forwarding the request to the server. Rewriting means replacing the URL in the request with a URL for the same video segment in another (lower) bitrate. Because the proxy server stores the HAS manifest, it is capable of making this mapping. Figure 2.2 shows an example of how the proxy server rewrites the segment requests of two HAS players that share a network connection.

The figure shows how the proxy server corrects the HAS player that requests a too high bitrate. However, the HAS player requesting a segment in a bitrate below the target bitrate is also corrected. In some cases, the proxy server also rewrites lower bitrates. If this happens, depends on the occupancy of the playback buffer in the

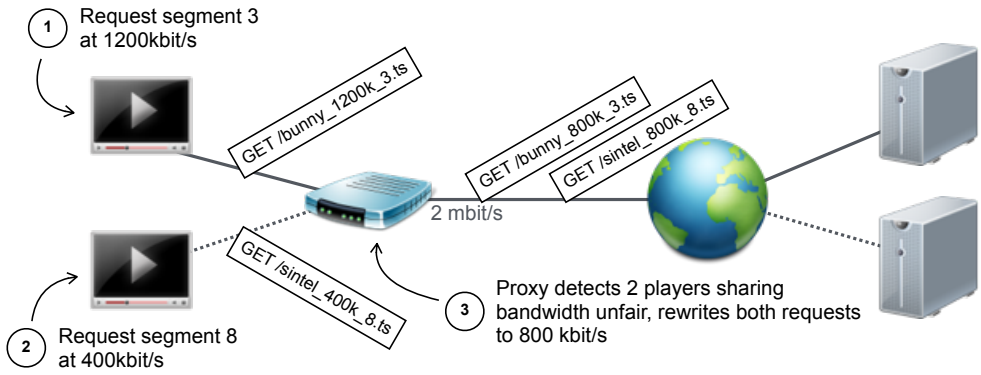


Figure 2.2: Proxy server corrects two HAS players that do not fairly share the network bandwidth.

player. In our implementation, the minimal buffer occupancy must be at least seven seconds before the proxy server rewrites segment URLs to a higher video bitrate.

Because the proxy server does not know the buffer occupancy in a player, it will make an estimation. Buffer occupancy can be estimated based on the time between consecutive segment requests, using a similar technique as in [9]. HAS players request video segments sequentially and predictably. The increase (or decrease) in buffer level can be estimated by comparing how much video is downloaded and how much video is played out, in the same period. The time between consecutive segment requests is approximately the segment duration. After a request, 'segment duration' seconds are added to the buffer. If two requests happen at τ_1 and τ_2, then no more than $\tau_2 - \tau_1$ seconds are played out. Equation 2.1 specifies the increase (or decrease) in buffer level. $T_{segment}$ is the segment duration.

$$\text{buffer}_{change} = T_{segment} - (\tau_2 - \tau_1), \tag{2.1}$$

When a user navigates within a video, the HAS player requests a non-consecutive segment. In this case, the proxy server assumes that the buffer is empty because many players do not store segments that are already played out or discard the buffer contents when a user jumps to a different position in the video.

The proxy server detects a starting player when it downloads the manifest. A player finishes after downloading the last segment in the manifest. However, since users frequently stop a stream before entirely watching it, a stream is also considered stopped after a certain period of inactivity. To set a value for this timeout, we make use of the periodic behavior of HAS players. In steady-state mode, HAS players request segments with intervals equal to the duration of a segment. For example,

a HAS player requests a segment every four seconds given a segment duration of four seconds. If the download of a segment takes shorter than the duration of that segment, there will be a period of inactivity.

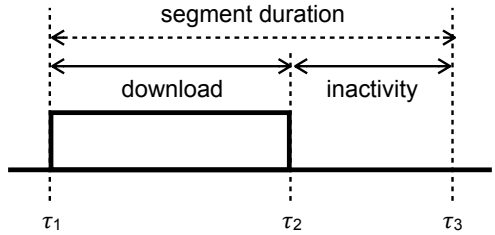


Figure 2.3: Period of download activity followed by a period of inactivity.

As depicted in Figure 2.3, τ_1 marks the start of a download of a segment, and τ_2 marks the stop of the download. Equation 2.2 specifies the maximum period that a player can be inactive before requesting the next segment, given a segment duration of $T_{segment}$ seconds.

$$I_{max} = \max(T_{segment} - (\tau_2 - \tau_1), 0) + 1 \tag{2.2}$$

The inactivity period is unlikely to be higher than the duration of the segment minus the download time. If the download of a segment takes longer than the segment duration, there will be no period of inactivity. To be sure that a player has finished, and to cope with small variations in periodicity, Equation 2.2 includes a margin of one second.

The optimizing proxy server heavily relies on monitoring HTTP traffic. All HTTP requests have to be scanned for manifest- or segments requests. When the proxy server detects a HAS manifest or video segment, it has to inspect it for modification. Being able to the scan HTTP traffic implies that the owner of the proxy server can see which videos have been watched. Therefore, large content providers are already moving their infrastructures to HTTPS [94]. Using HTTPS could bypass the proxy server (i.e., manual configuration of a non-transparent HTTPS proxy would be required) and users would lose the performance benefits provided by the proxy server. Therefore, we created a second implementation which separates control from the data plane, allowing the delivery of video streams to be encrypted, while maintaining a separate control channel with the Control Element. The next section in this chapter describes this implementation.

2.4 SDN-based Implementation

The second implementation is designed to be deployed in a SDN environment. In SDN [98], network control is separated from the data forwarding plane. Figure 2.4 shows how we follow this approach. The SDN-based Control Element includes four components: the Service Manager, Network Controller, programmable network hardware, and assistance-enabled HAS players. The Service Manager tracks the active HAS players. HAS players use a WebSocket [47] to provide the Service Manager with information. The Service Manager uses this WebSocket to signal the HAS players their target bitrates. After dividing the available bandwidth, the Service Manager instructs the Network Controller to configure routing and traffic control in the network. The Network controller uses SDN protocols (e.g., OpenFlow [89]) to configure the programmable network hardware.

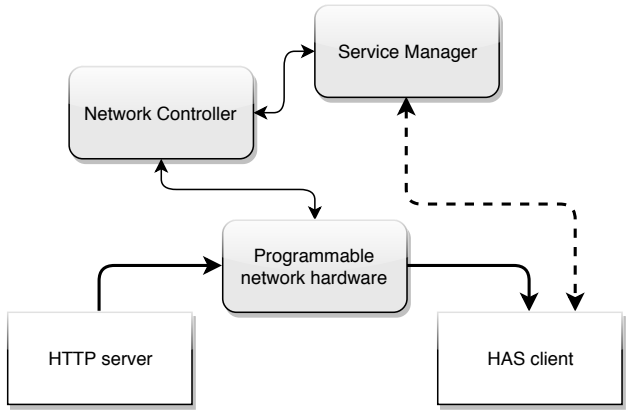


Figure 2.4: Design of our SDN-based implementation. The dashed line between the HAS player and the Service Manager marks the communication channel for sending adaptation assistance messages.

The Service Manager, Network Controller, and network hardware are separate components. Depending on the size of a network, physically different network devices can hosts the different components. In small network deployments (e.g., a home Wi-Fi network) all components fit on the Wi-Fi router. Communication between HAS player and Service Manager is lightweight, and given the limited number of potential devices, the amount of processing is small. For more extensive networks, the Service Manager and Network Controller run on separated servers. Even cloud-offloading is possible in ISP scale networks. The only two requirements are that the HAS players can establish a connection with the Service Manager and that the network hardware at the bottleneck link is SDN-enabled.

2.4.1 Programmable Network Hardware and Network Controller

Programmable network hardware (e.g., switches, routers, Wi-Fi access points) build the basis of this Control Element. In SDN, programmable network elements have two primary functions: they collect device status and network statistics, and they execute sophisticated forwarding rules. The Network Controller is the central entity that monitors and configures the SDN-enabled network elements. The Network Controller has a global view of the shared network connection and can dynamically add, adjust and remove forwarding rules in the network elements.

By setting up forwarding rules, the Network Controller provides transparent routing from the HAS players to the Service Manager. The HAS player may initially not know where to contact the Service Manager because it can be located at different locations in the network, or even outside the boundaries of the local network in a cloud. HAS players attempt to connect to the Service Manager by opening a WebSocket connection using a predefined IP-address. Traffic to this address routes to the actual Service Manager. For setting up the forwarding rules, we use the standard protocol OpenFlow [89] for the interface between the Network Controller and the network hardware.

Besides configuring forwarding rules, the Network Controller is also responsible for setting up traffic control. Although OpenFlow has the `enqueue` message to put packets into specific traffic shaping queues, at the time we presented this implementation there were no OpenFlow messages available for creating, modifying, and deleting queues. Therefore, we implemented an additional interface on our switches that allows for dynamic queue configurations. By default, the Network Controller configures one queue for normal (non-HAS) traffic. We will refer to this traffic as background traffic. Background traffic can use the full potential of the network connection when there are no HAS players active. When HAS players start streaming, the Network Controller creates additional queues for video streaming traffic. The size of the background traffic queue is also automatically adjusted.

We implement two different modes for traffic control queues: bandwidth shaping and prioritization. A bandwidth shaping queue restrains the throughput for packets in that queue. Each bandwidth shaping queue is separated from other queues. Unused network bandwidth within one queue cannot be used to transfer packets in other queues. In contrast, a prioritization queue guarantees a minimum throughput for packets in specific queues. This type of queue does not impose a maximum on the throughput, and remaining bandwidth in queues (i.e., the bandwidth that prioritized traffic does not use) can be shared among queues.

2.4.2 Service Manager

HAS players open a WebSocket connection to the Service Manager to receive adaptation assistance. When HAS players connect to the Service Manager, the Service Manager becomes aware of them. The HAS players send the characteristics of a stream (i.e., the list of available bitrates for video and audio tracks) and potentially share additional information about the device (e.g., the screen size or resolution) with the Service Manager. The Service Manager combines the information for all active HAS players. Based on the network capabilities provided by the Network Controller, the Service Manager divides the network bandwidth among the players. Depending on the sharing policy in the Service Manager, network bandwidth is equally shared among players, or the Service Manager takes other factors (e.g., device type) into account.

The decisions on bandwidth sharing from the Service Manager propagate in two ways: via target bitrate signaling using the control channel between HAS player and Service Manager, and through dynamic traffic control using either bandwidth shaping or prioritization queues. The HAS players use bitrate signaling as guidance in their (modified) adaptation algorithms. Traffic control queues ensure that HAS players have enough bandwidth available to stream at the target bitrate.

The Service Manager in our optimization platform can set up queues in two different ways: a single queue for all HAS streams or a dedicated queue per HAS player. The first method shares aspects with the DiffServ network architecture [21]. This method creates a single queue for HAS streams, while other traffic goes into the background traffic queue. The size of the queue depends on the number of active HAS players and the target bitrates of each player (i.e., the size of the queue is the sum of all target bitrates). The Service Manager instructs the Network Controller to resize the queue when a new HAS player starts or an active player stops.

The second traffic control configuration creates one queue per active HAS player. In this mode, the Service Manager separates HAS players from background traffic, and from each other. Using this configuration, the Service Manager can police network resource sharing more strictly when different HAS players get different video bitrates assigned. A new traffic shaping queue is created when a HAS player starts, and the queue is destroyed when streaming ends.

For both traffic control configurations, we add a safety margin to the size of a queue. Through experimentation, we found that the size of a queue should be at least 1.2 times the (aggregated) target bitrate(s), ensuring smooth playback and healthy buffer levels. The safety margin provides extra robustness again variations in video bitrate (e.g., due to VBR encoding).

2.4.3 Assistance-enabled HAS Player

Existing HAS players need to be modified to support adaptation assistance from the Service Manager. The HAS player connects to the Service Manager via a WebSocket. WebSockets offer a two-way communication channel and are a widely implemented web technology. Leveraging WebSockets makes our platform compatible with HAS players that work in the browser and WebSockets do not create issues with firewalls and NATs.

HAS players open the WebSocket connection to the Service Manager during their initialization phase. Opening the connection indicates the Service Manager that video streaming is imminent. After receiving the manifest from the server, the HAS player reports the bitrates from the manifest (for both audio and video) to the Service Manager. The player may leave out some (higher) bitrates in this report. For example, it can ignore the bitrates that exceed the capabilities of the device. If the player wants to change the set of bitrates during playback (e.g., the user resizes the player to full screen), it sends the new collection of bitrates to the Service Manager. The Service Manager redivides the bandwidth when it receives such an update. Players that pause or end a stream indicate this to the Service Manager using separate messages, or by closing the WebSocket.

The Service Manager applies the sharing policy after it receives a set of bitrates from a HAS player. The result is a target bitrate for each player. The target bitrate denotes the representation for which the Service Manager thinks it is the best bitrate given current network activities. The Service manager sends each HAS player its target bitrate in a control message. In our configuration, the HAS player follows the target bitrate from the Service Manager. However, players also perform bandwidth estimations as a backup. In case download speeds are too low, it might jeopardize the stream's continuity. In case of too low bandwidth, our modified HAS player decides to ignore the target bitrate and rely on bandwidth estimations for adaptation. We also maintain a threshold on the buffer level (10 seconds in our implementation) to switch between bandwidth estimations and target bitrate. The player has the freedom to select lower bitrate segments during the initial buffering phase to start playing sooner. The advantage of this approach is that it allows players to make informed adaptation decisions while maintaining their flexibility and robustness to changing network conditions.

2.4.4 Control Message Flow

The Service manager sends each HAS player its target bitrate in a control message. Besides target bitrate signaling, the Service Manager also instructs the Network Controller to configure traffic control, using the target bitrate as an indication for

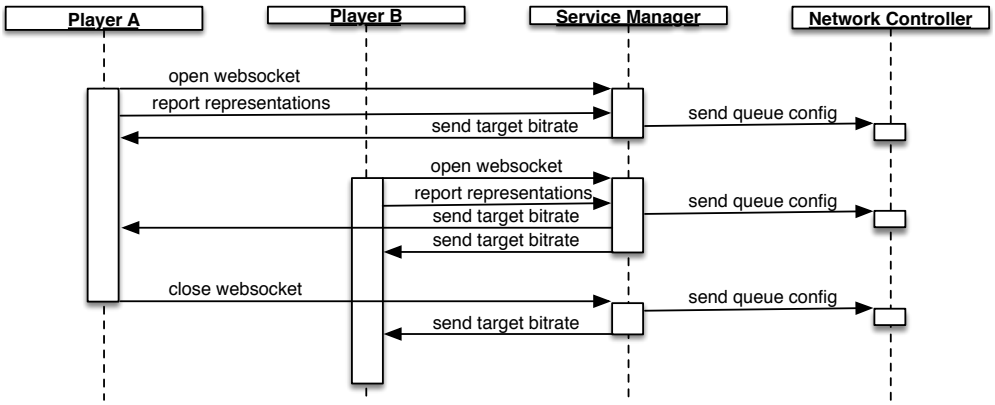


Figure 2.5: Message sequence between HAS player, Service Manager, and Network Controller.

the size of the queues. The sequence of messages between HAS players and Service Manager, and between Service Manager and Network Controller is shown in Figure 2.5. It shows the full cycle for Player A (both start and stop) and the actions taken when Player B starts while Player A is still active.

2.5 Related Work

The design and the two implementations of the Control Element focus on the network link close to the HAS players. The Control Element determines how active HAS players are supposed to use this network link. In this architecture, the intelligence is thus largely located in a component inside the network. As such, the Control Element is a network-based optimization technique. However, in relevant works on HAS optimization, the focus may also lie on different points in the HAS delivery pipeline, such as the client and server. Figure 2.6 shows a classification of HAS optimization techniques. This classification is consistent with the high-level classification schemes of Petrangeli et al. in [103] and Bentaleb et al. in [14]. At the top layer of this classification, HAS optimization approaches are divided by the elements in the HAS delivery pipeline where the intelligence resides: client, server, or network-based optimization approaches. The middle layer reflects the different optimization approaches for each of the elements. The bottom layer contains the commonly used optimization techniques.

Although the approaches in the three classes aim at different points, they share the common goal of improving video streaming quality, addressing the bitrate, stability, and fairness of HAS players in shared networks. Most of the relevant works that

Figure 2.6: Classification of HAS optimization techniques.

are discussed in the remainder of this chapter attempt to reduce the competition for bandwidth between HAS players and cross-traffic.

The client-based approaches focus on the improvement of adaptation algorithms. The selection of video bitrates can be improved by better estimations of the available bandwidth, by taking into account player state (e.g., buffer level), or through including network and context factors. The availability of network-layer information enables cross-layer adaptation approaches. Having context information allows the use of history-based heuristics.

Server-based approaches focus on actions that a server can perform to improve streaming (i.e., server assistance), or on strategically placing the servers and caches to create an optimal route from server to client. Servers may assist HAS players in detecting oscillations in the bitrate of video stream. Server localization focusses on the distribution of CDN nodes, proxy servers and caching strategies.

Techniques that optimize from a network perspective include low-level approaches that cover modifications of the transport protocols or using multiple paths between server and client. Higher-level approaches tend to focus more on managing network resources. Network resource management is done indirectly using traffic control, or directly via guidance (i.e., target bitrate signaling).

This thesis addresses a form of network-based optimization of HAS, introducing the Control Element that assists HAS players from within the network. We describe and discuss two different implementations of the Control Element: a proxy server implementation that is in the media delivery path and an SDN-based implementation where HAS players communicate with the Control Element via a separate channel. As such, this thesis compares two different forms of bitrate guidance. Using the SDN-based implementation, we further investigate the exchange of assistance messages (i.e., when and how often to transmit the messages), the client-side behaviour when receiving a target bitrate recommendation, and the impact on streaming performance of traffic control mechanisms in the network. This thesis thus goes beyond the concept of assistance messages (e.g., as described by Thomas et al. in [132]) and covers the procedures in both client and network that are necessary for optimizing HAS delivery. Furthermore, we present a novel method for analyzing relevant bandwidth sharing policies using Markov-based performance modeling, covering network-based optimization for HAS from both an empirical and theoretical angle.

The remainder of this chapter is structured following the classification of client-, server-, and network-based optimization techniques for HAS.

2.5.1 Client-based Approaches

The largest body of work on HAS optimization focusses on improvement of the HAS client, in particular the adaptation algorithm. Client-based approaches follow the HAS design rationale that all the intelligence and control over the stream resides in the client. These client-based approaches address the shortcomings of the conventional adaptation algorithm. The conventional algorithm uses the download speeds of the last video segments as an indication of the available bandwidth for downloading future video segments. In this algorithm, there is a direct relation between the download speed of past video segments and the video bitrate for future segments. The video bitrate for the next video segment is the highest available bitrate that is lower or equal to the measured download speed, possibly including a safety margin on the measurements.

Depending on implementation details, the conventional algorithm may only use the download speed of the last segment, allowing the highest flexibility and targeting networks with large fluctuation in available bandwidth. Alternatively, the conventional algorithm may use a (weighted moving) average of the download speed of several past video segments. For example, Rainer et al. implement an adaptation algorithm based on a weighted average in the DASH-JS video player [108]. In their algorithm, the download time for the manifest serves as an indication of the bitrate of the first video segment.

Using (weighted) averages is intended to increase the stability of the video quality, reducing peaks and drops in video quality as a result of sudden short download speed changes. Increasing the influence of past video segments increases quality stability, but it reduces the ability to react to sudden drops or peaks in available bandwidth. Responding too slow to a drop in available bandwidth creates the risk of video freezes and responding too slow to a bandwidth increase affects network utilization.

The assumption in the conventional adaptation algorithm is that download speeds of past segments are a good predictor for future available bandwidth. While the truth of this assumption largely depends on the network and the load it endures, it is also important to assess the accuracy of the bandwidth estimation method considering the current available bandwidth. Bandwidth estimations based on download speed could be wrong, especially when multiple HAS players share a network link with insufficient bandwidth for each player to stream at the highest bitrate in the manifest.

Akhshabi et al. demonstrated in [7] that HAS players generate so-called ON-OFF download patterns when downloading the video segments. Most often, the time to download a single video segment is slightly shorter than the playback time of this

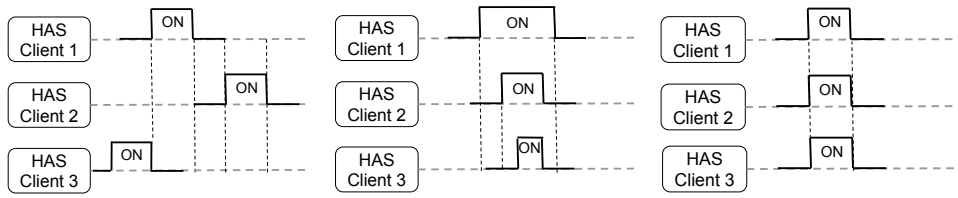


Figure 2.7: Different segment download patterns: unsynchronized, overlapping, and synchronized downloads [14].

segment. The difference between download and playback time is caused by the offset between the available bitrates in the manifest (a fixed amount with discrete increments) and the available bandwidth. HAS players use this difference to fill the internal playback buffer up to a certain threshold (e.g., 30 seconds of video). The state of the HAS player when filling the playback buffer is referred to as the buffer-filling state. Once the buffer is full, the HAS player enters the steady state. In this state, the HAS player maintains the buffer by downloading the same number of video segments as it plays out in the same time frame. Between two segment downloads, the HAS player pauses, creating a period of activity (i.e., the ON period) followed by a period of inactivity (i.e., the OFF period).

When multiple HAS players share a network connection, the level of synchronization between the ON-OFF periods of the different HAS players determines the accuracy of download speed as a measure for available bandwidth. As shown in Figure 2.7, download patterns may be unsynchronized, overlapping, or synchronized. Only when the ON-periods of all HAS players are synchronized, the HAS players can rely on TCP's fair sharing of resources, and obtain the correct available bandwidth. When a HAS player downloads a video segment while other HAS players are inactive, the active player overestimates the available bandwidth. As a response, the HAS player will increase the video bitrate to a representation that is too high, following the conventional adaptation algorithm. Once realizing that this bitrate is too high, the video player will lower the video bitrate. However, when all players are at the original bitrate, the process may repeat and the HAS players oscillate between high and low quality video representations.

The first group of improved adaptation algorithms still relies on download speeds. In [85], Liu et al. present an adaptation algorithm that reduces the influence of short-term throughput variations. The algorithm uses a smoothed segment fetch time (SFT) instead of the instantaneous download rate. Based on the smoothed rates, the algorithm increases the video quality step-wise when bandwidth becomes available. Decreasing the video bitrate when bandwidth becomes limited is aggressive. Using simulations, Liu et al. show that the algorithm eventually matches the available

bandwidth and effectively prevents buffer under-runs. In a later work, Liu et al. improve their adaptation algorithm, using a ratio of the expected segment fetch time (ESFT) and the measured SFT to detect network congestion [86]. The authors present two versions of the adaptation algorithm. One version is to be used when sequentially fetching video segment, while the second version covers parallel downloads from multiple CDN nodes. Tian et al. also address an adaptation algorithm designed to access content from multiple CDN nodes [133]. The authors use the machine learning method Support Vector Regress (SVR) to estimate TCP throughput.

In [81], Li at al. present the Prove-AND-Adapt (PANDA) algorithm. This algorithm specifically targets the ON-OFF periods of HAS players. When a HAS player is in the ON-period, it measures the TCP throughput. In the OFF-period, PANDA continuously probes the network by incrementing the sending rate or back-off in case of congestion. Although operating at the application layer, the mechanisms of probes resembles TCP congestion algorithms. The probes are used to fine-tune the bandwidth measurements. Later, in [82], Li et al. formulate video quality adaptation as an optimization problem. Assuming visibility into the incoming video within a finite horizon, the authors propose an optimization solution that uses an online algorithm based on dynamic programming. The algorithm uses the video buffer to account for network bandwidth variations and video quality variations. The adaptation algorithm is integrated into PANDA [81] and yields consistent quality without risking video freezes.

Where the PANDA algorithm uses probes to handle ON-OFF download patterns, Jiang et al. introduce the FESTIVE algorithm based on the insights which the authors gained when studying multiple HAS players that share a network connection [69]. FESTIVE reduces the impact of ON-OFF traffic patterns by randomly scheduling segment downloads. The ELASTIC algorithm in [31] avoids ON-OFF traffic patterns altogether. By generating a long-lived TCP flow instead of periodically requesting video segments, the algorithm can ensure fairness between multiple HAS players in the same network. Fairness in ELASTIC means that each HAS player receives the same bandwidth. However, the algorithm cannot account for different type of players with different video quality requirements. Similarly, the throughput-friendly DASH (TFDASH) algorithm avoids OFF-periods by perfect-subscription or over-subscription of bandwidth, always fully utilizing the network link [149]. In combination with a probability driven adaptation logic the TFDASH algorithm realizes video quality stability and fairness.

The algorithms above obtain the bandwidth directly through measuring the download speeds. Other algorithms obtain the download speeds indirectly. For instance, the piStream algorithm does not directly use bandwidth estimations, but uses infor-

mation from the physical layer of LTE networks [142]. By monitoring the resource allocation of an LTE base station, the piStream algorithm extracts an estimation of the available bandwidth. Experiments show that the piStream algorithm achieves a high video quality with minimum video freezes. Applicability of this approach may be limited, because it is limited to LTE networks and uses LTE specific information that might be shielded from a HAS client.

HAS clients might also use alternative sources to obtain throughput information. In [53], devices for video streaming are fitted with GPS receivers. The devices collect download speed measurements and report this to a central server. HAS players can query this server in order to better predict future mobile bandwidth based on the expected route. Similarly, Yao et al. use bandwidth maps to drive an adaptation algorithm that can be used for clients with high-speed mobility [144]. Building on top of the dataset provided in [53], Taani and Zimmerman present an adaptation algorithm that also takes into account the time a bandwidth measurement was taken [129].

Other algorithms may use the occupancy level of the buffer to determine the video bitrate for future video segments. Miller et al. present an adaptation algorithm that combines throughput-based estimations while controlling buffer occupancy [91]. Through this combination the video quality could be maximized under the condition that video freezes are minimized. The algorithm also minimizes the streaming start-up time. An implementation of the algorithm is evaluated under real-world conditions, showing stable and fair operation. Müller et al. also combine a buffer-based adaptation algorithm with a set of client metrics. The client metrics are intended to detect oscillation of video quality and drive a compensation mechanism for erroneous adaptation decisions. Compared to throughput-based estimation algorithms, the proposed algorithm is able to prevent oscillations. In the SARA algorithm, the network throughput, buffer occupancy and segment size variations are combined to predict future segment download time [72].

Spiteri et al. describe the BOLA algorithm in [126]. The authors formulate bitrate adaptation as utility maximization problem and use Lyapunov optimization techniques to reduce video freezes and maximize video quality. By strongly relating the video bitrate selection to the buffer occupancy, BOLA does not require network throughput predictions. The BOLA algorithm, including minor modifications to make the algorithm useful in practice [127], is implemented in the Dash.js reference player [37]. As such, the algorithm is used in production environments and used as benchmark algorithm. In this thesis, the performance of the Control Element assisted HAS players is also compared to the performance of BOLA.

In [13], Beben et al. describe the ABMA+ algorithm, an improved version of the ABMA algorithm [141]. AMBA+ uses a combination of segment download times and buffer occupancy. Using a pre-computed buffer map, the algorithm selects the video bitrate based on the probability that a video freeze will occur. As such, the HAS player selects the highest video quality that still ensures smooth playback. By pre-computing the buffer map, heavy online computation can be avoided increasing the applicability of the algorithm.

In [61], Huang et al. present a fully buffer-based adaptation algorithm. In later work, the authors discover that estimating download speeds may be useful during the buffer-filling state, but that they are unnecessary when HAS players are in steady state [62]. Based on this finding, the authors present the BBA-0 and BBA-1 algorithms. These algorithms rely on bandwidth estimations during start-up, but change to buffer-based video quality adaptation when the buffer occupancy reaches a certain level.

Yin et al. explore the trade-offs between bandwidth estimations based on download speed and using buffer occupancy in [145]. They investigate how each of the classes operates under different conditions, including balancing between the different QoE objectives. Based on the insights, the authors develop a model predictive control algorithm that combines throughput information with buffer occupancy. The algorithm is based on a predictive control-theoretic model to reason about the adaptation strategies. Sobhani et al. also combine throughput information with buffer occupancy, but use Fuzzy logic to combine the two metrics [125]. This adaptation algorithm specifically addresses fair sharing of bandwidth among HAS players in the same network. In [138], Wang et at. present a spectrum [150] based adaptation algorithm that combines both historical throughput information with buffer occupancy information, specifically targeting networks with cross traffic.

The authors in [143] present an adaptation algorithm that uses model-based buffer occupancy estimations. The authors model a HAS player as M/D/1/K queue, predicting the buffer occupancy in a HAS player based on video bitrate, network throughput, and the total capacity of the buffer. In [148], Zhou et al. formulate bitrate adaptation as MDP, focussing on maximizing the video quality in the long term. Chiariotti et al. present an online learning adaptation strategy for HAS that selects the video bitrate based on a long term reward [29]. The authors formulate a Markov Decision Process (MDP) optimization problem for the selection of bitrates. System dynamics are not known and obtained through reinforcement learning techniques. Mao et al. also use reinforcement learning techniques to train a neural network model for adapting the video quality [88]. Based on continuous observations collected by the HAS players, the Pensieve algorithm adapts to the environment. Gadaleta et al. combine reinforcement learning with deep learning

in D-DASH [48]. The authors propose different learning architectures, combining feed-forward and recurrent deep neural networks.

Although several of the before mentioned adaptation algorithms target fairness, all these algorithms are designed to operate from the single perspective of the HAS player and try to increase the video quality when possible. Bentaleb et al. acknowledge this aspect as a limitation of HAS adaptation algorithms in [16]. As a response, the authors present Game Theory Adaptive bitrate streaming. In this algorithm, HAS players reach a consensus about the selected streaming bitrate, enabling fair sharing of network bandwidth.

2.5.2 Server-based Approaches

In contrast to client-based optimization approaches, server-based approaches focus on the role of the server, either via adding intelligence to the server, or intelligently positioning servers in the network. The work of Akhshabi et al. in [8] falls into the first category of adding intelligence to the server. In this work, the authors add oscillation detection to the server. Once the server detects one, or more, oscillating HAS players it activates traffic shaping. The shaper dynamically adapts to the video bitrate that a HAS player would request when operating stable. The approach from Akhshabi et al. requires stateful servers. When frequently switching between servers in the CDN, the server is no longer able to detect oscillations, making this approach ineffective.

De Cicco et al. propose the Quality Adaptation Controller (QAC) which specifically targets live streams [39]. QAC is designed based on feedback control theory and throttles the video quality in the server to match the available bandwidth. It enables fair sharing of bandwidth between concurrent video flows or greedy TCP connections. Detti et al. also target live streaming, but focus on the challenges of bitrate adaptation in the presence of caching proxy servers [40]. The authors introduce so-called tracker functionality. The tracker, operating both in the proxy and the origin server, enables HAS players to share status information (e.g., selected video bitrate, download speeds) to make better adaptation decisions. Similar to the approaches above, the tracker functionality requires changes to the servers such that they can maintain state (i.e., a list of all connected HAS players).

Although experiments indicate that the approaches described above are effective, they go against the design rationale of HAS. One of the reasons for HAS to have the intelligence in the client is the need for stateless servers (i.e., to increase distribution scalability). Adding intelligence to the server violates this requirement. However, these works show that a node outside the HAS player can be useful to improve video streaming performance.

Besides the role of the server, the location of the server with respect to the client also influences the performance of HAS. In [2], Adhikari et al. investigate the influence of different CDN nodes on streaming performance. By measuring the performance of CDN nodes from Netflix and Hulu, the two leading content providers in the USA and Canada, the authors discover large differences between nodes. Furthermore, the work shows that CDN nodes are selected without taking the network conditions or the user-perceived video quality into account. The authors also propose a CDN selection strategy that bases the node selection on measurements.

Bouten et al. similarly argue that video quality selection should also include selecting the best performing video server in the CDN [23]. The authors propose a probability-based search strategy to explore the available servers and their characteristics. The search strategy is build into the HAS player and uses buffer occupancy information to prevent video freezes. Furthermore, the search strategy is compatible with any video quality adaptation algorithm. Wang et al. take a similar approach in [137]. The authors argue that the performance of CDN nodes in not only determined by the network performance, but also on resource contention between cloud virtual machines. These two factors create highly varying performance between nodes. However, the HAS player itself has the best perspective on delivering a high QoE to the user. Therefore, the authors propose a server selecting algorithm that is build into the HAS player. Simulation experiments and experiments using the Google Cloud infrastructure show significant performance improvements when selecting the best video server.

In [27], Carlsson and Eager address the problem of content placement on edge caches. Through studying a large 20-month dataset of YouTube video requests, the authors develop a novel workload modeling approach to study the performance of edge caching policies. Based on the model, the authors discover that there is significant room for improvement, but they are reserved about the improvements because they might be unachievable in practice.

2.5.3 Network-based Approaches

The third class of optimization techniques involve network elements for optimizing video streaming performance. Similarly to server-based approaches, network-based approaches recognize that nodes outside the HAS player can improve the HAS streaming experience. In contrast to server-based approaches, network-based optimizations do not require changes to the server architecture. Because most of these technologies can be deployed on network elements close to the HAS player (e.g., in the home Internet gateway), scalability of video delivery remains unaffected. Ahmad et al. also argue that a collaboration between content providers and network providers is essential to maximize revenue [4]. Using a collaborative

model, the authors show that revenue is largely dependent on user churn, which is in itself affected by network performance.

One of the earliest works in this class of optimization approaches is from Houdaille and Gouache in [59]. In this work, the authors target the problem of video quality instability for HAS clients that share a household Internet connection. Instead of equally sharing the network's bandwidth, the HAS players compete for bandwidth at the cost of the other players. By introducing traffic shaping at the residential gateway, the authors avoid the bandwidth competition and reduce the number of quality switches. Although it is shown that this approach is effective, the implementation details (i.e., how does the gateway detect HAS players) is not part of the work. Mansy et al. address these issues with their Video Home Shaper (VHS) [87]. In VHS, a traffic filter extracts the video requests and obtains information about the video and the device that made the requests. A session manager computes a fair distribution of bandwidth using a device dependent QoE metrics. The video flows are regulated through bandwidth shaping at the residential gateway. The VHS implementation also targets networks with cross-traffic.

In [78], Kua et al. investigate the performance of HAS delivery when using Active Queue Management (AQM) as traffic control mechanisms. Through experimentation, the authors evaluate and compare several AQM schemes, including PIE [100], FQ-PIE [116], CoDel [96], FQ-CoDel [54]. The paper discovers that the PIE scheme provides the best streaming performance when latencies are high. When the scheme is combined with a FlowQueue [54] scheduler for flow isolation, FQ-PIE protects DASH streams in networks with cross-traffic.

Bouten et al. propose an in-network video quality adaptation approach using proxy servers [24]. The authors claim that the streaming experience can be improved when intelligence is added to the network. In their implementation, a shared network link is governed by a proxy server that overrides adaptation decisions from the HAS players. The proxy server alters the HAS manifest, limiting the video representations that a HAS player can request. The available network bandwidth is divided over the players using a linear programming based solution. The approach from Bouten et al. shows similarities to the proxy server implementation presented in the first part of this chapter.

Petrangeli et al. extend the proxy server approach to a system of proxy servers [102]. HAS traffic passes several coordinating proxy servers before reaching the video server. The proxy servers inform each other about network resource sharing using a so-called Fairness Signal. The system of proxy servers allows to fairly share network bandwidth in deployments with multiple network bottlenecks. In [106], Pu et al. use a proxy server for improving streaming performance in mobile

networks (e.g., HSDPA+ or LTE networks). The simulation-based experiments in this paper show that splitting the TCP flow into two flows between the server and the WiDASH proxy, and between the WiDASH proxy and the HAS player increases the average throughput in the network without sacrificing fairness.

Mok et al. leverage proxy servers for assisting the adaptation algorithms in HAS players [92]. Using an in-network measurement proxy, HAS players will be better informed about the available bandwidth. The HAS players use an altered adaptation algorithm that takes the measurements from the proxy as an upper-bound for the video bitrate. Combined with a smooth adaptation algorithm that gradually changes the video quality, the streaming experience is improved. The authors verify this approach using subjective experiments.

An alternative use of a proxy server is made with BUFFEST [77]. Instead of inspecting the (HTTP) contents of the video flows, BUFFEST obtains information from the network interface of the proxy server. Using classification and prediction of TCP/IP packet-level information, BUFFEST is able to detect video flows and accurately estimate the HAS players' buffer levels. This information may for example be used to boost the network throughput for clients with a buffer below a certain threshold. Because the BUFFEST algorithm does not require content inspection, this approach is also suitable for HAS that is delivered over HTTPS.

El Essaili et al. address the mobile network scheduler with respect to HAS [44]. The authors show that the default LTE scheduler results in unsatisfactory streaming performance. Therefore, an altered LTE scheduler is proposed in combination with a proxy server that rewrites segment requests to match the video bitrate with the scheduled resources. As such, the proxy server uses the same mechanisms to change the video bitrate of HAS player as the proxy server that is described in the beginning of this chapter. Consistent with our findings, El Essaili confirm that request rewriting does not cause problems with unmodified and of-the-shelf HAS players. In [28], Chen et al. utilize a resource manager for HAS streaming in mobile networks. The resource manager computes the optimal video bitrate for each user in a cell, enforces the bitrates using QoS bearers with the mobile network scheduler and per-flow traffic shaping. By including the scheduler, the AVIS system is able to cope with the dynamics of the mobile network. Instead of using a proxy server, AVIS performs deep packet inspection to learn about HAS manifests and segment requests. Optionally, AVIS can be used with wireless network slicing [75] to separate HAS traffic from other traffic.

In [109], Ramamurthi and Oyman focus on resource scheduling in mobile networks to reduce video freezes. Based on buffer occupancy information received from the HAS players, the LTE scheduler boosts the performance of HAS players with low

buffer levels. Simulation experiments show a significant reduction of video freezes in LTE environments. In [110], the authors extend their system with a maximum video bitrate based on buffer occupancy information. Based on the same concept, Zahran et al. present Stall Aware Pacing (SAP) in [147]. SAP realizes a consistent streaming experience for HAS players sharing a mobile network based on buffer occupancy information provided by the HAS players combined with network level information. In later work, the authors present an improved version, Adaptive SAP, realizing an additional performance gain [146].

Under the assumption that network nodes not always operate with full knowledge about the network, D'Aronco et al. introduce a controlling node for coordinating HAS players in shared networks [36]. Based on measurements that HAS players share with the controlling node, the node can effectively detect network level congestion. By sending a congestion level signal to the HAS players, they can optimize their video segment requests. The authors formulate the bandwidth sharing problem as Network Utilization Maximum [99], maximizing the overall user satisfaction. The problem of dividing bandwidth in an overseeing network element is targeted by Joseph and De Veciana in [71]. The authors present a simple asymptotically optimal online algorithm that realizes the trade-offs among video quality, video quality switches, rebuffering, and fairness. The algorithm, named NOVA, distributes video quality adaptation between a network controller and HAS players. Similarly, but using simpler QoE models, the problem of dividing bandwidth is addressed in [70, 22].

Realizing these bandwidth sharing policies can be done with proxy servers and mobile network schedulers as overviewed above. Alternatively, SDN [76] techniques can be leveraged. In [104], Petrangeli et al. use SDN to prioritize the delivery of HAS segments based on the clients state. This approach has the effect to reduce video freezes due to congestion. Georgopoulos et al. propose the OpenFlow-assisted QoE Fairness Framework that uses SDN to create a control plane orchestrator [50]. The presented solution uses a network inspector and manifest parser to obtain information about the client. A utility manager and optimization function configure the network's flow tables and feed video quality selection information to the HAS player. The framework supports different types of viewing devices (e.g., TV, tablet, smartphone) and attempt to realize a consistent image quality of each type of device. In a later work, the authors present a solution for in-network QoE measurements using SDN [46].

In [18], Bentaleb et al. present SDNDASH. SDNDASH uses resource allocation based on metering in OpenFlow-enabled switches [89]. This implementation works well in networks with a varying capacity, because the resource manager performs bandwidth estimations on the network interface. The bandwidth estimator uses the

PANDA algorithm [81] for estimating the available bandwidth for all HAS players. The problem of dividing the bandwidth based on QoE parameters is solved using online Model Predictive Control [26]. In later versions of the work, the authors improve the scalability of the bandwidth division algorithm and add support for device heterogeneity [15, 17].

Cofano et al. use SDN to build a Video Control Plane (VCP) [32, 33]. VCP is intended to coordinate multiple HAS players sharing a network connection. The authors investigate the use of traffic control, bitrate guidance, and the combination of the two. As such, their implementation shows great similarities to the SDN-based implementation of the Control Element presented in the first part of this chapter. The authors conclude that combining traffic control with bitrate guidance does not provide a performance gain. We refute this statement. In the next chapter, we show with experiments in a Wi-Fi network that the combination of traffic control and bitrate guidance outperforms configurations with only a single mechanism activated. In networks with cross-traffic, only using bitrate guidance is not sufficient. Traffic control is essential to ensure that HAS players can stream at the target bitrates that they receive from the Control Element.

Where the works described above focus on network bandwidth sharing, SDN can also be used to route traffic in the backbone network. Wan et al. experiment with SDN in the GENI [49] testbed in [139]. The authors focus on optimizing the delivery of live streams, routing the streams over different routes in the network using SDN techniques. Nam et al. follow a similar approach, rerouting streams and increasing network reliability using Multi-Protocol Label Switching (MPLS) [121] traffic engineering [93]. When a HAS player experiences a video freeze, a new path is selected based on the available bandwidth on the different routes. This approach seems particularly useful during busy-hours. In [19], Bhat et al. combine information about the available bandwidth on different paths with cache status information of CDN nodes. The authors verify the approach in an experiment with 15 HAS players in the CloudLab [114] environment.

The network-based approaches, including the solutions in this thesis, have in common that they show that applying knowledge and control from within the network improves the streaming performance compared to solutions only addressing the HAS player. In this thesis, we look further into the interactions between the HAS player and the network. For example, most of the SDN-based implementations described above require inspection of the manifest file. Depending on the owner of the network element, the manifest exposes viewing activity to a third party. However, having access to the users' viewing activity is not necessary for effective management and violates the users' privacy. Furthermore, when using encryption (i.e., HTTPS), access to the manifest is shielded, rendering the approaches ineffec-

tive. A list of the available (or preferred) video quality representations for each client is sufficient. This fact motivated us to develop a solution that uses as little information about the video players as necessary. The resulting system, described in the first part of this chapter, uses a separate communication channel for control information, making our solution respecting the users' privacy.

Furthermore, we show that our SDN-based Control Element is lightweight and scalable, allowing a single Control Element to assist many HAS players, as demonstrated in our experiments with up to 600 concurrent HAS players. The thesis also investigates the impact of bitrate guidance, different traffic control configurations (e.g., comparing rate-limiting versus providing a guaranteed minimum rate), and the interplay between HAS players, the mechanisms, and their configurations. Through an extensive set of evaluations performed using real devices, we create the understanding how each part contributes to the streaming performance improvements. Besides investigating the mechanisms, this thesis makes the link between the mechanisms and the bandwidth sharing policies. A model-based study and validation experiments show the role of the sharing policy on the resulting performance.

2.5.4 Standardization: DASH and SAND

The HTTP Adaptive Streaming technology is standardized as Dynamic Adaptive Streaming over HTTP (DASH) in the ISO/IEC standard 23009-1 "Information technology – Dynamic adaptive streaming over HTTP – Part 1: Media presentation description and segment formats" [66]. This standard primarily focusses on the definition of the Media Presentation Description (MPD, i.e., the HAS manifest). In DASH, the XML-based manifest follows a hierarchical structure. At the top layer, the media presentation is divided in Periods. A period represents a part of the video (e.g., a selection of scenes or an advertisement block) and consists of AdaptationSets. AdaptationSets are different streams that, when combined, form a full media presentation. AdaptationSets are the video and audio streams (e.g., multiple languages) and each AdaptationSet contains several Representations. Different Representations in the same AdaptationSet are considered perceptually equivalent. A Representation is a deliverable encoded media stream, formed by a concatenation of Segments. Following this model, adapting the video quality in DASH means changing between representations in an AdaptationSet.

The DASH specification only describes the manifest and segment formats, but the specification does not prescribe player implementations. This allows for different players with different adaptation algorithms being compatible with the DASH formats. The DASH Industry Forum (DASH-IF) provides guidelines on how to implement DASH in [38] together with a reference DASH implementation that

adheres to these guidelines [37]. In parts of this thesis, we make use of the reference DASH player and use media content formatted following the DASH-IF guidelines (i.e., h.264 encoded video in fragmented MP4 container and AAC encoded audio).

The ISO/IEC standard 23009-5:2017 "Information technology – Dynamic adaptive streaming over HTTP – Part 5: Server and network assisted DASH (SAND)" specifies control messages between DASH players, servers, and network elements such as the Control Element [67]. With SAND, Thomas et al. introduce the concept of DASH Aware Network Element (DANEs) [132]. DANEs may be elements for media hosting, such as DASH aware servers or the proxy server implementation of the Control Element in this thesis. Alternatively, DANEs may be network elements similar to the SDN-based implementation of the Control Element.

In the specification, SAND describes the messages and their format that can be exchanged between DASH clients and DANEs (i.e., status messages, metrics, and Parameters Enhancing Reception (PER)), and between DANEs (i.e., Parameters Enhancing Delivery (PED)). With respect to the Control Element in this thesis, the following status and PER messages are the most relevant:

- *SharedResourceAllocation*: This message can be used by DASH player to inform a DANE about the available representation in the manifest. The DASH client may send the full list of available representations, or provide a selection of representations that it is interested in (e.g., to exclude representation that exceed the resolution of the screen).
- *SharedResourceAssignment*: This message is the response from the DANE to the SharedResourceAllocation message. The message contains the DASH representation that the DANE deems most feasible for the DASH client. The SharedResourceAssignment message allows the DANE to share the available bandwidth following a sharing policy. With respect to the Control Element, the SharedResourceAssignment is the SAND equivalent of the target bitrate message.
- *QoSInformation*, *Throughput*: These messages allow the DANE to inform the DASH client about QoS and guaranteed throughput when downloading the next video segment. Compared to the SharedResourceAssignment message, this message allows the DASH client more flexibility to select a video representation (e.g., select a lower bitrate representation in order to fill the playback buffer).
- *Metrics*: The metrics messages allow the DASH client to provide client-based metrics to the DANE. This message can, for instance, be used to inform the

DANE about buffer occupancy, information which we use in Chapter 5 of this thesis.

Similar to the DASH specification, the SAND specification only defines the interoperability points (i.e., the XML-based format of the messages and the messages itself). However, as shown in this thesis, the network optimization for HAS depends on message timing, the procedures following the reception of a message in both HAS client and Control Element[1], the mechanisms for traffic control in the network, the interplay between messages and traffic control, and the bandwidth sharing policy.

[1]At the time of implementing the Control Element in this thesis, the SAND specification was still under development. Therefore, our Control Element does not implement SAND. However, the messages used by our Control Element are comparable to the SAND messages listed above, and the work in this thesis thus also applies to SAND.

3 | Performance Evaluations

The Control Element manages HAS players in shared networks. It obtains knowledge about active HAS players, divides the available network bandwidth, and assists with adaptation by providing feedback to HAS players. As such, the Control Element follows a model of centralized intelligence for sharing network bandwidth. This chapter evaluates the performance of the Control Element and shows that bandwidth sharing based on central knowledge outperforms sharing based on adaptation algorithms in HAS players.

The evaluations start with the proxy server implementation of the Control Element. This proxy server rewrites HTTP requests when HAS players stream at too low or too high bitrates. The evaluations show that rewriting effectively reduces quality switches and it significantly improves fairness between players. The evaluation continues with the SDN-based implementation in a Wi-Fi-based streaming testbed. The bitrate signaling and traffic control mechanisms are assessed both alone and combined. Both mechanisms can improve the streaming performance. However, we will show that the performance is optimal only when bitrate signaling and traffic control are combined. Combining the mechanisms eliminates freezes, increases the video bitrate, reduces quality switches, and restores fairness.

The evaluations end with a demonstration of using the Control Element for specific use cases. We demonstrate that the Control element can distinguish between device types, improve the performance for players with non-optimal Wi-Fi reception, and scale to networks with at least 600 concurrent HAS players.

This chapter is based on publications [K4][K5][K7][K9][K11].

3.1 Introduction

The Control Element optimizes the division of network bandwidth between HAS players and between HAS players and cross-traffic. It features mechanisms to learn about active HAS players on a bottleneck network link. The Control Element collects information about the HAS players (e.g., their playback capabilities) and the stream (i.e., the available representations in the HAS manifest). By collecting this information, the Control Element builds a global view of video streaming activity in a network. Using global knowledge, the Control Element divides the available network bandwidth. This means that the Control Element specifies a target bitrate for each player. The target bitrate is the optimal representation given the current load on the network. The Control Element provides feedback about the target bitrate to the HAS players.

It is the hypothesis that using centralized knowledge and control will improve streaming performance compared to using adaptation algorithms in HAS players. In this chapter, we test this hypothesis by comparing the streaming performance of HAS with the Control Element with the streaming performance of off-the-shelf HAS players in networks without the Control Element. As such, the contributions in this chapter answer the second research question in this thesis:

Research question 2: *(How) do the mechanisms in the Control Element improve video streaming performance in shared networks in terms of video freezes, video quality, quality switches, and fairness?*

Depending on which mechanism is used in the Control Element, the Control Element is more or less effective. Therefore, this chapter covers performance evaluations with the two different implementations from Chapter 2 in different networks types and under varying load. The evaluations start in Section 3.2 with the assessment of the proxy server implementation of the Control Element. The proxy server should be deployed on a network gateway where it manages the shared inbound Internet connection. The proxy server scans all HTTP requests for HAS, and corrects requests from HAS players for segments in a too low or too high bitrate. This solution is transparent and compatible with off-the-shelf HAS players using the HLS manifest format. Section 3.2 evaluates the proxy server in a wired streaming tested. As such, this setup represents a (small) local network where its users share a single Internet connection, which is considered to be the bottleneck.

In Section 3.3, the evaluations turn to the SDN implementation of the Control Element. Different from the proxy server, the SDN implementation uses bitrate signaling and traffic control for managing HAS. Bitrate signaling means that the

Control Element sends the target bitrates to the HAS players. The HAS players will use the target bitrate in a modified adaptation algorithm. Traffic control ensures enough bandwidth for HAS players. Both bitrate signaling and traffic control are evaluated individually in Sections 3.3.3 and 3.3.4. This allows for identifying how each mechanism contributes to the performance increase of HAS with the Control Element. In Section 3.3.5, the Control Element is evaluated with both mechanisms available. The SDN-based Control Element is deployed in a Wi-Fi network with maximum five clients. We replicate the use-case of a home network that is shared by its users for video streaming.

In the evaluations with the proxy server and the SDN-based Control Element, all HAS players are considered equal. This means that each player will receive an equal share of the available bandwidth. In these cases, fairness is defined in the internal sharing policy as equal bitrates. However, one of the advantages of the Control Element is that it can distinguish between different devices, and treat them differently. Instead of achieving fairness on a bitrate level, fairness can be defined closer to the streaming experience of the user. As an example, Section 3.5 demonstrates the ability of the Control Element to handle different devices differently. In Section 3.5, the SDN-based Control Element divides the available bandwidth among HAS players based on the device type. Devices with a larger display and higher screen resolution get a relatively high target bitrate compared to devices with smaller displays. In addition to demonstrating how the Control Element can be beneficial for devices with different form factors, we show that devices with different Wi-Fi network conditions can also benefit from assistance by the Control Element. In Section 3.4, we evaluate HAS players in a Wi-Fi network and compare the performance of client near and further away from the base station.

Most of the evaluations in this chapter are performed with a relatively low number of players. These evaluations represent small networks with a shared bottleneck Internet connection. This bottleneck may for example be created by Digital Subscriber Line (DSL). DSL has a limited capacity and bandwidth sharing between a few HAS players will already be challenging. Although the Control Element shows significant performance improvements in small-scale networks, deployment of the Control Element is not limited to small networks only. Because communication between the SDN-based Control Element and HAS players is lightweight, and executing a sharing policy in the Control Element requires only limited computational power, the Control Element is able to handle many concurrent HAS players. Section 3.6 assesses the scalability of the Control Element in a testbed with up to 600 HAS players. The focus in this evaluation is on the configuration of bitrate signaling and traffic control, ensuring that both mechanisms are still effective given the large number of players.

3.2 Request Rewriting in the Proxy Server

The first evaluation in this chapter assesses the effectiveness of the rewriting mechanism implemented in the proxy server based Control Element. The proxy server processes each HTTP request. If an HTTP request is for a HAS manifest, then the proxy server will also read the manifest to learn about active HAS players in the network. Based on the number of players and available bitrates in the manifest, the proxy server divides the bandwidth of the network connection among the HAS players. Each active HAS player gets a so-called target bitrate. When HTTP requests for video segments come in at the proxy server, the proxy server looks up the bitrate of that request in the manifest. If a HAS player requests a too high bitrate (i.e., higher than the target bitrate assigned by the proxy server), the proxy server rewrites the URL in the HTTP request into one for the same video segment in the target bitrate. The proxy server also rewrites requests for a too low bitrate when it estimates that the buffer in a HAS player is sufficiently full. We maintain a threshold of seven seconds in our implementation.

3.2.1 Wired Streaming Testbed

The effectiveness of request rewriting in a proxy server on improving the streaming performance is evaluated in a virtualized streaming testbed. Each device in this testbed (PCs, routers, and servers) is implemented as a lightweight Linux-based virtual machine. The virtual machines share kernel and libraries with the host operating system, but maintain their own process- and network stack. As such, multiple devices could be hosted on a single server machine, while each device has independent processing and networking. The network connections between the virtual machines are emulated. Network connection properties, such as throughput and latency, are simulated via Linux tc [63]. The configuration of the devices and emulated network connections is made using the Common Open Research Emulator (CORE) [5, 134]. The virtual testbed runs on a host server with a 2.83 GHz Core2-quad processor and 8 GB memory, running the GNU/Linux Debian 6 operating system.

The testbed consists of 17 PCs, two routers, and three servers that are connected as illustrated in Figure 3.1. The shared bottleneck network link is the network connection between the two routers. The bottleneck network connection has a throughput of 8 Mbit/s. This setting represents a network connection that does not have sufficient throughput for all players streaming at the highest bitrates. The round-trip delays between the PCs hosting the HAS players and the servers with the video content is 10ms, 20ms, and 40ms respectively.

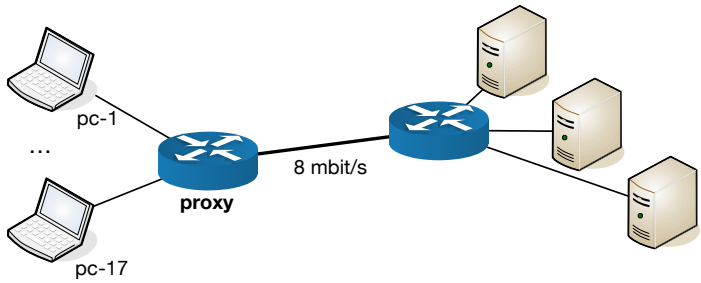


Figure 3.1: Streaming testbed setup.

The proxy server is installed on the router closest to the client machines. Although the throughput of the bottleneck link is 8 Mbit/s, the proxy server is configured with a maximum capacity of 6.8 Mbit/s, thus having a 15% safety margin on the real throughput. The safety margin is required for uninterrupted video streaming over TCP. A theoretical study indicates that the throughput should be twice the video bitrate [136]. However, this indication applies only to non-adaptive streaming. Others claim that a safety margin of 15% is sufficient for HTTP streaming [20], which we follow in this experiment.

HAS players are started on the PC virtual machines. Players are started on free client machines following a Poisson process. Each PC client can only host a single HAS player. This means that the total number of active HAS players is limited to 17. Starting more players would congest the network connection, and may diminish the streaming experience as a result of video freezes. Therefore, when the maximum of 17 players is reached, an 18th player is denied access to the network.

Although the proxy server is compatible with the Apple's QuickTime Player [12], we use a custom HAS player that does not render and display the video content to reduce CPU load and memory usage. The custom player is implemented in the Python programming language and uses an aggressive adaptation algorithm based on the throughput of past video segments. The player estimates future bandwidth as the weighted average ($\alpha = 0.75$) of the throughput of the last two segments. It selects the video bitrate that is lower or equal to the estimated bandwidth for the next video segment.

All HAS players stream a video with a duration of 140 seconds which is encoded at $\{400, 720, 1020, 2300, 4200\}$ Kbit/s. The video is segmented into 4.0 second segments using the HLS format [101]. The players randomly select the server for the video content. All three servers have the same manifests and video segments available. The Web servers run off-the-shelf Apache 2.2.22 software [11].

3.2.2 Performance Metrics

The video segment URLs are formatted to contain all information required to measure the streaming performance. The URLs include an identification number of the HAS player, an identifier of the video stream, the segment index, and the requested video bitrate. We collect HTTP traces with tcpdump [131] that is running on the router next to the HTTP servers. Based on the HTTP traces the following performance metrics are obtained:

- **Video quality**: The mean video bitrate in Kbit/s. The mean is computed over all bitrates of the requested video segments for all players in an experiment.
- **Quality switches**: The number of changes in video quality per stream.
- **Unfairness**: Unfairness is based on the adaptation of the Jain's fairness index [69] used by Jiang et al. to assess the performance of the Festive adaptation algorithm. The adapted Jain's fairness index is defined in Equation 3.1, where n is the number of players and q_i is the bitrate for each player. Unfairness values are in the range $[0, 1]$. A lower value means better fairness.

$$unfairness = \sqrt{1 - \frac{(\sum_{i=1}^{n} q_i)^2}{n \cdot \sum_{i=1}^{n} {q_i}^2}} \tag{3.1}$$

During the experiments, there is no background traffic. The HAS players generate all traffic on the bottleneck link. By limiting the maximum number of concurrently active HAS players to 17, the network never overloads. Because both the proxy server and the throughput based adaptation algorithm avoid video freezes, the analysis of video freezes is not part of the evaluation.

3.2.3 Performance Evaluation

We demonstrate the effectiveness of request rewriting in the proxy server by comparing the performance of HAS players in a testbed with, and without, the proxy server enabled. At the clients, video players start the test video sequence following a Poisson process with arrival intensities: $\lambda = 0.020$, $\lambda = 0.030$, and $\lambda = 0.045$. For each condition, a 24-hour experimental run is performed, both with the proxy server enabled and disabled. Table 3.1 provides a summary of the streaming performance.

The results show that for $\lambda = 0.020$ the number of switches is lower when using the proxy server, from 12592 switches to 2761 switches respectively. Accounting for the small difference in the total number of players, this is a reduction of 77%. For $\lambda = 0.030$, a similar reduction in the number of bitrate switches is observed.

	$\lambda = 0.020$	$\lambda = 0.030$	$\lambda = 0.045$
Proxy server enabled:			
Players	1631	2422	3751
Bitrate switches	2761	4197	9698
Mean unfairness	0.0099	0.0104	0.0132
Mean bitrate (kbit/s)	1543	1198	907
Proxy server disabled:			
Players	1674	2470	3783
Bitrate switches	12592	19787	34336
Mean unfairness	0.2107	0.2485	0.2607
Mean bitrate (kbit/s)	2242	1743	1208

Table 3.1: Comparison of the streaming performance between the proxy server and the rate-based adaptation algorithm in HAS players.

For $\lambda = 0.045$ the proxy server reduces quality switches by 72%. The distribution of number of quality switches per player is shown in Figures 3.2 and 3.3. For $\lambda = 0.020$, none of the players switches more than six times with the proxy server enabled. In contrast, the maximum number of switches without the proxy server is 18. For $\lambda = 0.045$, a similar effect is observed. Although quality switches are slightly more likely to occur, the number of quality switches is still significantly lower with the proxy server enabled compared to when the proxy server is disabled.

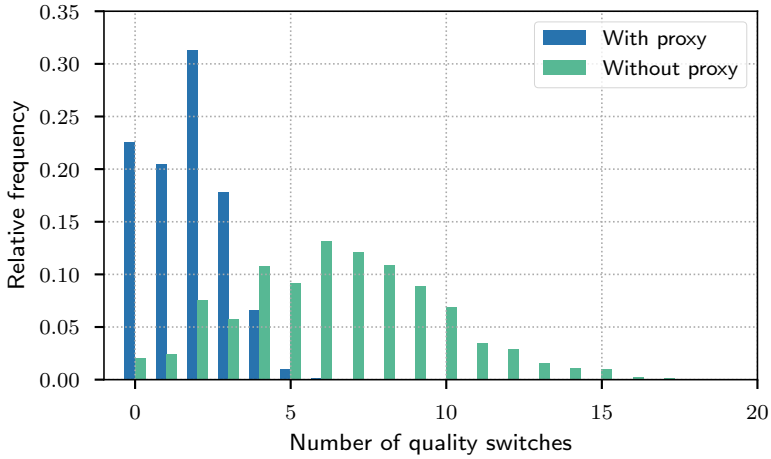


Figure 3.2: Comparison of the distribution of the number of quality switches per stream with the proxy server enabled and disabled for $\lambda = 0.020$.

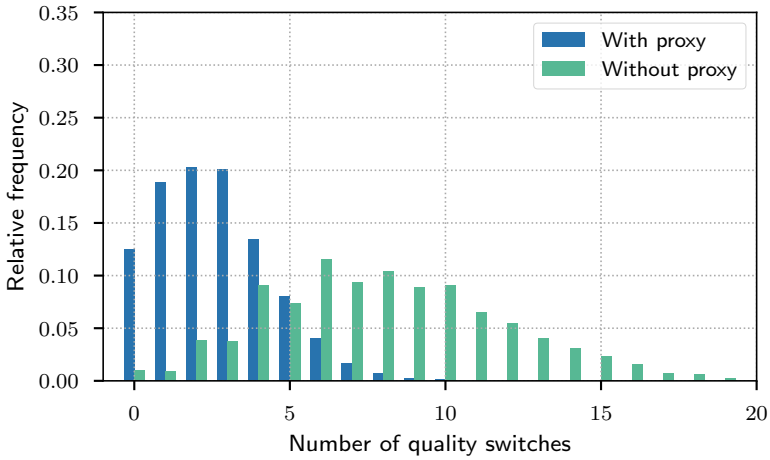


Figure 3.3: Comparison of the distribution of the number of quality switches per stream with the proxy server enabled and disabled for $\lambda = 0.045$.

Enabling the proxy server also results in a 95% reduction of average unfairness. For example, for $\lambda = 0.045$, the average unfairness goes down from an unfairness value of 0.261 to an unfairness value 0.013. For the other two test conditions, $\lambda = 0.020$ and $\lambda = 0.030$, the reduction in unfairness is similar. Figures 3.4 and 3.5 show the cumulative distribution of unfairness. They show that with the proxy server, HAS players stream the same video bitrate for more than 93% of the time (i.e., an unfairness value of zero). The reason that there is little unfairness is that segment requests are not aligned. Decisions from the proxy server to change the video bitrate need at most one segment size to propagate. The HAS players will select the appropriate target bitrate when requesting the next segment. With the proxy server disabled, the range of selected bitrates by the players spreads over the available bitrates. The wide variety of selected bitrates results in a much higher unfairness. Compared to using the proxy server, the time spent in a fair state is between 2.4 and 4.9 times lower without the proxy server.

The average bitrate with the proxy server enabled is around 30% lower. For example, at $\lambda = 0.020$, the bitrate without the proxy server is 2242 kbit/s, compared to 1543 kbit/s with the proxy server enabled. The fact that we avoid congestion by using a 15% safety margin partially explains the lower bitrate. The bitrate is also lower because the proxy server enforces fairness. Strict fairness typically comes at the price of lower utilization.

Given the results in this section, we confirm the performance problems in HAS. We observe video quality and instability, similar to the results in [9] and [7]. Based on

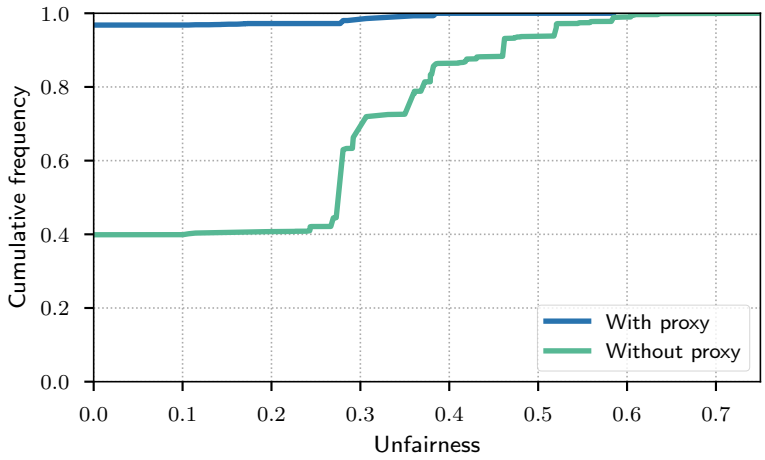


Figure 3.4: Comparison of the cumulative distribution of unfairness with, and without, the proxy server for $\lambda = 0.020$.

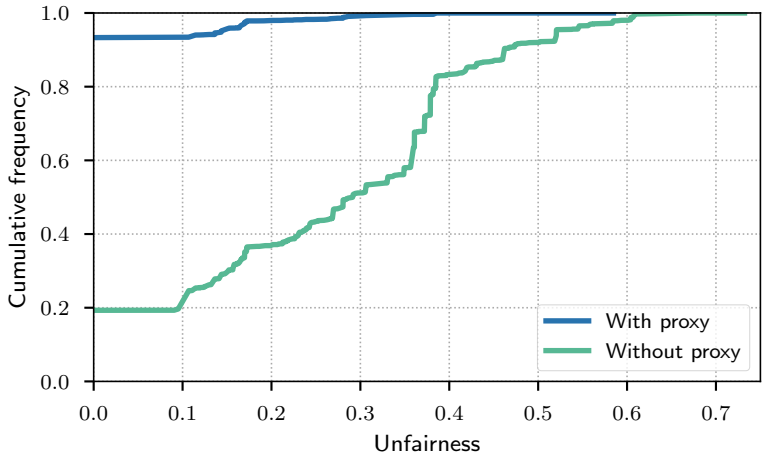


Figure 3.5: Comparison of the cumulative distribution of unfairness with, and without, the proxy server for $\lambda = 0.045$.

the results, it can also be concluded that the proxy server is a competent solution to the performance problems in HAS. Both the number of video quality switches and unfairness is significantly lower when using the proxy server. Because the proxy server corrects HAS players that select the wrong bitrate for video segments, HAS players closely follow the sharing policy in the proxy server. Even though the proxy server interferes with bitrate adaptation in the HAS players, this never compromised the continuity of the stream.

3.3 Bitrate Signaling and Traffic Control

In this section, the evaluations continue with the SDN-based Control Element. Although the proxy server satisfies the elementary requirements for managing HAS players in shared networks (see requirements R1-R4 in Chapter 2), the SDN implementation provides better privacy, scalability, and adaptability (see requirements R5-R7 in Chapter 2). Compared to the proxy server that features request rewriting to manage HAS players, the SDN-based Control Element uses bitrate signaling and traffic control. In the SDN implementation, HAS players exchange information about a video stream, as well as their state and device characteristics, with a component called the Service Manager. The Service Manager collects information about active HAS players and divides the available bandwidth. The Service Manager then communicates target bitrates to the players (i.e., the bitrate signaling mechanism) and instructs the Network Controller to provision the network element accordingly (i.e., the traffic control mechanism). In this section, bitrate signaling and traffic control are first evaluated separately, then combined.

3.3.1 Wi-Fi Streaming Testbed

The SDN-based Control Element is implemented using a Raspberry Pi 2B [111]. The Raspberry Pi acts as an OpenFlow-enabled switch, which is created using Open vSwitch (version 2.3.0) [84]. Open vSwitch is widely used software and offers a kernel-based switch implementation. This switch allows for relatively high switching performance compared to user-space-based implementations. The switching performance is not limited by the CPU power of the Raspberry Pi, but by the network interface. We confirmed this in test runs with Open vSwitch on the Raspberry Pi. The CPU load remains under 10%, even when fully loading the network interfaces. The traffic control mechanism is realized using Linux tc [63]. An additional interface is added to the switch to configure traffic control because OpenFlow 1.0 does not provide traffic control and queue configuration messages.

The Raspberry Pi also acts as the Wi-Fi access point. A Wi-Fi network is a realistic environment for experimenting because users often use Wi-Fi networks. Small

variations in throughput add an extra layer of difficulty for the HAS players and the Control Element. Nevertheless, the Control Element is generic. It also works on wired network connections, similar to the network used to evaluate the proxy server in Section 3.2 of this chapter.

A Wi-Fi access point is created using a USB Wi-Fi dongle (TP-Link TL-WDN4200 [83]) in combination with the hostapd software (version 2.5.0) [58]. We patched hostapd to become compatible with the Open vSwitch controlled network interfaces [68]. The Wi-Fi access point operates in the 5 GHz band on channel 44 (20 MHz wide channel). The Wi-Fi transmission power is 20 dBm, in compliance with Dutch regulations [113].

The Network Controller is implemented using POX (version 0.2.0) [105]. POX is a Python-based SDN controller platform that enables quick prototyping of network controllers. The southbound interface uses OpenFlow 1.0 to communicate with the Open vSwitch software switch. The Network Controller offers an interface to the Service Manager on the northbound interface. The Service Manager uses this interface to apply the HAS specific bandwidth sharing policy. The Network Controller and Service Manager may run on different hosts. However, in the experiments in this section, both processes run on the Raspberry Pi device.

The HAS player that is used is browser-based and implemented using the DASH.js player (version 2.2.0) [37]. We extend the reference player with the AssistanceController and AssistedAbrRule classes. The AssistanceController implements the communication between the HAS player and the Service Manager. The AssistedAbrRule is the actual adaptation algorithm that follows the target bitrate from the Service Manager when it can. Splitting the implementation into the AssistanceController and AssistedAbrRule classes allows us to maintain a communication channel with the Service Manager when using the built-in conventional (i.e., throughput-based) and BOLA [126] algorithms.

During the experiments, the HAS players run in the Chrome browser [51] on MacBook Pro and MacBook Air machines. The laptops have at least a dual-core Intel Core i5 processor and 4GB RAM. All laptops connect to the Wi-Fi access point from the Raspberry Pi. The Raspberry Pi stands in the middle of an office space with the laptops distributed around it.

3.3.2 Experiment Design

A scenario with four DASH players sharing a Wi-Fi network connection is evaluated. In this scenario, the Wi-Fi network forms the bottleneck link with a capacity of 25 Mbit/s. This scenario replicates a use-case of four family members sharing the household Wi-Fi, where each member watches video on their device. In experiments

with cross traffic, a fifth node joins to the Wi-Fi network. This node generates cross traffic using iperf [52].

The video nodes stream a ten-minute clip from the movie Sintel [123]. The nodes are manually started approximately one minute after each other. The video stream provides 12 different representations. The bitrates range from 300 Kbit/s at 240p to 10 Mbit/s at 1440p (2K), as listed in Table 3.2. The stream is formatted following the DASH format. The manifest is specified in the MPEG-DASH Live Profile [66] and is compatible with the DASH-IF guidelines [38].

Quality level	**1**	**2**	**3**	**4**	**5**	**6**
Bitrate (Kbit/s)	296	395	493	732	971	1459
Resolution	240p	240p	360p	360p	480p	480p

Quality level	**7**	**8**	**9**	**10**	**11**	**12**
Bitrate (Kbit/s)	1934	2878	3779	55544	7234	10563
Resolution	720p	720p	1080p	1080p	1440p	1440p

Table 3.2: Video bitrates and resolutions.

The focus of experiments in this section is on the effectiveness of the bitrate signaling and traffic control mechanisms from the SDN-based implementation of the Control Element. The individual mechanisms are evaluated first, then combined. As such, the evaluations cover three experiments:

- **Experiment 1: Impact of adaptation algorithms and target bitrate signaling.** The performance of the built-in adaptation algorithms (conventional and BOLA) and our target bitrate signaling mechanism are evaluated in Section 3.3.3. The algorithms are tested in an environment with and without background traffic.
- **Experiment 2: Impact of dynamic traffic control.** Section 3.3.4 covers experiments with four traffic control configurations. The first two configurations are using a dedicated HAS queue that puts all HAS players into a single queue, while using player queues puts each HAS player in its own queue. Both types of queues can work as rate limiting/bandwidth shaping queue (i.e., capping the throughput) or as minimum rate queue (i.e., providing a throughput guarantee without a cap). The focus in this experiment is on how traffic control helps the built-in adaptation algorithms. We evaluate both video bitrate and quality switches.

- **Experiment 3: Combination of target bitrate signaling and dynamic traffic control.** The experiments in Section 3.3.5 combine target bitrate signaling with traffic control. In this section, it is investigated if combining the two assistance mechanisms outperforms setups with only one of the two mechanisms enabled.

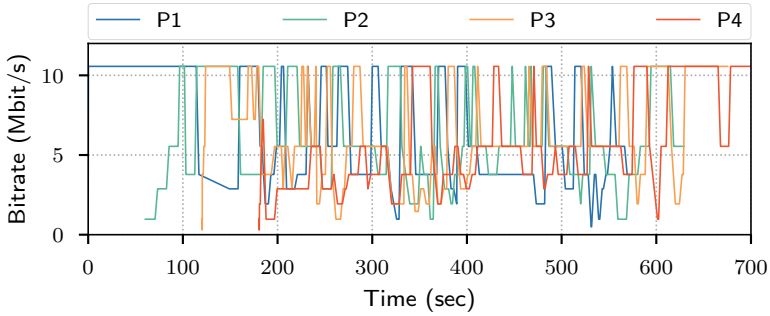
3.3.3 Target Bitrate Signaling

The conventional throughput-based algorithm is known to have difficulties in selecting a stable bitrate when HAS players share a network link [9, 7]. This problem is confirmed in experiments using the conventional adaptation algorithm in the DASH.js player. Even though the conventional algorithm is known for its instability, this algorithm is included in the evaluations because it is the default algorithm in the DASH.js player. Figure 3.6(a) shows the many quality switches when multiple players are active at the same time.

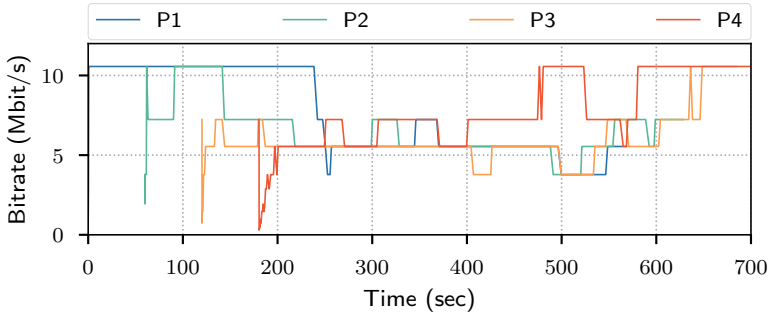
The DASH.js player also implements BOLA [126]. BOLA also uses the buffer level when selecting a video bitrate. Figure 3.6(b) shows that BOLA has significantly fewer quality switches compared to the conventional algorithm. Nevertheless, bandwidth sharing is unfair at times, for example in the period from $t = 400$ until $t = 550$. One player increases its video bitrate, while the other players have to adapt to lower bitrates. Although BOLA proves to outperform the conventional algorithm regarding quality switches, it is not immune to unfairness.

The experiment with target bitrate signaling shows to perform best regarding stability and unfairness, as shown in Figure 3.6(c). The HAS players strictly follow the target bitrate from the Service Manager. Figure 3.6(c) shows that target bitrate signaling limits quality switches to the moments where DASH players start and stop. It also shows that the response to a starting player is much quicker. The quick reaction means that active players immediately release bandwidth to the new player. Regarding video bitrate, target bitrate signaling has a lower bitrate compared to BOLA (4739 Kbit/s versus 6307 Kbit/s). The lower video bitrate is an effect of the sharing policy that strictly assigns the same bitrate to each player. However, if the Service Manager would assign a higher bitrate to one or two players, this effect would disappear.

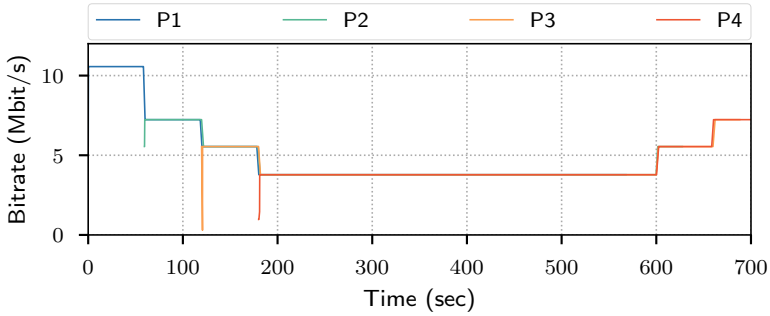
Adding target bitrate signaling from the Service Manager improves the stability of the streams. However, without traffic control, the Service Manager cannot guarantee a sufficient throughput for the HAS players. Figure 3.7 shows the distribution of video bitrates in the experiments without cross traffic. Figure 3.8 shows the experiments with cross-traffic (i.e., an iperf download), where the video bitrates drop significantly, also when target bitrate signaling is used.



(a) Conventional adaptation



(b) BOLA



(c) Assisted adaptation using target bitrate signaling

Figure 3.6: Video bitrates for four competing HAS players in a network without cross traffic.

In the experiments with cross traffic, the assisted algorithm in the HAS player does not adapt to the target bitrate. The bandwidth and buffer estimations detect that the achieved throughput is too low compared to the target bitrate. A fallback mode is triggered in the HAS players, resulting in a lower video bitrate. However, forcing the target bitrate would have led to more and longer interruptions in the video streams.

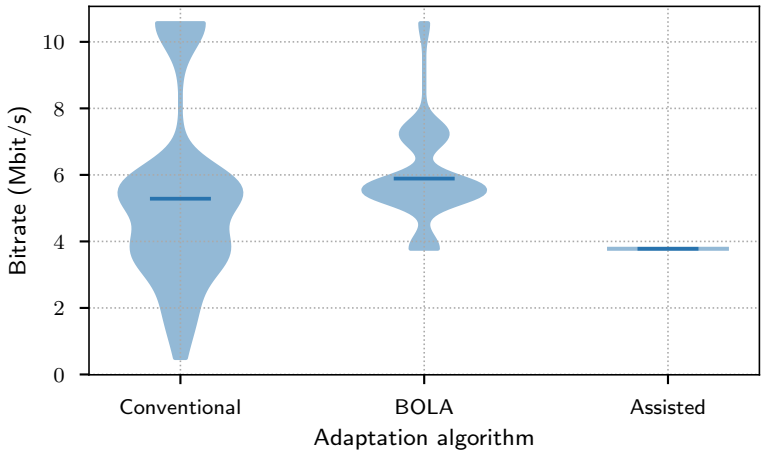


Figure 3.7: Distribution of video bitrates in experiments without cross traffic. Whiskers show the mean video bitrate.

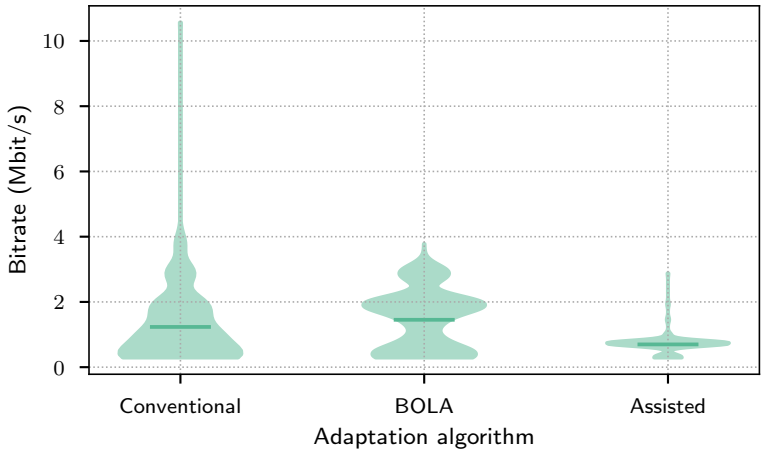


Figure 3.8: Distribution of video bitrates in experiments with cross traffic. Whiskers show the mean video bitrate.

Figures 3.9 and 3.10 show the buffer occupancy for each player, comparing BOLA and the assisted adaptation algorithm that uses target bitrate signaling from the Service Manager. The buffer occupancy in the HAS players are estimations based on the timing of segment requests in the traces. When the interval between two segment requests is less than the segment size, the buffer grows. When the interval

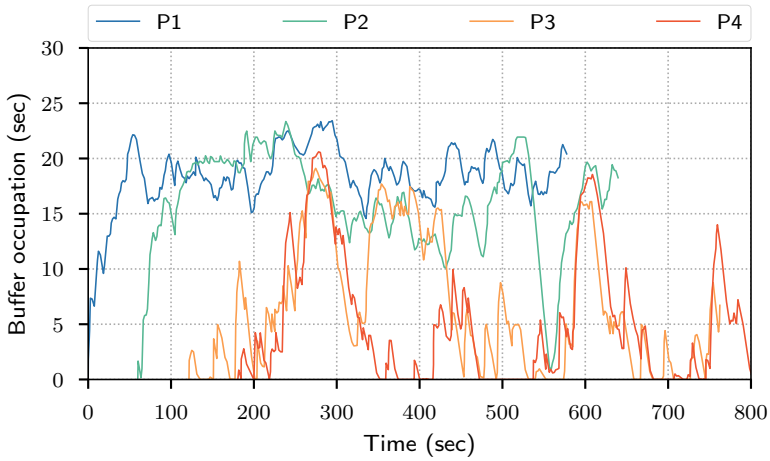


Figure 3.9: Buffer occupancy for the HAS players using BOLA. Experiment with cross traffic.

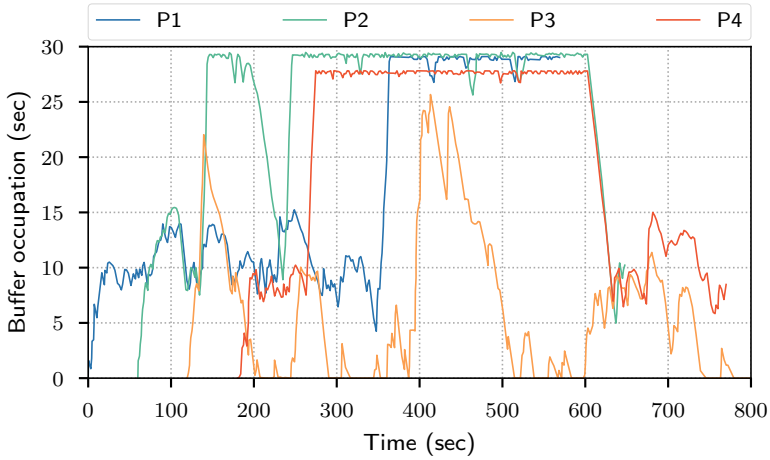


Figure 3.10: Buffer occupancy for the HAS players with assisted adaptation. Experiment with cross traffic.

between two segment requests is larger than the segment size, the buffer shrinks. Figure 3.9 shows that BOLA players 3-4, and assisted player 3 in Figure 3.10, have difficulties in maintaining a healthy buffer level. The buffers grow slow and are often empty. Figures 3.9 and 3.10 also reveal an inequality in buffer occupancy between the players. This shows that HAS players that enter a crowded network have difficulties getting enough bandwidth. Although target bitrate signaling from the Control Element improves buffer occupancy, Player 3 (P3 in Figure 3.10) often freezes due to an empty buffer.

Low buffer occupancy is the result of a competition for bandwidth between HAS players and the background TCP flow, where the background flow gets too much of the available bandwidth. Figure 3.11 shows that the performance of the background flow is only marginally affected by the HAS traffic. Given the 25 Mbit/s capacity of the network, each TCP flow (DASH.js uses HTTP/1.1 and keeps the connection to the server open, as such it counts as one TCP flow) should have a throughput of 5 Mbit/s. However, the background node uses a manifold of that. The HAS players stream far below this rate. The HAS players stream on average at 1157 Kbit/s ($\sigma = 1341$ Kbit/s) given the conventional algorithm, 1434 Kbit/s ($\sigma = 955$ Kbit/s) given BOLA, and 1305 Kbit/s ($\sigma = 1433$ Kbit/s) for assisted adaptation based on target signaling.

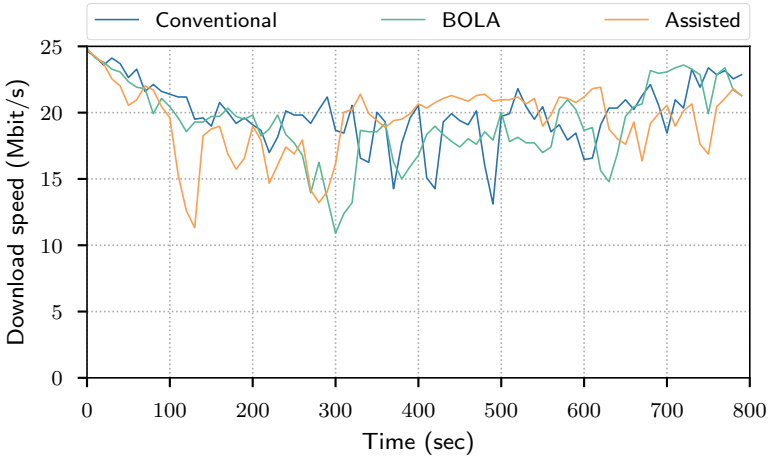


Figure 3.11: Throughput for cross traffic.

From the results, it can be concluded that the adaptation algorithm actively contributes to the streaming stability and the division of available bandwidth among players. However, adaptation algorithms show to strongly react to the download

speeds of the video segments. They do this directly by using throughput measurements or indirectly by observing the buffer level. If the download speeds are not sufficient, HAS players adapt to a lower video bitrate. When cross traffic is added to the network, download speeds are no longer sufficient. Then, improving client-side adaptation algorithms or using target bitrate signaling is not enough. In the next section, it is shown that traffic control solves these issues and realizes high-quality streaming.

3.3.4 Traffic Control

As shown in the Section 3.3.3, the problem with HAS players in a network with significant cross traffic is, that HAS players cannot reach sufficient download speeds. The ON-OFF download patterns of HAS [7] cause suboptimal TCP performance [45]. Segment downloads often start with the minimal TCP window. The TCP window size does only slowly increase, because there is already a high load in the network. TCP does not have enough time to obtain a fair share of the available bandwidth because the video segments are short (e.g., between two and ten seconds). Mitigating this issue can increase download speeds for HAS, for example by using traffic control.

In this part of the evaluation, the effectiveness of the traffic control mechanism is evaluated. In this experiment, the HAS players communicate with the Service Manager. However, instead of receiving target bitrates from the Server Manager, HAS players get adaptation assistance through traffic control. Internally, the HAS player uses the built-in adaptation algorithms (conventional algorithm and BOLA). The impact of traffic control on the streaming bitrate is evaluated. First, this evaluation looks into differences between using a single HAS queue and using player queues. Second, the impact of rate limiting queues versus minimum rate queues is evaluated. In any of the configurations, the Service Manager allocates an equal bitrate for each HAS player.

Single HAS Queue Versus Multiple Player Queues

Figures 3.12 and 3.13 show the bitrates for the conventional algorithm and BOLA when using traffic control. In these experiments, the Service Manager creates one queue per HAS player and one queue for cross traffic. Each queue has a rate limit and completely separates each flow from each other. The SDN-enabled switch creates queues using Linux tc [63] and hierarchical token buckets (HTBs) [42]. Given traffic control, the players with the conventional algorithm can stream at the preferred bitrate. However, they show a considerable number of quality switches. BOLA provides a stable stream as it closely matches the video bitrate to the queue size set by the Service Manager.

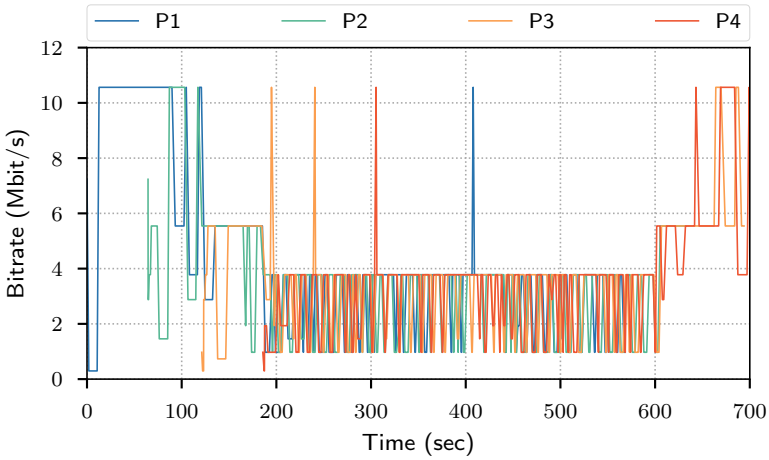


Figure 3.12: Video bitrates for the four HAS players with the conventional adaptation algorithm in a network with cross traffic and player queues with a rate limit.

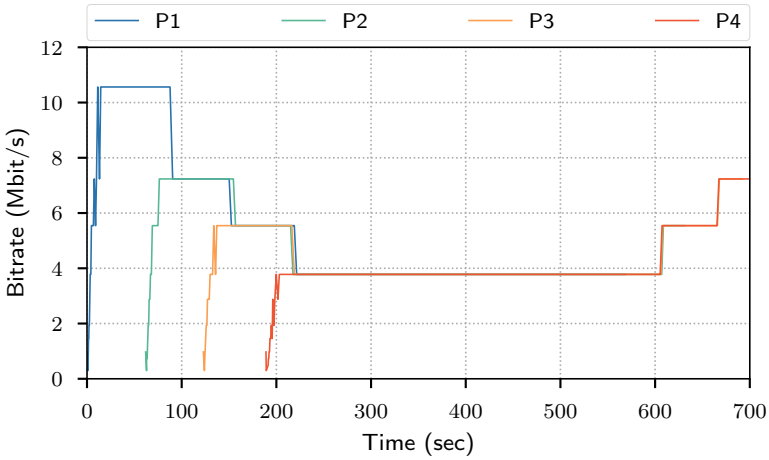


Figure 3.13: Video bitrates for the four HAS players using BOLA in a network with cross traffic and player queues with a rate limit.

Linux tc allows us to create large queueing configurations. However, other SDN switches may not offer this functionality and put a limitation on the number of queues. Therefore, a setup with a single queue for HAS traffic is evaluated as well. Figures 3.14 and 3.15 show the comparison of the video bitrates without using traffic control, comparing the use of a single HAS queue to using multiple queues. The video bitrate is slightly higher when using the HAS queue: 4768 kbit/s ($\sigma = 2442$ kbit/s) compared to 3578 kbit/s ($\sigma = 2507$ kbit/s) given the conventional algorithm, and 5191 kbit/s ($\sigma = 2329$ kbit/s) versus 4570 kbit/s ($\sigma = 1837$ kbit/s) with BOLA. Both adaptation algorithms increase the video bitrate when possible (i.e., when they either reach sufficient download speeds or have sufficient buffer occupancy). When combining the four players into a single queue, some players occasionally adapt to a higher bitrate compared to the target bitrate. These temporary upgrades in bitrate have an increase of the mean bitrate as a result. In contrast, the conventional algorithm shows a large number of requests for low bitrate video segments when using player queues. In this case, the players did not fully utilize the bandwidth in the queue, with a lower mean bitrate as a result.

For BOLA, in-stream variability is different when using a HAS queue compared to using player queues. Using a HAS queue, BOLA selects some of the segments in higher bitrates (relative to the target bitrate). However, other segments, also from other players, are requested at a lower bitrate for compensation. Regarding quality switches, using player queues increases performance. BOLA with player queues reduces the total number of switches by 52%, from 122 switches with the HAS queue to 59 switches with player queues. The reduction of quality switches of BOLA with players queues is also visible in Figure 3.13.

Regarding background traffic, there was no difference between the different queueing configurations. In fact, the queuing configuration does not affect the size of the background queue (i.e., the HAS queue has the same size as the player queues combined) Figure 3.16 shows how the throughput of cross traffic is adjusted and how network bandwidth is now available for HAS streaming. A starting player (the first players starts at $t = 0$) causes a decrease in throughput for the background flow, except when starting the fourth player at $t = 200$. The sudden increase in throughput is a direct effect of the sharing policy that divides bandwidth among players. The sum of the target bitrates cannot exceed the reserved bandwidth for HAS. In our policy, this limitation is 20 Mbit/s. Three players at 6653 Kbit/s use more bandwidth than four players at 4535 Kbit/s (19959 Kbit/s versus 18140 Kbit/s). This behavior is inevitable and is the result of the limited number of available bitrates in the HAS manifest. In this experiment, adding the fourth player causes a decrease of only one quality level.

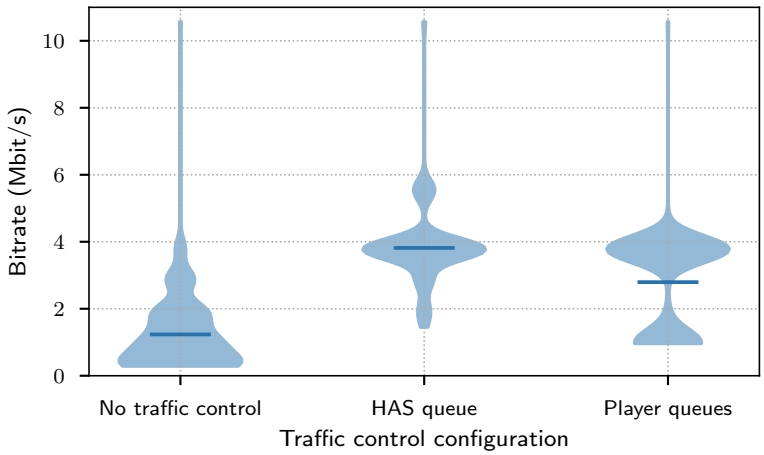


Figure 3.14: Distribution of video bitrates from HAS players using the conventional adaptation algorithm in a network with cross traffic and traffic control. The whiskers denote the mean video bitrate.

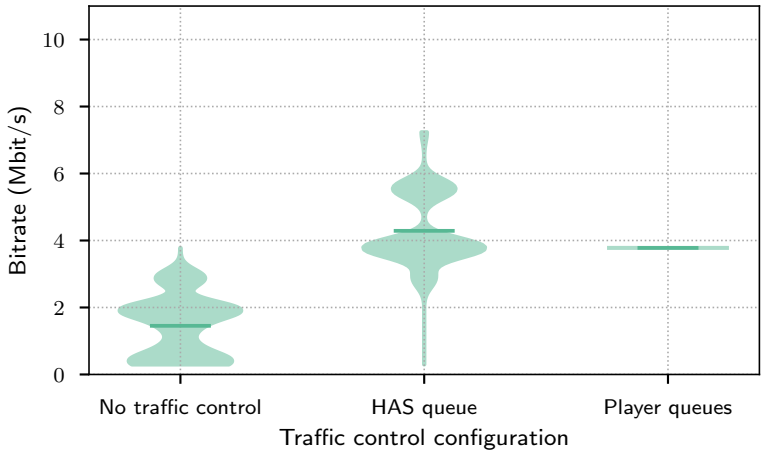


Figure 3.15: Distribution of video bitrates from HAS players using BOLA in a network with cross traffic and traffic control. The whiskers denote the mean video bitrate.

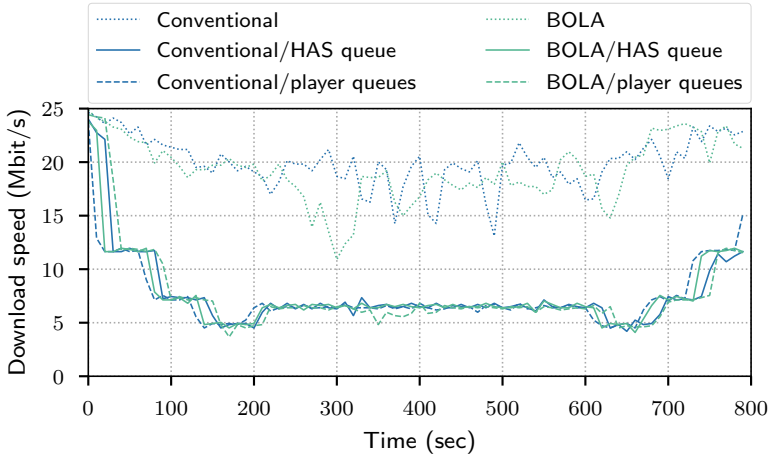


Figure 3.16: Throughput for background traffic when using a service queue and client queues.

Rate Limiting Queues Versus Minimum Rate Queues

This section compares the differences between rate limiting and minimum rate queues. Both types of queues guarantee bandwidth for HAS, given a target bitrate. For rate limiting queues the guaranteed bandwidth is the maximum bandwidth, where the throughput for HAS with minimum rate queues can exceed the target bitrate when there is bandwidth free. Figures 3.17 and 3.18 compare video bitrates given the two different queue types. The results show a higher number of quality switches with minimum rate queues. The distribution of requested bitrates is spread wider over the available bitrates. This effect is the strongest for the conventional algorithm, but can also be observed for BOLA. The results also show that the mean video bitrate using minimum rate queues is at least 1 Mbit/s higher compared to rate limiting queues. This result is expected and comes from the greediness of the HAS players (i.e., they will increase the video bitrate when possible). In our system, we implement a throughput guarantee for HAS traffic, but not for background traffic. Due to the implementation of hierarchical token buckets (HTB) in Linux tc, HAS players have an advantage over cross traffic.

Users typically prefer an improvement in video bitrate [120, 130]. However, we argue that the increase in bitrate when using minimum rate queues is unwanted behavior in this scenario. We assume that the Service Manager makes an optimal division of the available bandwidth (also including sufficient bandwidth for cross traffic), providing that dividing bandwidth is the Service Manager's primary objective. We observe that both the conventional algorithm and BOLA sometimes

decide on higher bitrate segments. Shorty increasing the bitrate causes more quality switches. These switches will eventually result in a lower QoE for the user because the higher video quality cannot be maintained for long.

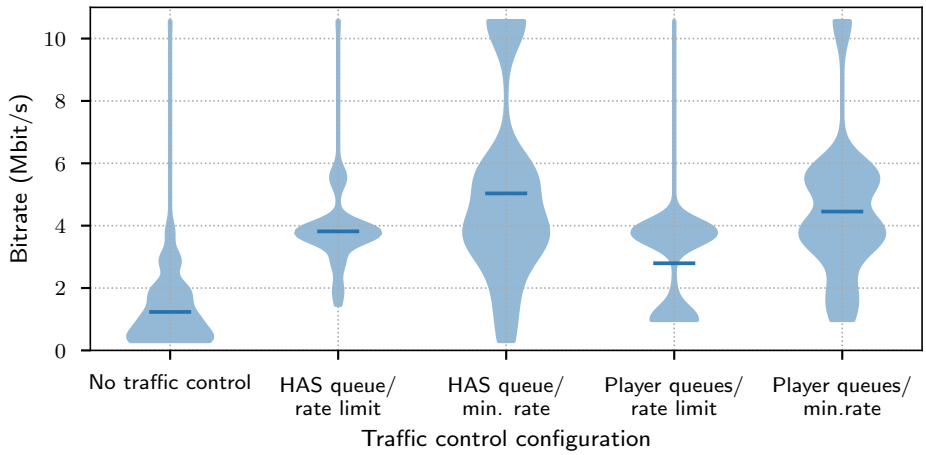


Figure 3.17: Distribution of video bitrates for players with the conventional adaptation algorithm in a network with cross traffic. Rate limiting queues versus minimum rate queues.

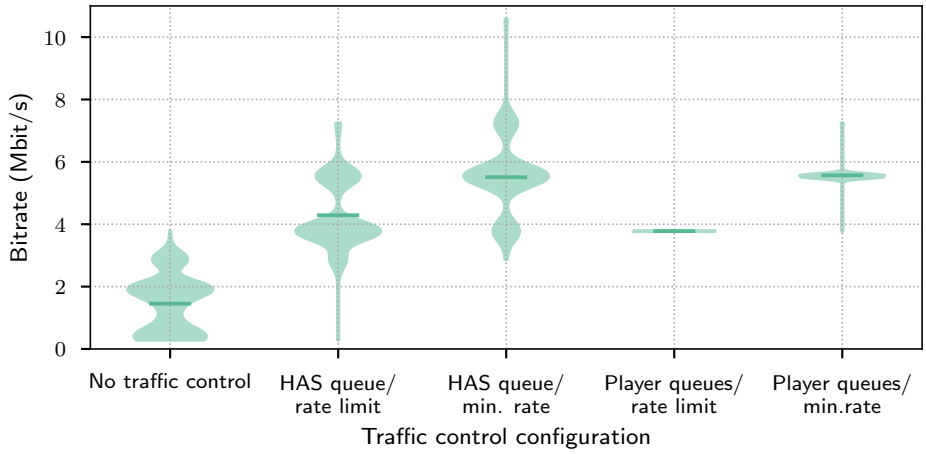


Figure 3.18: Distribution of video bitrates for players using BOLA in a network with background traffic. Rate limiting queues versus minimum rate queues.

In general, when using minimum rate queues, the number of quality switches is higher compared to when using rate limiting queues. Figures 3.19 and 3.20 display the total number of quality switches (i.e., adding up the number of switches from the four players) for the experiments without cross traffic and with cross traffic.

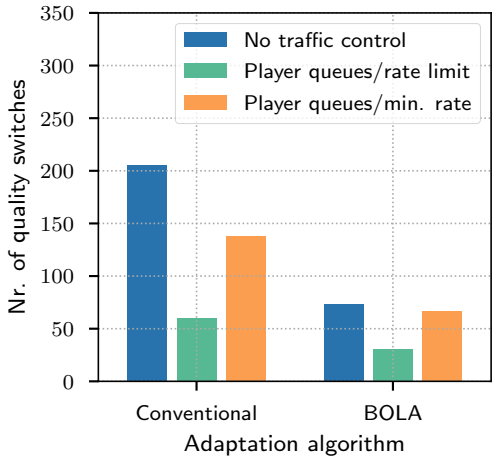


Figure 3.19: Comparison of the number of switches when using rate limiting queues and minimum rate queues in a network without cross traffic.

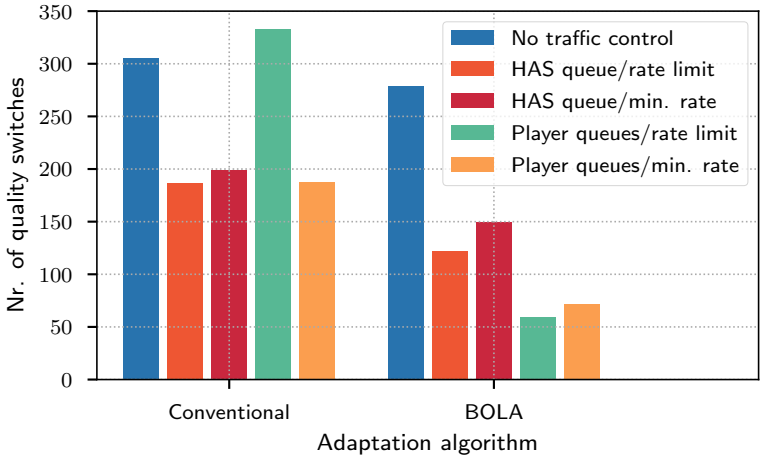


Figure 3.20: Comparison of the number of switches when using rate limiting queues and minimum rate queues in a network with cross traffic.

In the experiment without background traffic, we only evaluate player queues. A single HAS queue would not affect streaming performance because the size of the queue would then be equal to the capacity of the network. Figure 3.19 shows that rate limiting queues reduce quality switches the best. For the conventional algorithm, rate limiting queues reduce the number of switches by 71%, from 205 to 60. For BOLA we observe a reduction of 58%, from 73 to 31 switches. The impact of minimum rate queues is less compared to rate limiting queues. When using minimum rate queues, the number of quality switches is 32% lower given the conventional algorithm, and 8% lower given BOLA.

In the experiments with cross traffic, we observe a similar effect. However, the differences between rate limiting and minimum rate queues are smaller. The experiment with the conventional algorithm in combination with rate limiting client queues shows to be an exception. In this setting, HAS players often alternate between the target bitrate and a lower bitrate. The conventional adaptation algorithm is aggressive and reacts quickly to small changes in throughput. This results in over 330 quality fluctuations. Overall, BOLA in combination with player queues performs best regarding quality switches.

Regarding the throughput of cross traffic, minimum rate queues cause a relatively low throughput. The cross traffic queue does not have a minimum rate guarantee and thus gives HAS players an advantage over cross traffic. Figure 3.21 shows the decline in throughput when HAS players get active. Comparing download speeds of cross traffic for both rate limiting queues and minimum rate queues, a

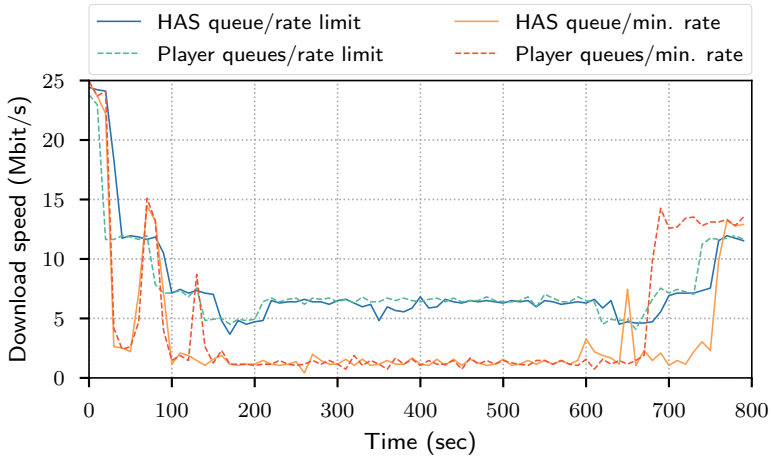


Figure 3.21: Throughput for cross traffic when using BOLA with rate limiting and minimum rate queues.

decrease of around 80% is observed. For example, the average download speed drops from 6506 Kbit/s ($\sigma = 2050$ Kbit/s) to 1250 Kbit/s ($\sigma = 1952$ Kbit/s) given BOLA-based players with a single HAS queue (measured in the interval where all four HAS players are active, $t \in [200, 600]$).

Based on the results in this part of the evaluation, it can be concluded that traffic control is essential for realizing high-quality HAS streaming. Given the optimal target bitrate selected by the Service Manager, player queues with a rate limit provide the best performance. Nevertheless, using a single service queue is still useful, and it protects HAS players against cross traffic. Using minimum rate queues is not optimal. The greediness of the adaptation algorithms comes at the cost of instability and poor throughput for background traffic.

3.3.5 Target Bitrate Signaling with Traffic Control

Based on the experiments in Sections 3.3.3 and 3.3.4, we conclude that target bitrate signaling and traffic control both improve streaming performance. However, the two mechanisms improve streaming performance in different ways. Each mechanism has its advantages, but also its drawbacks. In this part of the evaluation, we evaluate streaming performance when combining target bitrate signaling with traffic control. Bitrate signaling can increase confidence to HAS players when selecting bitrates, while traffic control can ensure sufficient bandwidth. In this combination, the two mechanisms optimally work together. By combining the two mechanisms, HAS players can quickly adapt to the target bitrate and thus react faster to other HAS players starting and stopping. Figures 3.22 and 3.23 show a comparison of the video bitrate for BOLA-based HAS players and players with the assisted adaptation algorithm. In this experiment, we use player queues with a rate limit. Both players with BOLA and the assisted adaptation algorithm stream closely to the target bitrate selected by the Service Manager. BOLA mostly uses buffer occupancy to determine the target bitrate where our assisted adaptation algorithm gets the bitrate directly from the Service Manager. Comparing the time that a player needs before streaming at the target bitrate, the assisted adaptation mechanism adapts 51% faster than BOLA. We observed an average reduction from 16.75 segments (33.5 seconds) for BOLA to 8.25 segments (16.5 seconds) when using target bitrate signaling.

Combining target bitrate signaling with other queuing configurations results in similar performance with respect to video bitrate and quality switches. However, Figure 3.24 shows that different queueing configurations facilitate different buffer occupancy. The buffers fill slower when using queues with a rate limitation. Rate limiting queues only offer little room for the HAS players to grow their buffer, mainly because the throughput in a queue cannot exceed the target bitrate. Queues

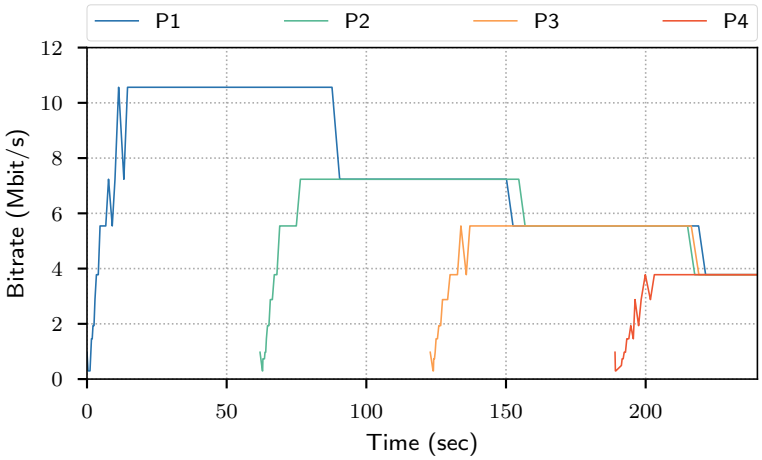


Figure 3.22: Video bitrates for HAS players with the BOLA adaptation algorithm in a network with background traffic and player rate limiting queues.

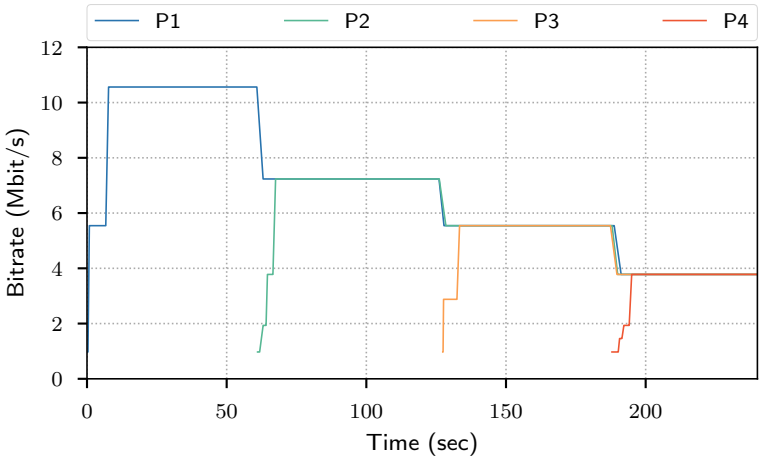
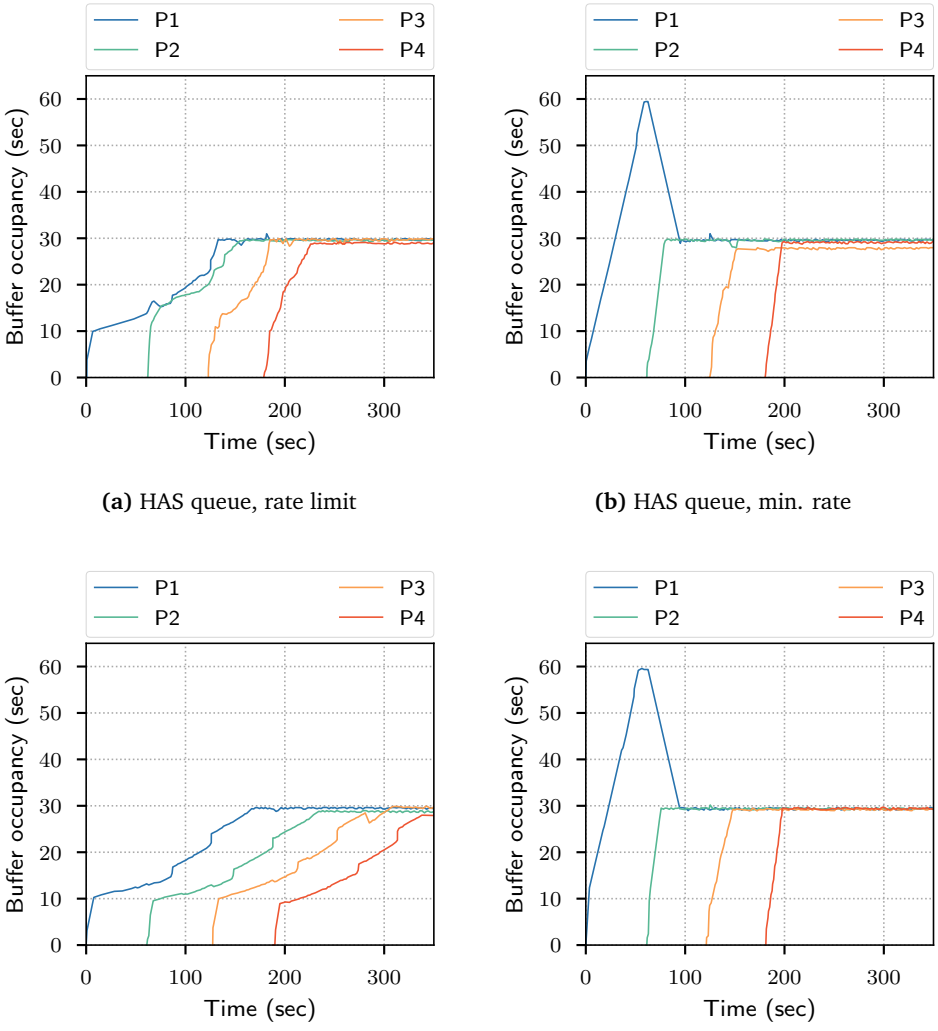


Figure 3.23: Video bitrates for HAS players with the assisted adaptation algorithm in a network with background traffic and player rate limiting queues.



(a) HAS queue, rate limit

(b) HAS queue, min. rate

(c) Player queues, rate limit

(d) Player queues, min. rate

Figure 3.24: Buffer levels when using assisted adaptation with different queuing configurations.

with a minimum rate guarantee provide a solution for this problem and allow HAS players to increase their download speeds temporarily. Minimum rate queues shorten the time that HAS players need to fill a buffer of 30 seconds. The HAS queue with a minimum rate reduces the average buffering time from 91 to 24 seconds compared to rate limiting queues. For player queues, this reduction is from 167 to 21 seconds on average. The peak in buffer occupancy for the first HAS player in Figures 3.24(b) and 3.24(d) is the result of a bug in the scheduler of the DASH.js player that we use in the experiments. Although we configured a maximum buffer of 30 seconds, the DASH.js player sometimes ignores this limit. Nevertheless, this bug does not affect the bitrate for the other players.

The problem with client-side adaptation algorithms, such as the conventional algorithm and BOLA in the DASH.js player, is that they are greedy and try to increase the video bitrate at all times. This behavior causes problems with the available bandwidth for cross traffic when using minimum rate queues. Target bitrate signaling not only prevents HAS players from selecting a too low bitrate, but it also restricts players from picking bitrates that are higher than the target bitrate. This approach has the advantage that it does less harm to cross traffic. Figure 3.25 shows that assisted streaming has a smaller impact on the cross traffic throughput compared to using client-side adaptation algorithms. Buffering HAS players cause a temporary reduction of download speed for cross traffic, but minimum rate queues overall induce an increase in download speed.

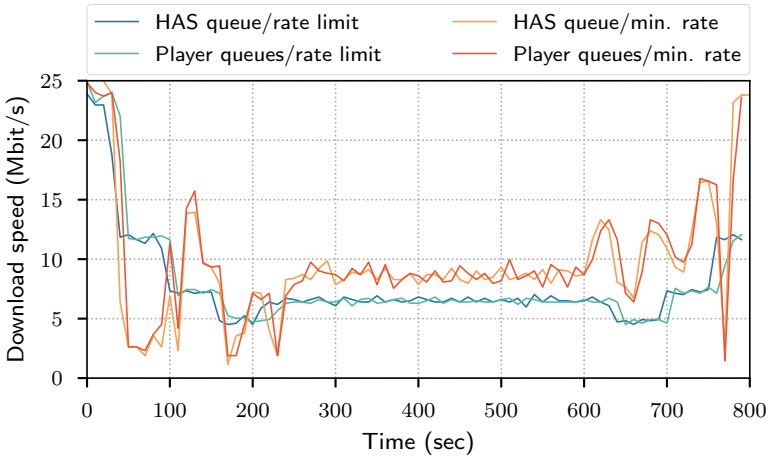


Figure 3.25: Download speeds for cross traffic when using target bitrate signaling with different queuing configurations.

Based on these results, it can be concluded that combining target bitrate signaling with traffic control produces a high-quality video stream with excellent stability. Both mechanisms contribute to the streaming performance. However, each mechanism contributes differently and targets a different optimization objective. Target bitrate signaling enables HAS players to adapt the video bitrate quickly when a new player starts, or an active player stops. It supports players in fairly dividing the available bandwidth. With target bitrate signaling, players do not have to rely on (inaccurate) bandwidth estimations. As such, bitrate signaling provides stability and fairness. Traffic control ensures sufficient bandwidth for both HAS and cross traffic, thus preventing video freezes and increasing the video bitrate. The two mechanisms work together for optimal delivery of the video streams. Compared to using bitrate signaling or traffic control individually, combining the mechanisms provides better robustness.

3.4 Wi-Fi Network Quality

In the experiments in the previous section, all nodes are located close to the Wi-Fi access point. This means that the nodes have optimal Wi-Fi reception. For none of the nodes, the distance to the access point is a cause for lowered network performance. However, in many Wi-Fi deployments, it is common that there are nodes further away from the access point. For such nodes, it would not be possible to use the full capacity of the Wi-Fi network, because Wi-Fi lowers the transmission speed when radio conditions decrease. A node at the edge of reception can potentially stream still stream at high bitrates, but only when this node is the only active node in the network. When there is other traffic in the network, the network performance of this node will decrease, and a HAS player may have difficulties maintaining a high quality.

In the following experiment, one node is located at a distance from the access point such that it can stream at the highest bitrates, but it does not have any remaining capacity left. In this experiment, we use a ten-minute video clip from the movie Sintel [123] that is available in 14 representations, from 296 Kbit/s (240p) up to 18 Mbit/s (4K). The streaming performance for this node is shown in Figure 3.26 on the left. For better visibility, the figure uses the representation index instead of the video bitrate. The lowest video quality has index 1, the highest video quality index 14. Figure 3.26 shows that when the node is alone, its HAS player selects the high-quality segments. When the other HAS players are active (Figure 3.26 center), the quality of the HAS stream of the node further away is significantly lower. Adding traffic control in the network cannot restore the streaming performance of this player. The video quality is degraded further in the scenarios with cross traffic. In these scenarios, the video quality degrades to the lowest levels.

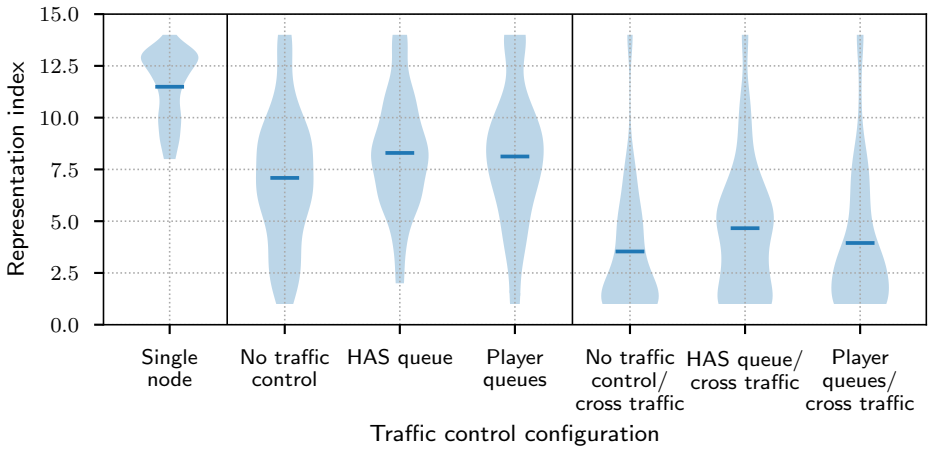


Figure 3.26: Distribution of video quality levels for the node further away from the Wi-Fi access point given QoS support in the network. One node (left) vs. all video clients active (center) vs. all video clients with background traffic (right).

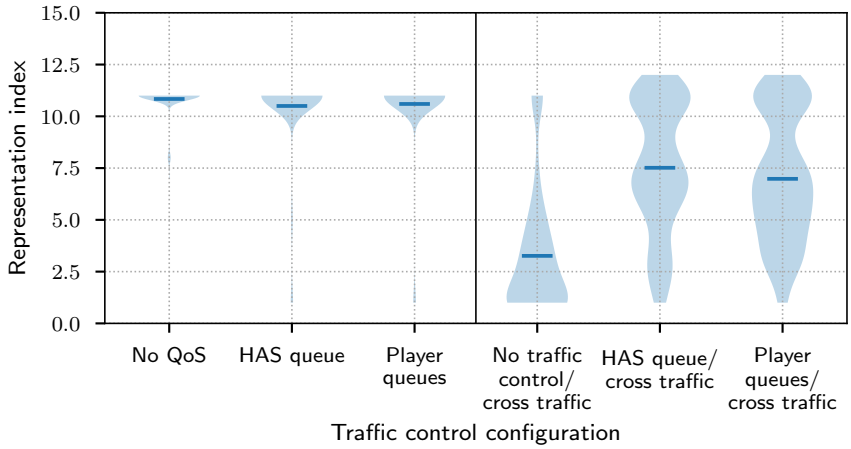


Figure 3.27: Distribution of video quality levels for the node further away from the access point with bitrate signaling assistance enabled. A network without background traffic (left) vs. a network with background traffic (right).

The results from this experiment show again that in a Wi-Fi network traffic for HAS players should have enough room to get through. Providing bandwidth is partially accomplished via bitrate signaling. Signaling is effective in two ways: On the one hand, it tells the player far away from the Wi-Fi access point that it does not have to select the lower bitrates and that inconstancies in bandwidth are only temporary. On the other hand, it prevents players close to the Wi-Fi access point from (selfishly) increasing the video capacity and putting a too large demand on the network, negatively affecting the node with poor Wi-Fi radio conditions.

Figure 3.27 on the left shows that without cross traffic, the node further away from the access point streams at the same quality level as the nodes that are close to the access point. With cross traffic, the network load is too high to reach the same quality. However, compared to the setting without assistance, the video quality almost doubles, and the number of quality switches is lower, as shown in Table 3.3.

	Mean quality level (σ)	**Nr. switches**
Without bitrate signaling:		
No traffic control	2.54 (3.18)	89
HAS queue	3.66 (3.29)	85
Player queues	2.95 (3.45)	82
With bitrate signaling:		
No traffic control	2.26 (3.02)	76
HAS queue	6.51 (3.09)	53
Player queues	5.98 (3.05)	43

Table 3.3: Mean quality and number of quality switches for the node further away from the access point in a network with background traffic.

The video quality for the far-away node could potentially improve by lowering the throughput of cross traffic with traffic control. However, it is up to the users or the network administrator to decide whether further limiting cross traffic weighs up to the increase in video quality for a single user.

3.5 Heterogeneous Devices

In the experiments in Sections 3.3 and 3.4, the SDN-based Control Element executes a sharing policy that assigns all players the same bitrate. As such, the Control Element implements a bitrate fairness policy. Bitrate fairness works when each player is similar (i.e., each playback device has the same properties) or all users have the same priorities in the network. However, given that HAS-based streaming

is available on a variety of devices, from smartphones with small screens to Internet-enabled TVs with large high-resolution screens, the Control Element should take device characteristics and user priority into account when dividing the available network bandwidth. In fact, the ability to treat different devices differently is a major advantage of using the Control Element. Because the Control Element has the overview of active HAS players, it can make an optimal bandwidth division. Where individual HAS players may try to increase their video bitrate at the cost of players with larger screens or higher priority, the Control Element prevents this behavior.

In this section, the Control Element is used with a sharing policy that is aware of device characteristics. Using the Wi-Fi streaming testbed that is described in Section 3.3.1, the players will pretend to be two smartphones, a laptop, and a TV. In the following experiments, HAS players communicate their device type to the Service Manager. We configured two players to report themselves as a smartphone (small screen), one as a laptop, and one as a TV. The Service Manager assigns representations up to 360p for the smartphones, up to 1080p for the laptop, and up to 1440p for the TV. The players start in the order: smartphone, smartphone, laptop, TV.

Table 3.4 lists the mean bitrates for the four experiments using the device aware sharing policy. In the first experiment, the players are allocated to a single HAS queue and a maximum bitrate is manually configured in the players. In the second and third experiments, the maximum bitrate is enforced using player queues with a maximum rate. The conventional adaptation algorithm and BOLA implementations in the DASH.js player handle video bitrate adaptation in these experiments. The last experiment combines target bitrate signaling and player queues with a rate limit.

The results show that in all four experiments the bitrates of the TV are higher than the bitrates of the laptop, and the bitrates of the laptop are higher than those of the smartphones. However, the mean bitrate of the TV is too low when manually configuring a maximum bitrate in the players. Frequent quality switches between the target bitrate and a lower bitrate cause the mean video bitrate to be lower. Using the conventional algorithm with player queues shows a similar effect. BOLA and assisted adaptation show mean bitrates closest to the target bitrates.

Table 3.5 lists the number of quality switches for each player. The number of switches shows similar effects to previous results in Section 3.3. BOLA produces fewer switches compared to the conventional adaptation algorithm. When using target bitrate signaling, players adapt quicker to the target bitrate and show the least quality switches. The results show that the Control Element is effective in

Adaptation algorithm	Phone 1 Kbit/s (σ)	Phone 2 Kbit/s (σ)	Laptop Kbit/s (σ)	TV Kbit/s (σ)
Conventional algorithm with fixed maximum	711 (89)	696 (121)	3401 (995)	8365 (3089)
Conventional algorithm with traffic control	717 (122)	712 (177)	2953 (1338)	10101 (1813)
BOLA with traffic control	725 (51)	880 (140)	3676 (579)	10077 (1916)
Bitrate signaling with traffic control	726 (55)	728 (58)	3707 (425)	10391 (1216)
Target bitrate	**732**	**732**	**3779**	**10563**

Table 3.4: Average video bitrates per device when using a device-aware policy.

Adaptation algorithm	Phone 1 # switches	Phone 2 # switches	Laptop # switches	TV # switches
Conventional algorithm with fixed maximum	4	7	12	46
Conventional algorithm with traffic control	16	31	79	8
BOLA with traffic control	5	11	7	12
Target bitrate signaling with traffic control	2	2	3	3

Table 3.5: Number of quality switches per device when using a device-aware policy.

enforcing policies that go beyond equally sharing the network bandwidth. The advantage of assisted adaptation is that sharing policies only have to be configured in a single place (i.e., the Control Element). The Control Element can potentially change sharing policies, based on which clients are active.

In the current implementation, HAS players send a special message informing the Control Element about the device form factor. This implementation thus does not require the Control Element to know which devices are going to be active and what their characteristics are. Having clients report their characteristics provides little means against cheating players. For example, players could report a higher resolution screen and benefit from relatively higher quality video.

3.6 Scalability

In the experiments in Sections 3.2, 3.3, and 3.4, we use a small number of players. In related work on HAS performance problems and network-based HAS optimization solutions, the number of players is between 1 and 150 players. An overview of the number of players in related work is given in Table 3.6. As such, experiments mostly represent scenarios like homes, small offices or small hotels. In this section, we assess HAS performance in larger networks. It is investigated if the problems of low video bitrates, quality switches, and unfairness also exist for HAS in larger networks. Furthermore, it is evaluated if the target bitrate signaling and traffic control mechanisms are effective in a network with up to 600 concurrent HAS players.

Topic	**Reference**	**Max. # players**
HAS performance problems	Huang et al., 2012 [60]	1
	Esteban et al, 2012 [45]	1, or more
	Akhshabi et al., 2011 [9]	2
	Akhshabi et al., 2012 [7]	12
Network assisted HAS	Houdaille and Gouache, 2012 [59]	2
	Georgopoulos et al., 2013 [50]	3
	Kleinrouweler et al., 2016 [74]	4
	Kleinrouweler et al., 2015 [73]	17
	Bouten et al., 2012 [24]	50
	Bentaleb et al., 2016 [18]	50
	Petrangeli et al., 2015 [102]	90
	Cofano et al., 2016 [32]	150

Table 3.6: Overview of the number of HAS players used in related work.

3.6.1 Large Scale Streaming Testbed

The Control Elements in the scalability experiments is based on the SDN implementation described in Chapter 2. However, we adjusted the implementation for better performance and the ability to handle large numbers of clients. The Control Element in this section consists of three elements: the network bridge, network controller, and service manager.

The Network Bridge is a PC with two gigabit ethernet interfaces. The interfaces are bridged together and perform packet forwarding and traffic control. Traffic control is implemented using Linux tc [63] and hierarchical token buckets [42]. The Network Controller is software that configures traffic control on the Network

Bridge. It provides a programming interface that the Service Manager uses to reserve bandwidth for HAS players. By default, the Network Controller configures one traffic queue, but it can set up additional queues dedicated to HAS traffic. The Network Controller can create any number of HAS queues. Active HAS flows will be automatically balanced over the different queues. The bitrates of the HAS flows determine the minimum throughput for traffic in that queue (i.e., minimum rate queues). One queue is designated for control messages between the HAS players and the Service Manager. Prompt delivery of control messages is essential the Control Element to be effective. The queue is over-provisioned at a rate of 50 Mbit/s, to make sure that control messages are delivered without delay. HAS players can use bandwidth in the queue when it is not used for control messages.

The Service Manager is the entity that assists the HAS players. A schematic overview of the interactions involving the Service Manager is shown in Figure 3.28. HAS players request assistance from the Control Element by connecting to the Service Manager via a WebSocket and report the representations from the HAS manifest (step 1). The Service Manager takes the reported representations and divides the available bandwidth among the players. For the experiments in this section, we assume that all devices have the same form factor and priority. Therefore, the available bandwidth is equally divided among HAS players. The Service Manager communicates the target representations to the Network Controller (step 2) which will configure traffic control on the Network Bridge (step 3). The target representations are also sent to the HAS players (step 4).

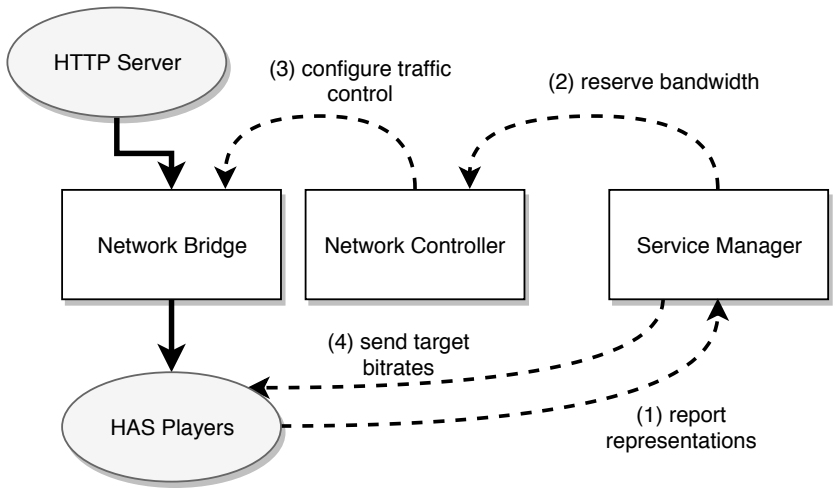


Figure 3.28: Control Element components and interactions. The solid lines indicate data flow, the dashed lines indicate control messages.

The Service Manager is implemented in JavaScript within Node.js [97]. Node.js can handle a large number of simultaneous WebSocket connections. We restricted the number of updates from Service Manager to HAS players and Network Controller to one update per two seconds. Update messages are typically triggered by starting and stopping HAS players. In large networks, the number of starting and stopping players is large, resulting in a high number of update messages. Because HAS players can only change the video quality when requesting new segments, sending update messages faster than the segment duration thus has no effect.

For the HAS players, 30 Raspberry Pi 3 Model B [112] are used. To be able to start multiple HAS players on a single Raspberry Pi, we created a custom headless version of the DASH.js player [37] using Node.js. Video decoding and rendering consume most CPU resources. Decoding and rendering are not a problem when using a single player on a Raspberry Pi. However, we want to investigate network transmission for many HAS players. Therefore, we use a headless player and multiple players per Raspberry Pi. A timer simulates playback of a video segment with the length of the video segment. Disabling decoding and rendering does not change the networking behavior of the player.

The headless player gives comparable results to the DASH.js player. To further reduce memory usage, the player maintains a buffer of small fake video segments. After downloading a video segment, it is discarded and only metadata is stored (i.e., the size and duration of the video segment) in the buffer. With this approach, we are able to run up to 20 DASH players on a single Raspberry Pi while keeping CPU usage below 60% and memory usage below 720 MB (out of 1 GB available), ensuring that the hardware is not the limiting factor in our experiments. Running more players at the same time showed increasingly unstable results. Given the testbed with 30 Raspberry Pis and 20 DASH player instances, up to 600 simultaneously active HAS players can be emulated.

The headless player implements three adaptation algorithms: conventional (i.e., throughput-based), BOLA, and assisted adaptation. The conventional algorithm and BOLA (i.e., the adaptation- and abandonment rules) are implemented as in the DASH.js (v2.2.0) player [37]. For assisted adaptation, we use a custom rule. The rule works as follows: HAS players receive quality recommendations from the Service Manager. If the buffer level is higher than 10 seconds and BOLA indicates a video quality equal to or higher than the signaled quality, the signaled quality is used. Otherwise, the HAS player selects the quality as determined by BOLA. Assisted adaptation has the effect that in most cases the quality recommendation from the Service Manager is adopted by the HAS players. In case of a starting player or unexpected lower network performance, the assisted adaptation rule relies on BOLA to provide a better quality recommendation than the Service Manager.

Algorithm 1 lists the pseudo-code for the assisted adaptation rule. The adaptation rule executes before each segment download.

Algorithm 1: Assisted adaptation rule.

Inputs : $b \leftarrow$ current buffer level
$q_{ce} \leftarrow$ target quality from Control Element
$q_{bola} \leftarrow$ quality determined by BOLA
followCE $\leftarrow$ *true* if previous segment got quality level from Control Element, *false* else
Outputs : $q_{segment} \leftarrow$ quality for next segment
followCE $\leftarrow$ *true* if next segment gets quality level from Control Element, *false* else

```
1  q_segment = min(q_ce, q_bola)
2  if b ≥ 10 seconds then
3      if q_bola ≥ q_ce ∨ followCE then
4          q_segment = q_ce
5          followCE = true
6      else
7          followCE = false
8  else
9      followCE = false
10 return q_segment, followCE
```

The conventional algorithm, BOLA, and assisted adaptation are evaluated in a wired streaming testbed. The testbed consists of three network switches, an HTTP server, a PC that functions as our Control Element, and 30 Raspberry Pis hosting HAS players. Figure 3.29 shows the network diagram for our experiments. The network switches, Control Element, and HTTP server are connected via 1 Gbit/s Ethernet. The Raspberry Pis are connected via 100 Mbit/s Ethernet, because of Raspberry Pi network interface limitations. The secondary switches connect to five Raspberry Pis each, to prevent internal bottlenecks (i.e., five Raspberry Pis together cannot download more than 500 Mbit/s on a shared 1 Gbit/s link). The primary switch is configured as a plain Layer-2 switch (not prioritizing or limiting the network performance of the ten Raspberry Pis connected to the secondary switches). Hard- and software settings are listed in Table 3.7. An impression of the Raspberry Pi-based streaming testbed is shown in Figure 3.30.

The HAS players stream a ten-minute clip from the movie Sintel [123], the same video clip that has been used in the previous sections. The bitrates range from 300 Kbit/s at 240p to 10 Mbit/s at 1440p (2K), as listed in Table 3.2. The stream is formatted following the DASH format. It implements the MPEG-DASH Live Profile [66] and is compatible with the guidelines from the DASH-IF [38].

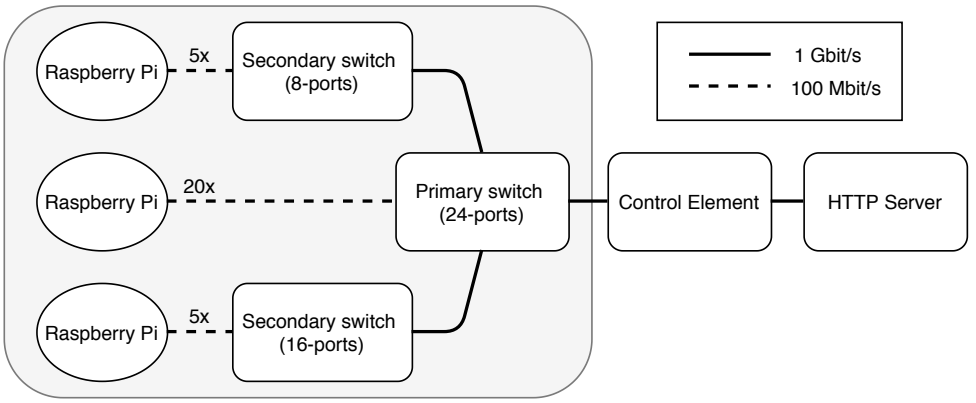


Figure 3.29: Network diagram showing the experimental setup.

Device	Description	#
Raspberry Pi 3 model B, Raspbian Jessie Lite, nodejs 6.9.4	HAS player host	30
HP ProCurve 25106-24 (J9279A), 24 ports, firmware Y.11.49, configured as L2 switch	Main switch	1
NETGEAR ProSafe GS116, 16 ports	Secondary switch	1
NETGEAR ProSafe GS108, 8 ports	Secondary switch	1
PC with Intel Core i5-5250U (quad core) 1.6 GHz, 8GB RAM, Debian Linux 8, nodejs 6.9.4	Control Element and traffic shaper	1
PC with Intel Core i3-M350 (quad-core), 2.27 GHz, 4GB RAM, Debian Linux 8, nginx 1.10.1	HTTP server	1

Table 3.7: Overview of testbed hardware and software.

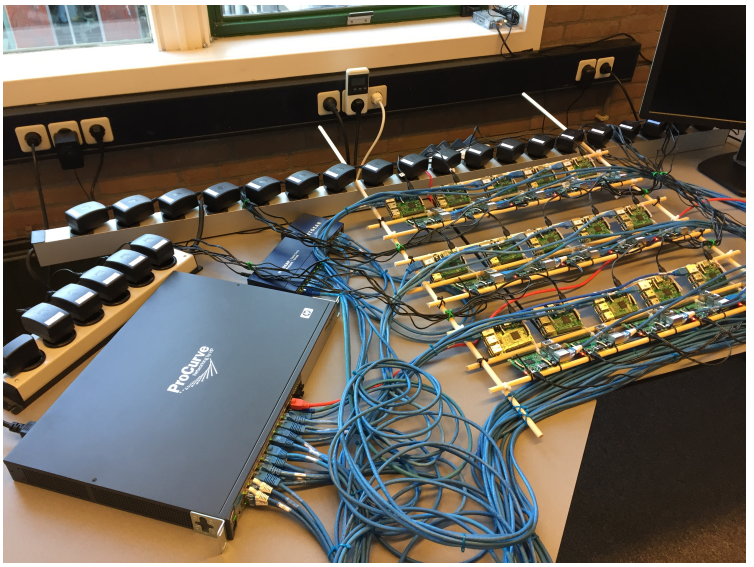


Figure 3.30: Raspberry Pi based streaming testbed.

3.6.2 Performance Evaluation

The evaluation of the adaptation algorithms and our Control Element is split into two experiments:

- **Experiment 1: Parallel start-up.** In this experiment we start $n \in \{60, 240, 420, 600\}$ HAS players at the same time. This scenario resembles the start of a popular stream (e.g., a video going viral or an important live event such as a big soccer match). We perform 20 runs per setting to account for the small variations between the runs. Each HAS player streams a video of 180 seconds.
- **Experiment 2: Poisson process start-up.** HAS players are started following a Poisson process with arrival rates $\lambda \in \{0.9, 1.9, 2.9\}$ and a video duration of 180 seconds. This will give on average 150, 330, and 510 simultaneously active HAS players. The actual number of HAS players varies over time. The maximum number of active HAS players cannot exceed 600. Due to the use of a Poisson process, we perform one run of two hours for each arrival rate λ. After two hours the mean number of players is converged with a variation of less than 0.05 players per 10 seconds.

The evaluation in this section focusses on video freezes, video bitrate, and quality switches. A freeze is an interruption in playback and occurs when the buffer in a HAS player is empty. In this analysis, we count the number of freezes per player.

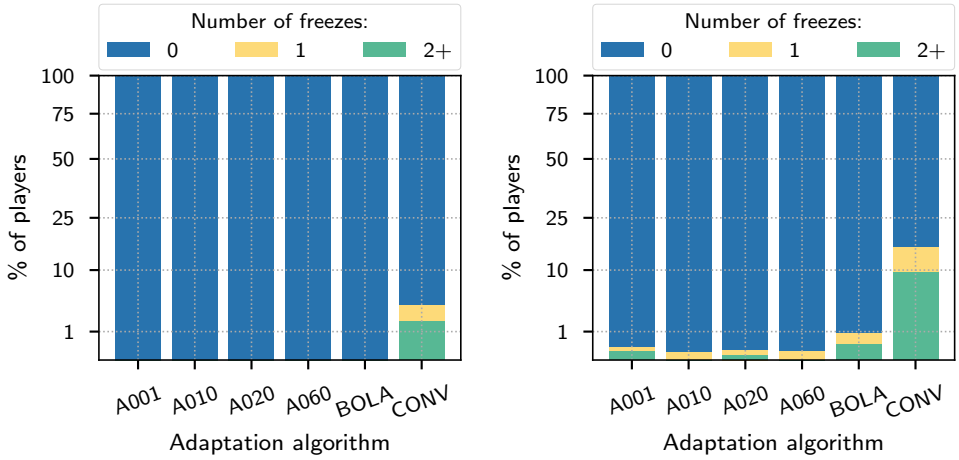
Freezes are reported as percentages of players that experienced none, one, and two or more freezes while streaming the video. This evaluation does not consider the duration of a freeze. The video bitrate is determined for each player as the mean of the bitrates of the video segments for that player. The standard deviations (denoted by σ) indicate the variations of mean bitrates between the players.

Each time that two consecutive segments have a different bitrate is considered a switch. The number of switches is counted independent from the size of the switch (meaning that differences of more than one quality level according to the HAS manifest are still counted as one switch). The following acronyms are used: *"CONV"* for the conventional adaptation algorithm, *"BOLA"* for BOLA, and *"Axxx"* for assisted adaptation, where *xxx* stands for the number of traffic shaping queues.

Parallel Start-Up Results

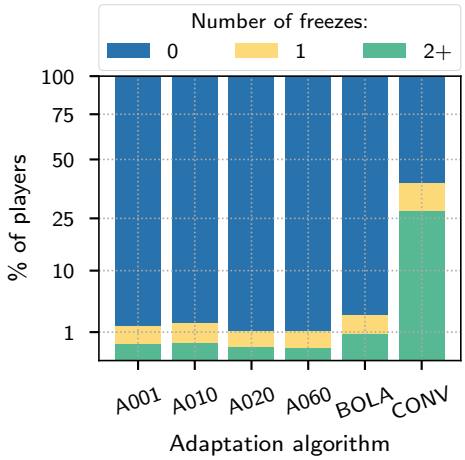
The results of the experiments where HAS players are started in parallel are presented first. This experiment represents events such as the start of a favorite live stream. The setting poses extra difficulty for the HAS players because a large number of buffering HAS players will put extra demand on the network compared to players that have a full buffer. This difficulty is visible when looking at freezing players as shown in Figure 3.31. CONV shows a high number of freezes, even when only 60 players are active. The percentage of players that experienced a freeze went up to 54.8% for 600 concurrent players. When using BOLA, the number of freezes was lower, but still, 10.5% of the player froze in the setting with 600 simultaneously active players. This indicates that freezes are a performance problem for HAS adaptation algorithms when a large number of players starts streaming at the same time. The number of players that experiences a freeze increases with the total number of players. We were able to the reduce the percentage of freezing players to 2.6% for 600 started players using our Control Element. By the Control Element assisted players will not request video segments in a bitrate higher than recommended (see Algorithm 1, line 1), and thus reduce competition for bandwidth between players that causes freezes.

The distribution of mean video bitrates is shown in Figure 3.32. The results show that the mean video bitrate is generally lower when using the Control Element compared to CONV and BOLA. For example, with 240 players the bitrate for A060 is 17% lower compared to BOLA and 10% lower compared to CONV (A060: 2.66 Mbit/s ($\sigma = 0.06$), BOLA: 3.22 Mbit/s ($\sigma = 0.17$), CONV: 2.95 Mbit/s ($\sigma = 0.53$)). The reason for the lower video bitrate when using the Control Element is twofold. First, the Service Manager in the Control Element maintains a 20% safety margin when assigning bitrates to HAS players. As a result, at least 20% of the bandwidth

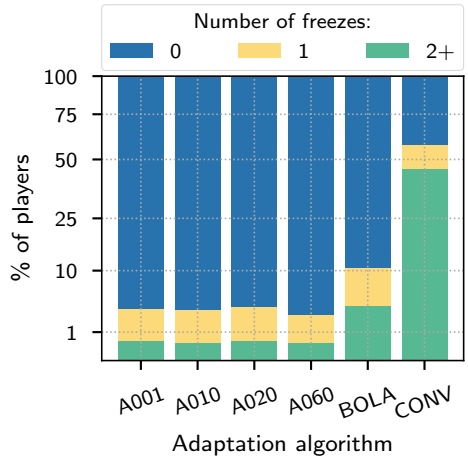


(a) 60 players

(b) 240 players



(c) 420 players



(d) 600 players

Figure 3.31: Percentage of HAS players that experiences none, one, and two or more freezes. Y-axis has squared scale.

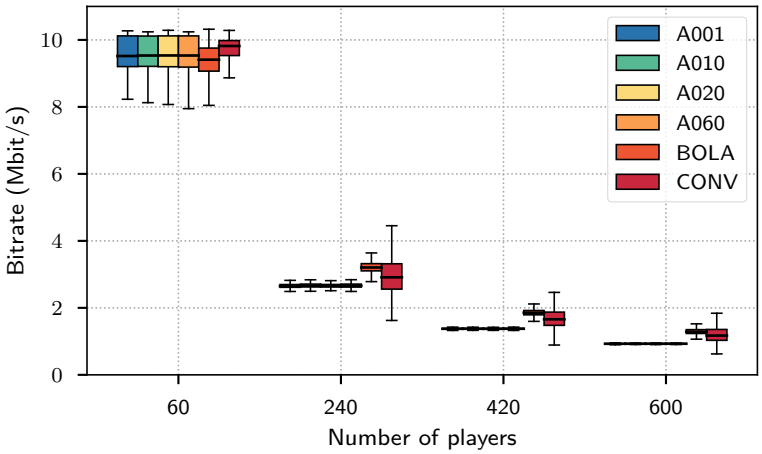


Figure 3.32: Distribution of mean video bitrates.

remains free. However, through experimentation, we found that this safety factor is necessary for stable streaming (reduce freezes and number of quality switches), especially when such a large number of HAS players is concurrently active. The second reason for the lower bitrate is the fact that the Service Manager assigns all HAS players the same bitrate. This policy could be improved by assigning a subset of HAS players a higher bitrate to increase network utilization. For example, players with a higher screen resolution could get a higher bitrate assigned, following the sharing policy described in Section 3.5.

The number of quality switches is shown in Figure 3.33. Both BOLA and TPUT cause a high number of quality switches when the number of players increases. BOLA switches on average 27.60 ($\sigma = 5.37$) times for 600 concurrent players . TPUT switches on average 72.54 ($\sigma = 6.44$) times. This high number of switches means that for the three-minute video clip (90 segments of two seconds), 81% of consecutive segments has a different bitrate. Frequent quality switches are thus also a HAS performance problem in large networks with up to 600 players. Using the Control Element to assist HAS players significantly reduces the number of quality switches. For 240 players with A060, the mean number of switches was 4.11 ($\sigma = 1.32$), a reduction of 85% compared to BOLA and 94% compared to CONV. The number of switches for A060 decreases with the number of players, in contrast to the other algorithms. More players result in a lower recommended bitrate from the Service Manager. It takes the HAS players fewer switches at start-up to reach this bitrate.

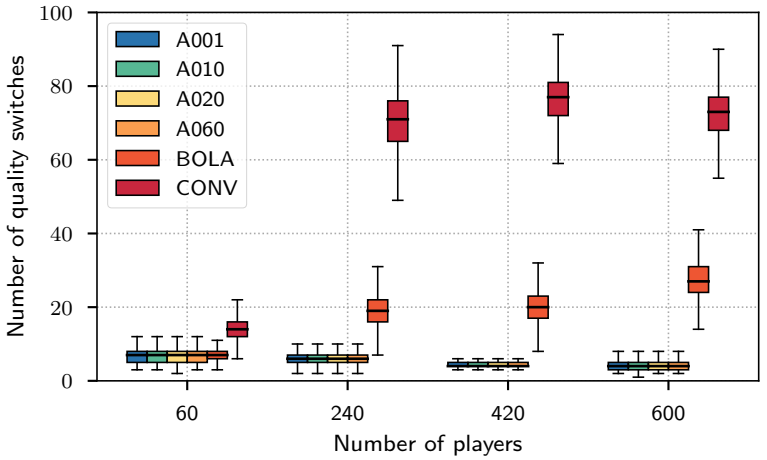


Figure 3.33: Distribution of the number of quality switches.

The differences between the different queuing configurations (A001-A060) regarding freezes, bitrate, and quality switches are small. A Kruskal-Wallis test (numerical data, non-normal distribution, unpaired, $\alpha = 0.05$) is performed with a multiple comparison posthoc test ($\alpha = 0.05$) to determine the differences. The results can be summarized as follows: There are no significant differences between the number of queues with regards to freezes. There are also no significant differences with 60 simultaneous players with regards to video bitrate and quality switches. With regards to averages for video bitrate and the number of switches for 240 players and more, there are slight differences between settings A001 and A060, in favor of A060. Nevertheless, the practical values show small differences and the number of queues thus has a limited effect.

Poisson Process Start-Up Results

In this experiment, HAS players are started following a Poisson arrival process, to target scenarios where players randomly start and stop. This setting targets environments such as hotels where people continuously watch video clips. In this experiment, we observe a lower percentage of players that freezes during playback. For example, given $\lambda = 2.9$ with an average of 510 concurrent players, CONV causes freezes in 13% of the players. For BOLA the percentage of freezing players was below 1%. With assisted adaptation (A060), freezes are eliminated ($\leq 0.01\%$). The difference in the number of freezes means that freezes are a performance problem in networks with a large number of HAS players depending on the algorithm. For $\lambda \leq 1.9$, the percentages of freezing players is below 1% for all algorithms.

	Adaptation algorithm	Mean bitrate Mbit/s (σ)	Mean nr. switches (σ)
$\lambda = 0.9$	A060	3.91 (0.43)	**6.20 (2.97)**
	BOLA	**5.17 (0.71)**	20.80 (4.11)
	CONV	4.81 (0.90)	47.81 (9.38)
$\lambda = 1.9$	A060	2.11 (0.29)	**5.02 (1.30)**
	BOLA	**2.38 (0.33)**	17.97 (3.92)
	CONV	2.17 (0.34)	54.21 (7.96)
$\lambda = 2.9$	A060	1.50 (0.25)	**7.57 (4.32)**
	BOLA	**1.54 (0.24)**	21.40 (4.83)
	CONV	1.36 (0.21)	63.27 (7.89)

Table 3.8: Bitrates and quality switches for Poisson process start-up. The best performing algorithm is marked in bold.

The mean bitrates for the three different arrival rates are listed in Table 3.8. Regarding mean bitrate, the three adaptation algorithms show comparable performance. During the experiments, there is always a fair share of the HAS players that is buffering. Those players put a more considerable demand on the network compared to players in steady-state mode. We expect that buffering players limit steady-state players in selecting higher-bitrates. With the Control Element, buffering players use the free bandwidth created by the 20% safety margin, thus not affecting the video quality of steady-state players.

Regarding the number of quality switches, the performance is similar to the experiment where the HAS players start at the same time. This means that the performance issue of quality fluctuations also exists in the scenario where HAS players gradually come and go. With the Control Element, we reduce the number of quality switches on average by 65% compared to BOLA, and 88% compared to CONV for $\lambda = 2.9$. This shows that the Control Element is not only able to provide more stable streaming in scenarios where players start in parallel but also in scenarios where the number of simultaneously active HAS players continuously changes.

3.7 Discussion

Video streaming is a popular Internet application. Given that 70% of the global Internet traffic can be accounted to video streaming, it will not be an exception to have one or more video streaming flows active in a network. Several studies, however, have shown that HAS players have difficulties sharing network bandwidth,

both with other HAS players and with cross-traffic. The problem for HAS players with respect to bandwidth sharing is two-fold. First, using HTTP/TCP for transport, a protocol that once was intended to deliver small text documents, is ill-suited for delivering high bandwidth streams with tight temporal restrictions. Nevertheless, given the reliance of streaming providers on HTTP, largely because HTTP enables massive scale delivery, we do not foresee changes to the streaming protocol in the (near) future. The second reason for performance is the lack of a common goal between HAS players (i.e., optimize the delivery of video streams for all players in the network). HAS players maximize the video quality from their own perspective, unaware of other players that may require at least part of the bandwidth.

The Control Element addresses both these issues, especially in the form of the SDN-based implementation. The communication channel between the HAS players and the Control Element facilitates finding the common goal, having the Control Element as the orchestrator. Traffic control ensures that HAS players receive the network throughput they expect, and require. In addition, traffic control prevents from overshooting HAS clients that do not adhere to the instructions from the Control Element.

The experiments in the chapter show that both mechanisms are required for optimal streaming performance, contrasting the findings from Cofano et al. in [32, 33]. In our experiments, we show how the combination of the two mechanisms outperforms configurations with only one of the mechanisms active. When only using traffic control, HAS players still have to rely on their internal bandwidth estimations. Depending on the adaptation algorithm, quality adaptation may be too conservative, thus not utilizing the full bandwidth that was reserved. From the perspective of the HAS player, the conservative attitude is understandable, because the HAS player is supposed to work in different networks with changing networking conditions. However, because traffic control is a purely network-level solution, there is no mechanisms to inform HAS players that they can use an adaptation scheme better suited for when QoS is provided.

In a similar fashion, solving the bandwidth sharing problem only on the application level is too limiting. With the Control Element, HAS players are coordinated and they fairly share the network bandwidth. However, this mechanisms lacks the guarantee that the bandwidth for streaming at the target bitrate is available. Cross-traffic and non-controlled HAS players can still take over the bandwidth assigned to HAS traffic.

This discussion shows that optimizing video streaming, particularly HAS, in shared networks requires a solution that works both on the network level as well as the application level. The Control Element is the bridge, providing integration between

media applications and networks. Because the Control Element is aware of HAS and the characteristics of the HAS players and devices, it can make decisions optimized for video streaming, as such optimizing the largest portion of network traffic.

Because the Control Element acts as a bridge between HAS players and the network, it is suitable to work with different networks. In this chapter, we used the Control Element with Wi-Fi and wired networks. In Chapter 5 of this thesis, we show the applicability of the Control Element in LTE-based mobile networks. The interface between the HAS player and the Control Element remains the same, since the Control Element adapts to the network on the opposite interface. This way, HAS players can use the Control Element independently of the network they operate in, and the Control Element can optimize its bandwidth sharing policy to the network instance.

4 | Modeling Bandwidth Sharing Policies

In the previous chapter, we report the performance of the Control Element. The evaluations show that with the Control Element the streaming experience improves in terms of less video freezes, a higher video quality, less quality switches, and fair sharing of bandwidth. This shows that the mechanisms in the Control Element are sufficient for improving the streaming experience. The Control Element divides the available bandwidth among the HAS players according to a policy. The sharing policy determines the bandwidth share of each player. Testing sharing policies in different networks is time consuming. Therefore, in this Chapter, we present a performance modeling method for analyzing sharing policies, such that results and insights can be obtained faster and policies can be tailored to the user requirements given a network environment.

This chapter is based on the publications [K7][K8][K10][K11].

4.1 Introduction

The policy that has been used in most of Chapter 3 is simple: all HAS players get the same video bitrate. While this policy is effective for demonstrating the mechanisms in the Control Element, realistic policies where different players can be treated differently may be preferred. When defining a sharing policy, one must analyze the effect of a sharing policy on the streaming experience. Depending on how bandwidth is shared, the video quality, video quality switches, and fairness between different streams is affected. The effect of a sharing policy in a small network with only a handful HAS players is clear. However, we also aim for the Control Element to be deployed in larger networks with many HAS players. For instance, the Control Element may deployed in the access network of an ISP, where a network link may be shared with tens or hundreds of players. In such larger networks, analyzing the expected viewing experience is not straightforward. Players may start and stop, and the number of active HAS players might strongly vary over time. Depending on how often players start a video and the duration of this video, the effect of a sharing policy on the overall streaming experience may change.

The empirical approach of experimenting with policies in testbeds is costly and time-consuming. The experimentation process involves long setup- and run times. Furthermore, simulations and experimental runs are prone to errors. Because analyzing and adapting sharing policies include frequent changes to the policies, new and faster methods for analyzing policies may be preferred.

In this chapter, we investigate the potential of using Markov modeling for the analysis of sharing policies. In the first part of this chapter, a Markov model is formulated that takes a sharing policy and the expected load on the network as input, and outputs the expected streaming experience in terms of video quality and quality switches. In the second part of this chapter, the model is validated agains experiments using the proxy server implementation of the Control Element. As such, the contributions in this chapter answer the third research question in this thesis:

Research question 3: *How can behavior of sharing policies in the Control Element be stochastically modeled and evaluated?*

4.2 Model Description

Sharing policies divide the available bandwidth based on the number of HAS players. For estimating the result of a sharing policy, the number of players that is active at the same time must be known. Since players can start and stop, this means

keeping track on how the number of players changes over time. In this chapter, a Markov-based birth-death process is used to model the number of players and its changes. Because a sharing policy must be able to distinguish between different types of players, a multi-dimensional birth-death process is used. Each dimension represents a different group. Let K be the number of groups in the modeled sharing policy, the state space $\mathcal{S}$ of the model is defined by the vector $(n_1, n_2, n_k, \ldots, n_K)$, where n_k is the number of players in group k.

The state space $\mathcal{S}$ is limited by the capacity C of the network connection and the lowest available bitrate for players in group k, denoted by $\tilde{B}_k$. Allowing more players than there is capacity for would create a too heavy demand on the network. This could result in video freezes, which should be avoided at all times. Therefore, the Control Element denies players access when the network is full. This implies that the state space of the model is finite. The state space is defined as all states with non-negative integer valued entries $(n_1, n_2, n_k, \ldots, n_K)$ that satisfy the condition stated in Equation 4.1.

$$\sum_{k=1}^{K} n_k \tilde{B_k} \leq C \tag{4.1}$$

Transitions between states in $\mathcal{S}$ represent starting and stopping players. Assuming that players start at random, players of group k start according to a Poisson process with intensity λ_k. The overall process of starting players is the K combined Poisson processes with intensity denoted as λ. The video duration is a good indicator for the rate at which players stop. As such, the rate for transitions $n_k \rightarrow (n_k - 1)$ becomes n_k/β_k, where β_k is the mean duration of videos (or mean duration of video being watched in the case of early stopped videos) in group k.

The Markov model that is described here is equal to the well-known Erlang multi-rate model. For this model, a stationary distribution exists with a product form solution that is given in Equation 4.2. The normalization factor G is described in Equation 4.3.

$$\pi(n_1, n_2, n_k, \ldots, n_K) = \frac{1}{G} \prod_{k=1}^{K} \frac{(\lambda_k \beta_k)^{n_k}}{n_k!} \tag{4.2}$$

$$G = \sum_{\underline{x} \in \mathcal{S}} \prod_{k=1}^{K} \frac{(\lambda_k \beta_k)^{n_k(\underline{x})}}{n_k(\underline{x})!} \tag{4.3}$$

Notation	Description
Input	
C	Capacity of the network connection
λ_k	Rate of the Poisson process at which group-k users start the video streams
β_k	Mean duration of the video streams for group k
B_k	Available bitrates for video streams for group k
$T_{segment_k}$	Segment size used in the video streams for group k
Intermediate	
$\mathcal{S}$	State space of the Markov process
$\pi(\underline{x})$	The probability that the Markov process is in state $\underline{x}$
$n_k(\underline{x})$	The number of group-k players in state $\underline{x}$
$q_k(\underline{x})$	The bitrate of group-k players in state $\underline{x}$
$\gamma(\underline{x} \rightarrow \underline{y})$	The number of group-k players that change video bitrate when transitioning from state $\underline{x}$ to state $\underline{y}$
Output	
$\mathbb{E}[N_k]$	The expected number of players of group k
$\mathbb{E}[Q_k]$	The expected number of bitrate switches for players of group k
$\mathbb{E}[B_k]$	The expected bitrate for players of group k

Table 4.1: Model notation.

The Erlang multi-rate model is insensitive to the service time distribution. Given that the model in this chapter is based on the Erlang multi-rate model, it has the advantage that the insensitivity property is inherited. This means that the model for estimating the results of a sharing policy is insensitive to the video duration distributions β_k.

The expected number of players can be found by averaging over the number of players in each in each state, weighting them by the probability that the Markov process is in that state $\pi(\underline{x})$. Equation 4.4 formulates the expected number of players in group k. A shorthand notation $\underline{x}$ is used to describe a state in $\mathcal{S}$. A summary of the notations used for the model in this Chapter is listed in Table 4.1.

$$\mathbb{E}[N_k] = \sum_{\underline{x} \in \mathcal{S}} \pi(\underline{x}) n_k(\underline{x}). \tag{4.4}$$

4.2.1 Video Quality

To analyze a policy, the policy has to be computed for each possible combination of players. Therefore, the policies will be applied to every state in $\mathcal{S}$. This means that for every state, the video bitrate for players of all groups in that state is known. The notation $q_k(\underline{x})$ is used to denote the video bitrate of group-k players in state $\underline{x}$. From this information, the expected bitrate can be obtained by averaging the video bitrates in all states, weighted by the number of players $n_k(\underline{x})$ in each state and the probability that the Markov process is in that state $\pi(\underline{x})$. The expected video bitrate for players in group k is defined in Equation 4.5.

$$\mathbb{E}[B_k] = \frac{1}{\mathbb{E}[N_k]} \sum_{\underline{x} \in \mathcal{S}} \pi(\underline{x}) n_k(\underline{x}) q_k(\underline{x}). \tag{4.5}$$

This estimation of the video bitrate relies on the number of parallel streams, which in turn is influenced by the duration of the video streams β_k. However, from a network perspective, the time that HAS players are actively downloading is typically shorter than the video duration. This effect is caused by a prefetching process in the player, sometimes known as initial buffering. At the start of a stream, a HAS player builds up a buffer by downloading more video segments than it will play out. Once a sufficient buffer level is reached, HAS players change from prefetching to steady-state mode. In steady-state mode, download and playback of video segments are at the same rate.

The difference between prefetching and steady-state mode is shown in Figure 4.1. The figure shows the downloads of video segments. In steady-state mode, a HAS player will wait before downloading the next video segment, when downloading a video segment takes shorter than the playback time of that segment. With prefetching, players will continue immediately with the download of the next segment. In Figure 4.1, prefetching stops after the third segment, resuming in steady-state mode. This illustrates how prefetching shortens the time active in the network compared to the video duration.

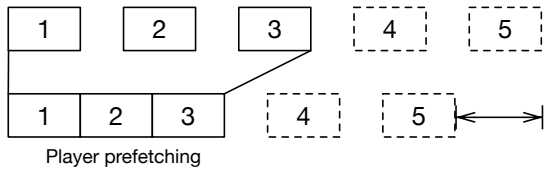


Figure 4.1: Steady-state mode (top) versus prefetching (bottom): prefetching causes a player to be network active for a shorter period of time.

Once the last video segment is downloaded, the Control Element assigns the bandwidth that becomes available to the remaining players. This causes Equation 4.5 to underestimate the video bitrate. The impact of prefetching on the video bitrate increases with the buffer size. Depending on the stream, different buffer sizes may be used. For live streams a small buffer will keep the play-out point close to the actual event. For Video on Demand (VoD) this timing requirement can be relaxed and larger buffers, with a better resilience against video freezes, are more common. To account for large buffers and increase the accuracy of the model, the model is corrected to use the network active time instead of the video duration. The network active time is estimated based on both prefetching- and steady-state phase.

To reach a certain buffer level *Buff* so that the player can go into steady-state mode, $\lceil \alpha + 1 \rceil$ segments have to be downloaded. The definition of α is given in Equation 4.6. The download of the first video segment – while there is no outflow from the buffer – is accounted for by subtracting $T_{segment}$ (the segment size) from the total buffer level in Equation 4.6, and increasing α by one to come to the total number of video segments that is required to be downloaded to reach *Buff*. The number of segments is rounded up, $\lceil \alpha + 1 \rceil$, because moving from prefetching to steady-state phase can only occur in-between segments, but not during segment downloads.

$$\begin{aligned} Buff - \mathrm{T}_{\mathrm{segment}} &= (T_{segment} - T_{download})\alpha \\ \alpha &= \frac{Buff - \mathrm{T}_{\mathrm{segment}}}{T_{segment} - T_{download}} \end{aligned} \tag{4.6}$$

The effective service time β_{eff} that video players are active in the network, given a certain video length β_{video} and $\beta = \beta_{video}$, is defined in Equation 4.7. The effective service time β_{eff} is lower than the actual service time β that is used in the model. Therefore, to obtain a more accurate mean bitrate, β has to be lowered to match β_{eff}. However, $T_{download}$ and α are dependent on β and lowering β will thus affect β_{eff}. The intersection $\beta = \beta_{eff}$ is found by iteratively lowering β while keeping β_{video} constant.

$$\beta_{eff} = \lceil \alpha + 1 \rceil (T_{download} - T_{segment}) + \beta_{video}. \tag{4.7}$$

By using the effective service time instead of the video duration, the number of players active in the network is modeled closely to the real operation of the Control Element, thus it increases the accuracy of the model. This is proven later in this chapter using experiments dedicated to different buffer sizes.

4.2.2 Quality Switches

The number of quality switches relates to how often the Markov process transitions between states. When the Control Element assigns a different bitrate for a group of players in two states, a bitrate switch is made when the process transitions between those states. The intuition behind determining the number of bitrate switches is that by observing the number of transitions between states with different target bitrates, the number of quality switches can be obtained. However, HAS players can technically only switch in between segments, and not during the download of a segment. The reason for this is that the requested video quality is part of the HTTP request, and only when making a new request a new bitrate can be selected. Therefore, to include this behavior of the HAS player in the model, the Markov process is observed in intervals that are equal to the segment size. The probability that the process transitions from state $\underline{x}$ to state $\underline{y}$ in $T_{segment}$ seconds can be retrieved via uniformization of the continuous time Markov chain, by conditioning on the number of steps m.

If $P^m_{\underline{x},\underline{y}}$ is the probability that the Markov process transitions from state $\underline{x}$ tot state $\underline{y}$ in m steps, and if b is the uniform rate parameter, then the probability that a transition $\underline{x} \to \underline{y}$ occurs in $T_{segment}$ seconds is defined in Equation 4.8.

$$P_{\underline{x},\underline{y}} = e^{-bT_{segment}} \sum_{m=0}^{\infty} \frac{(bT_{segment})^m}{m!} P^m_{\underline{x},\underline{y}} \qquad \text{for } \underline{x}, \underline{y} \in \mathcal{S}. \tag{4.8}$$

The duration of the videos is variable, therefore we express the number of switches in video quality not as an absolute number but as a rate: number of bitrate switches per second. The expected bitrate switch rate is defined in Equation 4.9.

$$\mathbb{E}[Q_k] = \frac{1}{T_{segment}\mathbb{E}[N_k]} \sum_{\underline{x},\underline{y} \in \mathcal{S}} \pi(\underline{x}) P_{\underline{x},\underline{y}} \gamma_k(\underline{x} \to \underline{y}), \tag{4.9}$$

$$\gamma_k(\underline{x} \to \underline{y}) = \begin{cases} 0 & \text{if } q_k(\underline{x}) = q_k(\underline{y}), \\ \min(n_k(\underline{x}), n_k(\underline{y})) & \text{if } q_k(\underline{x}) \neq q_k(\underline{y}) \end{cases} \tag{4.10}$$

Equation 4.10 defines $\gamma_k(\underline{x} \to \underline{y})$, which is the number of players in group k that make a bitrate switch on the transition $\underline{x} \to \underline{y}$. Note that the bitrate instability rate is defined from the viewpoint of a single player. Players in group k only make a bitrate switch when the bitrate in state $\underline{x}$ is different from the assigned bitrate in state $\underline{y}$. Furthermore, the number of players that make a switch is limited to the players that are both active in $\underline{x}$ and $\underline{y}$. A player that is started will begin streaming

at the selected bitrate and does not have to make a switch. Similarly, a player that terminates a stream cannot make bitrate switches anymore.

4.2.3 Estimating the Overall Performance

Equations 4.5 and 4.9 are defined per group k to allow for a more detailed analysis. Equation 4.11 shows how the overall expected bitrate and quality switches can be found via a weighted average, weighted by the mean number of players for each group.

$$\mathbb{E}[B/Q] = \frac{\sum_{k=1}^{K} \mathbb{E}[N_k] \cdot \mathbb{E}[B_k/Q_k]}{\sum_{k=1}^{K} \mathbb{E}[N_k]}. \tag{4.11}$$

4.3 Validation Experiments

The model that is presented in the first part of this chapter focusses on predicting the video quality and video quality switches given a network environment and sharing policy. The accuracy of this model and the sensitivity of this model to different sharing policies is evaluated in the remainder of this chapter. To assess the accuracy of the model, the outcomes of the model are compared to the experiments with the proxy-server based implementation of the Control Element. The expressiveness of the model is demonstrated through modeling several sharing policies that differ in the number of groups and how the bandwidth is shared among active HAS players.

4.3.1 Experimental Setup

The Control Element and its sharing policies are intended to be used on bottleneck network connections that are shared by multiple HAS players. For the experiments validating the model in the chapter, a simulated environment with 20 HAS players that share a single network connection is used. All devices in the testbed, including the HAS players, network elements, and HTTP servers, are implemented as Linux-based lightweight virtual machines. All VMs run on a host server with a 2.83 GHz processor, 8 GB memory, running the GNU/Linux Debian 6 operating system.

The VMs are lightweight, meaning that they share the kernel and libraries with the host operating system, but each VM maintains its own process- and network stack. The network connections between the VMs are emulated. The properties of the network connections are set using Linux tc [63]. The configuration of VMs

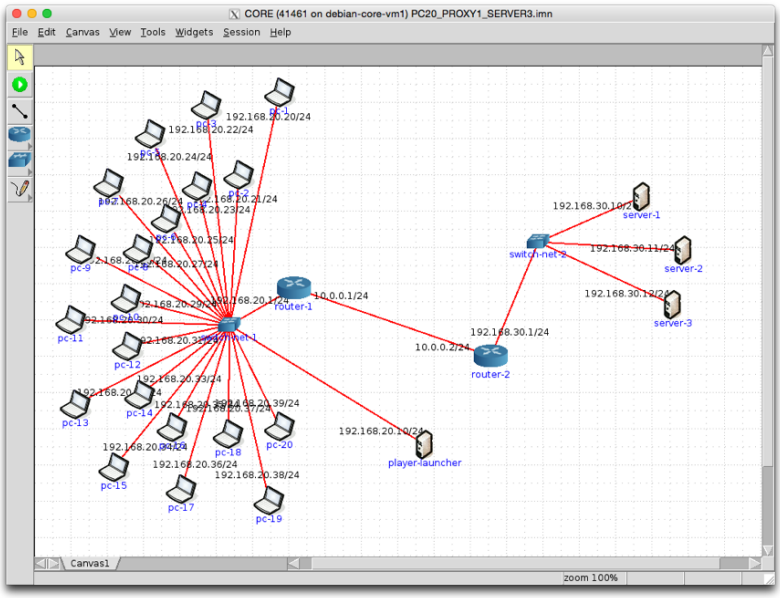


Figure 4.2: Testbed setup with lightweight virtual machines and emulated network connections.

and network connections was done using the CORE network emulator [134]. A screenshot of the setup in the CORE network emulator is shown in Figure 4.2.

The test environment consists of 20 virtual PCs, two switches, two routers, three HTTP servers, and a machine that controls the experiments. The bottleneck link is the network connection between the two routers. The capacity of this link is limited to 8 mbit/s, representing a network links that is not sufficient for multiple video streams at the highest quality. In other words, given the capacity of the bottleneck link, the Control Element has to execute a sharing policy that allocates the bandwidth among HAS players. The round-trip delay between the HAS and HTTP servers is set to 10, 20, and 40 milliseconds respectively.

The router closest to the HAS players (router-1 in Figure 4.2) acts as the Control Element. The proxy server implementation of the Control Element is installed on this router. Although the capacity of the bottleneck link is 8 Mbit/s, the proxy server is configured with a maximum of 6.8 Mbit/s, implementing a 15% safety margin on the total capacity of the network connection. The safety margin allows HAS players to establish healthy buffer levels. Furthermore, a capacity margin would allow for light background traffic, although background traffic is not used during the experiments in this chapter.

HAS players are started by the player launcher. The player launcher signals clients to start a HAS player according to one or more Poisson processes. The player launcher only starts new HAS players on free clients, thus each client only hosts a single instance of the HAS player at a time. Different video streams with different available video bitrates are used during the experiments. However, the lowest available bitrate is 400 Kbit/s for all used video streams. Therefore, the maximum number of concurrent HAS players should not exceed 17. An 18th HAS player will be denied service by the proxy server, ensuring sufficient bandwidth and preventing video freezes for the 17 already active video players.

The customized HAS player used in the experiments is stripped down to reduce the load on the host machine. Decoding and rendering of the video streams is disabled, significantly lowering the CPU and memory consumption by the HAS players. All video streams used in the evaluations are segmented in 4 second segments in Transport Stream (TS) format. A manifest file following the HLS [101] specification is added. The manifest and segment files are hosted on three off-the-shelf HTTP servers using Apache 2.2.22.

The video segment URLs are formatted such that they contain all information required to measure the streaming performance. The URL includes an identification number for the HAS player, an identifier of the video stream, the segment index, and the requested video bitrate. During the experiments, HTTP traces were collected using tcpdump [131]. Through analysis of the HTTP traces the mean video bitrate and the mean number of quality switches can be obtained.

4.3.2 Bitrate Fairness Policy

The first validation experiment focusses on the the bitrate fairness policy that has been used throughput Chapter 3. Bitrate fairness means that every device, disregarding the device type or user, will be allocated the same video bitrate. The first experiments covers the bitrate fairness policy in a scenario where video streams origin from a single content provider that uses a fixed encoding scheme. As such, this scenario may be relevant for ISPs that provide an on-demand video streaming service on a section of their network managed by the Control Element. For this scenario, it can be assumed that the available bitrates in the HAS manifest are similar for each player. Therefore, from a modeling perspective, the players do not have to be distinguished from each other using different groups. The bitrate fairness policy in this scenario can thus be modeled with the one-dimensional version of the model.

The bitrate fairness policy is applied to each state in $\mathcal{S}$. For each state, the target bitrate for HAS players in that state is modeled as the highest available bitrate in the

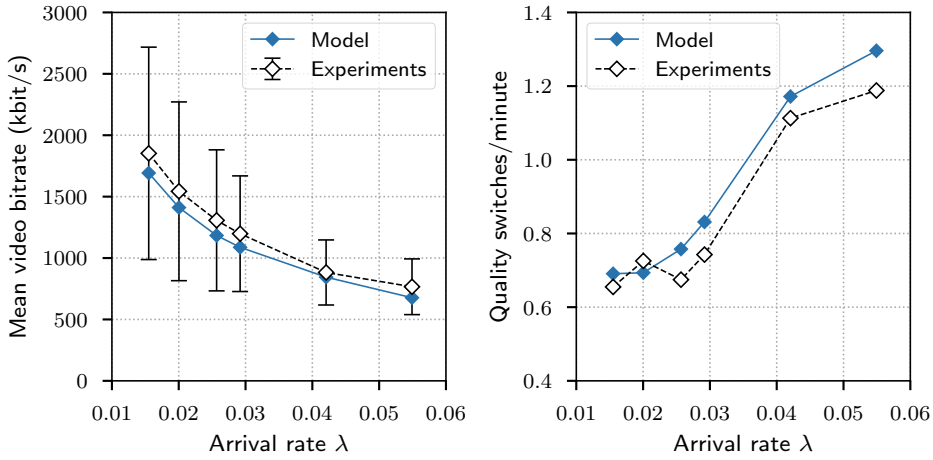


Figure 4.3: Comparison between the model and the performance measured in experiments for the bitrate fairness policy: mean video bitrate (left) and mean video quality switches (right).

manifest, that is equal or lower to the capacity of the network connection equally divided by the number of players represented by the state. The parameters in this experiment that apply to both the model and the experimental runs are as follows: Video streams are encoded using the bitrates $B_1 = \{400, 720, 1020, 2300, 4200\}$. The segment size $T_{segment}$ is 4.0 seconds. The mean video stream duration β_1 is 140 seconds. The Poisson arrival process for starting players has rates that are varied between $\lambda_1 = 0.015$ to $\lambda_1 = 0.055$. Each experiment runs for 24 hours per condition.

The comparison of the expected streaming performance based on the model and the actual performance measured in the testbed is shown in Figure 4.3. The comparison for the mean video bitrate is shown on the left. For all arrival rates, the modeled bitrate is lower compared to the mean bitrate obtained in the experiments. The modeled mean bitrate in this experiment does not include the correction for buffer pre-fetching (the effect of this correction is demonstrated in a dedicated section later in this chapter). This means that the mean video duration is used to model the expected video bitrate, which will result in a slightly less accurate prediction. Nevertheless, the model still provides a reasonable prediction of the mean video bitrate. On average, the difference between the modeled bitrate and the experiments is 8.8%. This is well in between the standard deviation of the measured bitrates, indicated by the error bars in Figure 4.3 (left).

The right side of Figure 4.3 shows the comparison between the modeled quality switches and the quality switches measured in the experiment. This comparison shows that the model also provides a proper indication for the number of quality switches that one can expect when using the bitrate fairness policy in the Control Element. The largest difference in quality switches rate between the model and the experiments is at $\lambda_1 = 0.055$, a difference of 9%. To put this into perspective, for every 556 seconds of video stream, the model counts one switch more than was measured in the experiment. As such, the differences between model and experiments are negligible.

4.3.3 Groups with Different Bitrates

The previous evaluation is useful when managing streaming for a single service with the Control Element. When managing HAS streams on a regular network connection, it is likely that video from different content providers are streamed. Those streams may have different characteristics. Therefore, in this section, the bitrate fairness policy is modeled for video streams with different bitrates in the manifest and with a different mean video duration. In addition, the starting rate for players in one group is configured higher. The evaluations covers both the streaming performance per group, as well as the overall performance estimated using Equation 4.11.

To demonstrate the performance of the model with multiple groups, the bitrate fair policy is modeled for two groups with different stream characteristics. The first group streams a video available in the bitrates $B_1 = \{400, 720, 1020, 2300, 4200\}$. The mean duration of the streams in group one is $\beta_1 = 140$ seconds. The bitrates in the manifests for the second group are $B_2 = \{400, 800, 1200, 1600, 2200, 3000, 4000\}$. As shown in this example, both the bitrates and the number of bitrates can vary between groups. The mean duration for streams in group two is $\beta_2 = 100$ seconds. The segment size $T_{segment}$ for both streams is 4.0 seconds. The arrival rates for group one are varied between $\lambda_1 = 0.005$ and $\lambda_1 = 0.019$. The arrival rates for the second group follow an independent Poisson process with rates varying between $\lambda 2 = 0.010$ and $\lambda_2 = 0.038$. Each experiment is run for 24 hours per condition.

Figure 4.4 shows the modeled and measured video bitrates for the bitrate fair policy with two groups. It shows that for both groups, the model can reasonably predict the mean video bitrate. Splitting the analysis into two groups reveals that the overall video bitrate is higher for players of the second group, even though all players get the same share of bandwidth assigned at all times by the policy. The explanation for this difference is two-fold. First, the combination of λ_2 and β_2 give a higher probability that players from the second group are active, compared to the probability that players of the first group are active given λ_1 and β_1. As such,

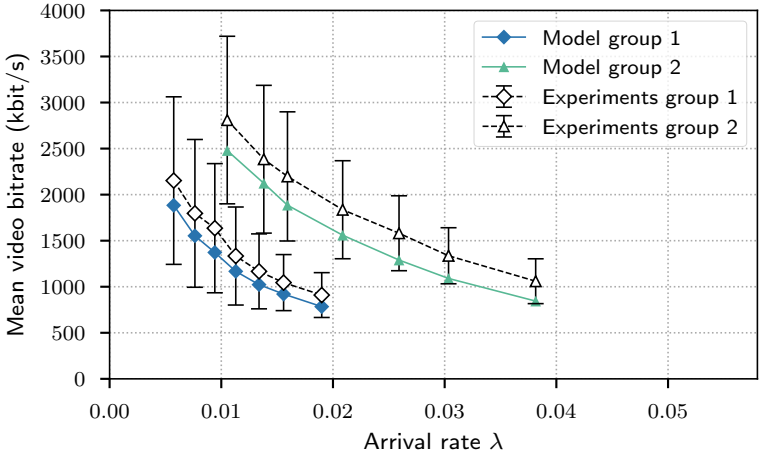


Figure 4.4: Comparison of the modeled video bitrate and the mean video bitrate measured in experiments using the bitrate fair policy for two groups of players with different stream characteristics.

a starting player of the first group will most likely enter a network where there are players of the second group already active. When often entering an already crowded network, has the effect of decreasing the overall video bitrate.

The second factor for the higher mean bitrate of players of group two, is that the second group has more bitrates available in the HAS manifest. Given the nature of the bitrate fair policy, where the highest bitrate that is equal or lower than the fair share of bandwidth is selected as the target bitrate, more available bitrates make a closer mapping between target bitrate and bandwidth share. In other words, overall, players of the second group can use more of the bandwidth share that they get assigned, compared to player from group one.

The difference in available bitrates in the manifest also explains the difference in accuracy of the model between the two groups. Because the model without the buffer correction overestimated the number of players, the bitrates will be lower in the model than in the experiments. When a new player starts, it is evaluated by the sharing policy if the video bitrates should be lowered. Changing the bitrate goes in discrete steps, following the bitrates that are listed in the HAS manifest. With a large step size (i.e., the difference between two consecutive bitrates), it may require several starting players before the target bitrate is lowered. This means that overestimating the number of players, thus counting an extra active player, may not always affect the target bitrate, when steps between consecutive bitrates in the manifest are large. Reasoned from the perspective of group two, the effect of

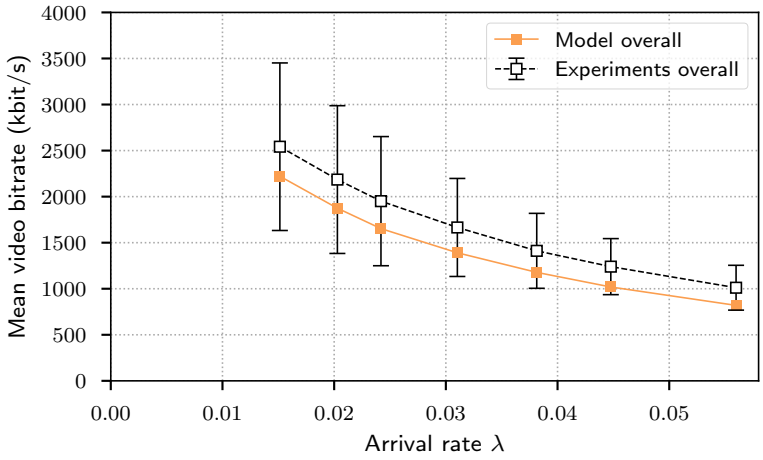


Figure 4.5: Comparison of the overall modeled video bitrate and the mean video bitrate measured in experiments for the bitrate fair policy.

overestimating the number of players is bigger for the group with a higher number of available bitrates in the manifest.

The comparison of the overall modeled and measured mean video bitrate is shown in Figure 4.5. Similar to the predictions of the video bitrate for the individual groups, the model predicts a lower video bitrate than was measured in the experiments. On average, the modeled bitrate is 16% lower. This difference clearly indicates that for a more accurate prediction of the video bitrate, an alternative for the video duration as input parameter has to be used.

The rate of quality switches when using the bitrate fair policy in the Control Element is visualized in Figure 4.6. This figure shows that it is more likely that players from the second group encounter a quality switch. The two extra bitrates in the HAS manifest are (again) the source for the differences between the two groups. The two extra bitrates cause quicker quality switches as a response to starting and stopping players. However, the differences between the two groups are small. It can even be argued that one additional quality switch per minute is barely noticeable by the user, especially when the quality switch size is smaller. Nevertheless, the model provides an accurate prediction and insight to when the Control Element will be active (i.e., send target bitrates to the HAS clients).

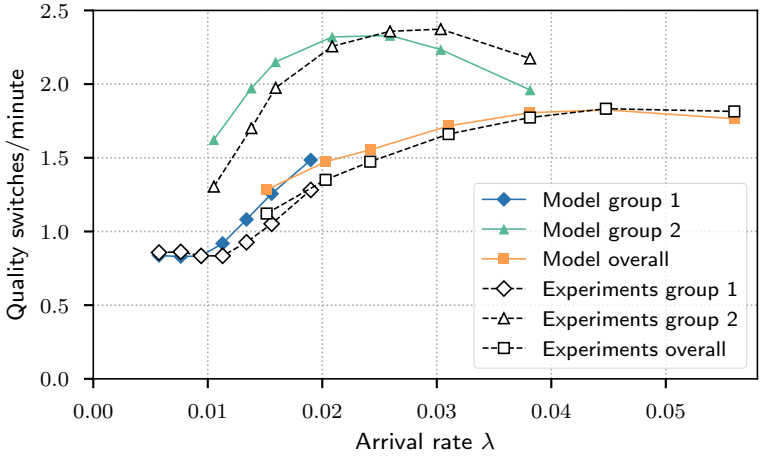


Figure 4.6: Comparison of the modeled quality switches rate and the quality switches as measured in experiments for the bitrate fair policy.

4.3.4 Video Buffer Size Correction

The validation experiments with the bitrate fair policy revealed an inaccuracy of the model towards the mean video bitrate. The model consequently predicts the video bitrate lower than it was measured in experiments with the Control Element. As stated before, the difference between the modeled video duration and the network activity time used in the Control Element, causes the model to overestimate the number of concurrent HAS players. This difference is caused by the prefetching behavior in HAS players. The more video segments a HAS player buffers, the shorter the network activity time will be, increasing the differences between model and reality.

The impact of the buffer size on the mean video quality in shown in Figure 4.7. The HAS players were configured to use 8, 16, and 24 seconds buffers respectively. All players streamed the same video with bitrate $B_1 = \{400, 720, 1020, 2300, 4200\}$ kbit/s. The video duration was $\beta_1 = 144$ seconds. A segment size $T_{segment}$ of 4.0 seconds was used. In 12-hour experiment runs, varying the starting player rates from $\lambda_1 = 0.015$ to $\lambda_1 = 0.055$, it is shown that the video quality increases with the buffer size in the HAS players.

HAS players have the ability to buffer video segments because of the 15% safety margin on the total capacity of the bottleneck network connection. In addition, the sum of target bitrates almost never matches the network connection's capacity. The bitrate fair policy assigns target bitrates that are equal or lower than the fair share

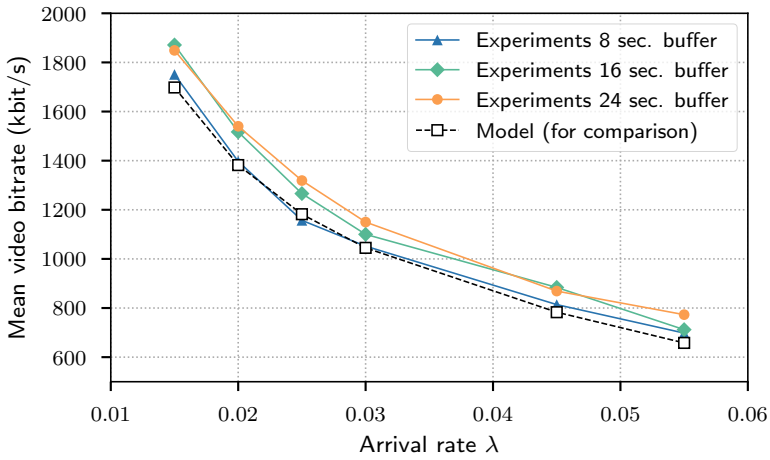


Figure 4.7: The effect of the players' buffer size on the mean bitrate of HAS streams. The modeled bitrate is included for comparison.

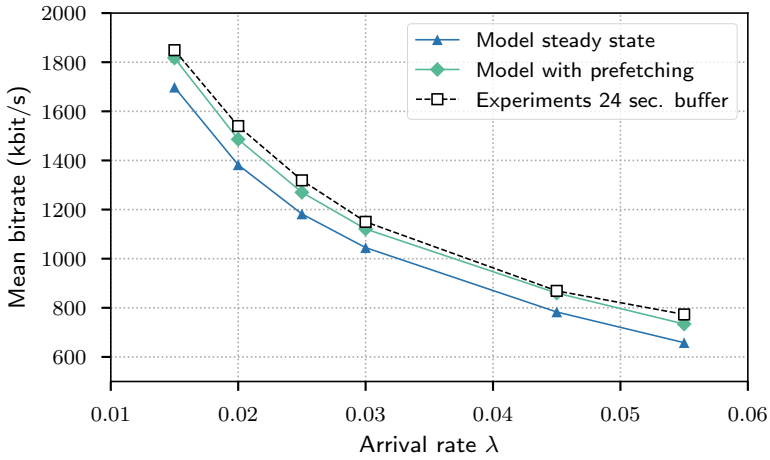


Figure 4.8: Comparison of the steady-state model-based expected bitrate, the expected bitrate using the model including prefetching, and the mean bitrate achieved in experiments.

of bandwidth. The free bandwidth between an available bitrate and the fair share of bandwidth can be used for increasing the video buffer. Such a policy may not be the most efficient in terms of network usage and maximizing video bitrate, it does provide great robustness against video freezes.

Preventing video freezes is the main argument for using large video buffers. Even though the Control Element ensures sufficient bandwidth, a large buffer provides additional safety to counter temporary drops in throughput happening outside the bottleneck links, and thus outside the scope of the Control Element. Figure 4.7 shows that the steady-state model is accurate for small buffers. However, considering the impact of video freezes on the streaming experience, it is more realistic that large buffers will still be used with the Control Element. Therefore, a correction to the model was proposed in the first part of this chapter. The results for the buffer correction on the prediction of the mean video bitrate by the model is shown in Figure 4.8. It shows that including prefetching behavior in the model makes the model a highly accurate predictor for the mean video quality.

4.3.5 Device Heterogeneity

The bitrate fairness policy operates under the assumption that all devices are equal, or that assigning the same target bitrate to each device will yield an equal streaming experience. However, nowadays a variety of devices may be used for video streaming. For instance, mobile devices have become more powerful and are fully enabled for video streaming using wireless network connections. Traditional devices, such as television sets, now also come with network connections. Video streaming services are available on a broad range of devices, creating an interesting mix of devices with potentially different usage patterns. Because of the different screen sizes and resolutions, it is not fair to equally divide the available bandwidth among the HAS players. Doing so, would not provide each user a similar video streaming experience.

Georgopoulos et al. describe how different bitrates and resolutions can be compared among devices with different form factors [50]. In the evaluations in this section, a similar mapping between bitrate and screen resolution is used. Devices are classified into three groups. The first group is smartphone sized devices that stream a maximum 360p resolution video of $\beta_1 = 60$ seconds, encoded at bitrates $B_1 = \{400, 600, 1000\}$ kbit/s. The second group represents tablet viewers that stream a maximum 720p resolution video of $\beta_2 = 120$ seconds, encoded at $B_2 = \{400, 600, 1000, 1500, 2000\}$ kbit/s. The third group is large screen devices that stream a maximum 1080p resolution video of $\beta_3 = 180$ seconds, encoded at $B_3 = \{400, 600, 1000, 1500, 2000, 4000\}$ kbit/s.

Quality level	360p	720p	1080p
1	1000	2000	4000
2	600	1500	2000
3	400	1000	1500
4	400	600	1000
5	400	400	600
6	400	400	400

Table 4.2: Device-aware video quality mapping (in kbit/s).

Players report their (simulated) screen resolution to the Control Element. This screen resolution puts the players in one of the three groups as defined by the policy. Based on the screen resolution and available bitrates, a device-aware quality mapping is created and listed in Table 4.2. Depending on the number of players in each group, the device-aware policy selects the highest quality level from Table 4.2 that fits the capacity of the bottleneck network connection. For example, Equation 4.12 shows the test if the current active players would fit the capacity of the network given quality level 2.

$$\#360\text{p} \cdot 600 + \#720\text{p} \cdot 1500 + \#1080\text{p} \cdot 2000 \leq C. \tag{4.12}$$

For quality level 1-3, the perceived video quality is supposed to be similar for the different screen resolutions. Devices with a larger screen will get a comparable increase in bitrate. From level four and up, it is not possible to maintain similar perceived quality between screen resolutions. However, when the available bandwidth allows it, the bitrate of the 1080p group of devices is higher than that of the 720p group, and the bitrate of 720p group is higher than the 360p group.

The bitrate fair policy from the previous sections is compared to the device-aware policy. Figure 4.9 shows the modeled mean bitrates for the two policies. The arrival rates for the three groups are varied between $\lambda_{1,2} = 0.0025$ and $\lambda_{1,2} = 0.0030$ for the groups of 360p and 720p resolution devices, and between $\lambda_3 = 0.00125$ and $\lambda_3 = 0.0150$ for the group with 1080p resolution devices. The arrival rate λ in Figure 4.9 is the combined arrival rate $\lambda_1 + \lambda_2 + \lambda_3$, for the three independent Poisson processes. The mean duration of the video streams is $\beta_1 = 60$, $\beta_2 = 120$, and $\beta_3 = 180$ seconds respectively.

For the bitrate fair policy the differences in mean bitrate between the groups is explained by the extra (higher) available bitrates for larger screen devices. When the network gets more crowded, the higher resolution devices cannot stream at

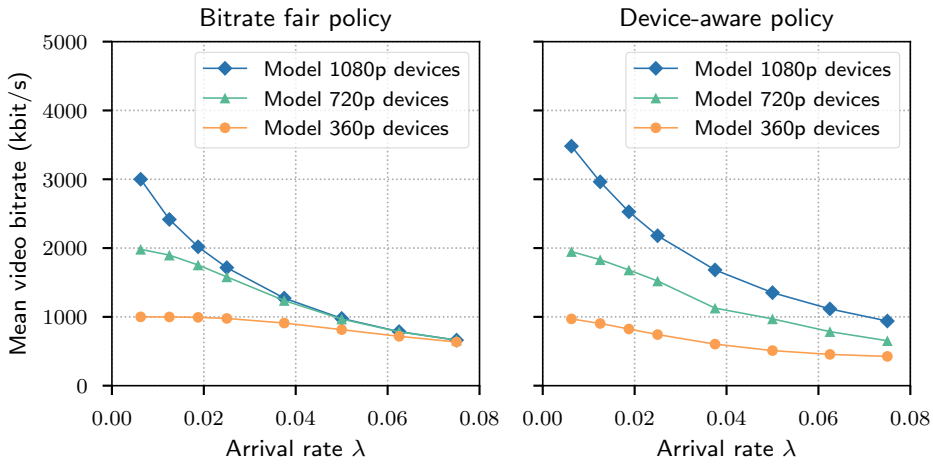


Figure 4.9: Model-based comparison of the mean bitrate for the bitrate fair policy (left) and a device-aware policy (right).

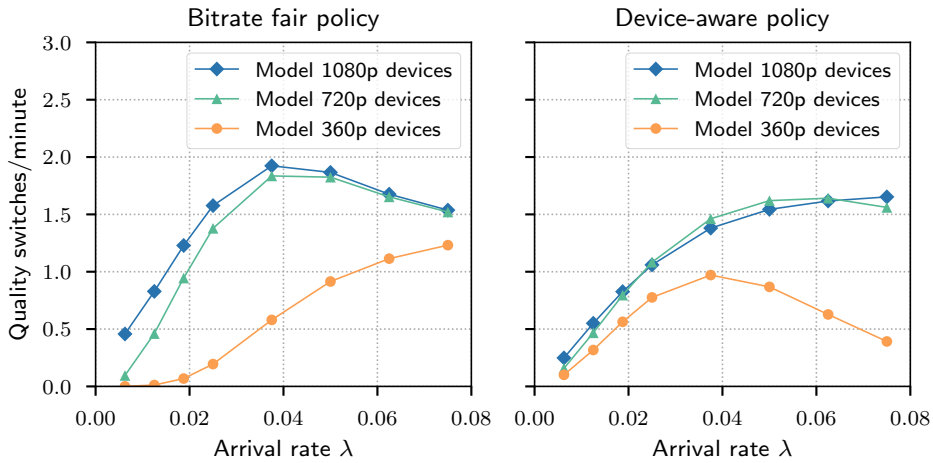


Figure 4.10: Model-based comparison of the quality switches for the bitrate fair policy (left) and a device-aware policy (right).

the higher available bitrates. When increasing the number of active players, the mean bitrates for the three groups converge to the same bitrate. This effect can be expected from a policy where each player gets the same share of network resources. The device-aware policy does not show this convergence (although it will eventually happen when increasing the network load much further). For 720p resolution devices, there is no difference between the two policies. However, Figure 4.9 shows that 360p devices will go to lower video bitrates to make room in the network for 1080p devices.

The process of making room for large screen devices also shows in the number of quality switches, especially in the different slopes for 360p resolution devices in Figure 4.10. For the bitrate fair policy, the number of quality switches is highest around $\lambda = 0.075$. The device-aware policy creates a peak in quality switches around $\lambda = 0.04$. When the number of players is low under the bitrate fair policy, 360p resolution devices can stream a video at the highest available bitrate of 1000 Kbit/s. The fair share of bandwidth is likely to be much higher than 1000 Kbits/s. While increasing the intensity of starting players, the difference between the fair share of bandwidth and the highest available bitrate becomes smaller. Therefore, the bitrate fair policy starts to cause quality switches for the higher arrival rates. In case of the device-aware policy, there is no free bandwidth between the video bitrate and the network share. The device-aware policy assigns the free space to higher resolution devices. Therefore, when using the device-aware policy, 360p resolution devices are supposed to lower there video quality sooner, compared to the bitrate fair policy. For the higher arrival rates, the 360p resolution devices are most likely already at the lowest video bitrate, and stay at the lowest bitrate during the course of a stream. As such, the number of quality switches will be lower for the higher starting rates.

Because the accuracy of the bitrate fair policy is already addressed in the previous sections, this section will evaluate the accuracy of the model given the device-aware policy. The model-based performance parameters and measurements in 12-hour experimental runs are compared. The comparison of the modeled and measured mean video bitrate for the device-aware policy is shown in Figure 4.11. Figure 4.12 shows the accuracy of the model-based quality switches rate compared to the quality switches rate achieved in experiments. Both figures show that the model in this chapter a highly accurate predictor for both the expected video bitrate and the quality switches.

4.3.6 Premium Users

The bitrate fair policy assumes equal devices, but it is was evaluated using streams with different characteristics. The device-aware policy addressed the differences

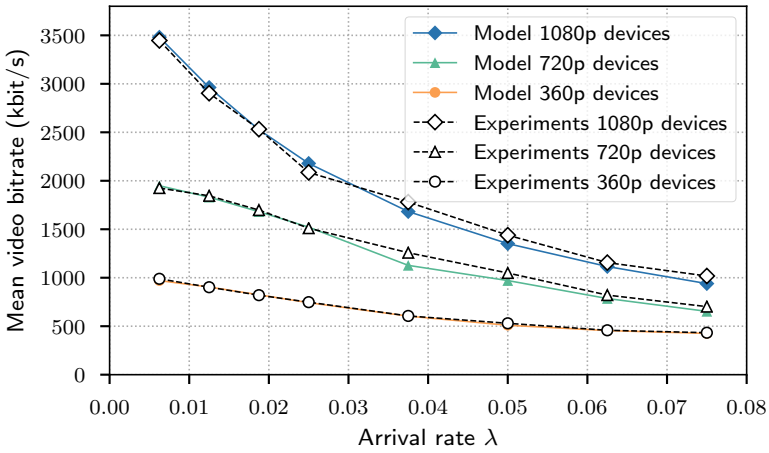


Figure 4.11: Model-based bitrate versus mean bitrate measured in experiments for the device-aware policy.

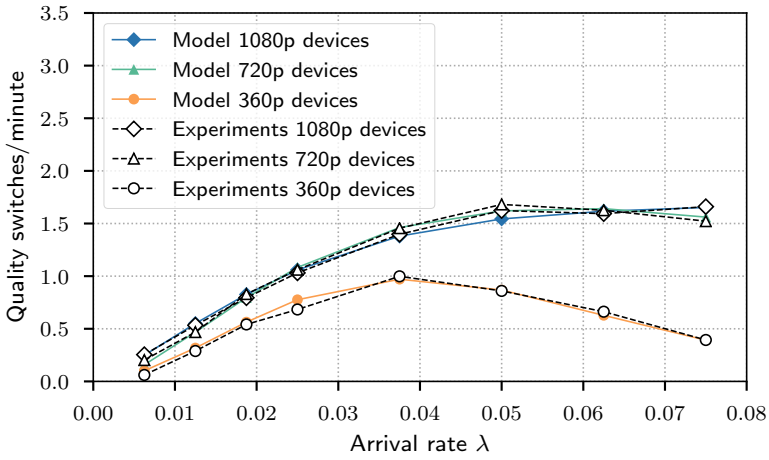


Figure 4.12: Model based quality switches versus mean quality switches rate measured in experiments for the device-aware policy.

between devices with regards to the screen resolution. Distinguishing between devices when modeling a sharing policy is done by assigning different devices to different groups. A third way to use the groups in the model is to distinguish between the type of user. This section evaluates sharing policies that include premium users. The existence of premium users may have different reasons. For example, some devices are considered more important because they are watched by multiple persons, or the stream that is watched is more important (e.g., because it is a remote lecture), or some users pay more for Internet access with the assumption that they will receive higher video quality.

Two groups of users are considered for the premium user policies: regular users and premium users. Premium users can expect a higher video bitrate when the network allows for it. For a highly loaded network, servicing all users instead of only premium users is preferred. Therefore, premium users will also be set back to the lowest video quality, similar to regular users, when the network becomes fully loaded. Two different policies are compared. The first policy is more aggressive, it gives premium users higher video quality regardless the bitrate of regular users. The second policy is more moderate, it also gives premium users higher video quality, but there will never be more than two quality levels between premium and regular users.

For both groups, the mean video stream duration is $\beta_{1,2} = 140$ seconds, and streams are available in the following bitrates: $B_{1,2} = \{400, 720, 1020, 1600, 2300, 4200\}$ Kbit/s. Regular players start according to a Poisson process with arrival rates varied between $\lambda_1 = 0.0025$ and $\lambda_1 = 0.0300$. Premium players start at arrival rates between $\lambda_2 = 0.00125$ and $\lambda_2 = 0.0150$. This results in a network environment where there are on average twice as many regular players active as there are premium players.

Figure 4.13 shows the model-based comparison of the two policies for premium users. The results show that the moderate policy is indeed more friendly towards regular players. The difference in mean video bitrate is smaller for the moderate policy compared to the difference for the aggressive policy. Figure 4.14 shows the comparison between the policies with respect to the quality switches. A similar effect as for the device-aware policy can be observed for the quality switches with the aggressive policy. Instead of low resolution devices making room in the network for high resolution devices, the aggressive policy forces regular players to make room for premium players. Regular players are the first to switch to lower video bitrates when the network becomes loaded. At higher arrival rates, it more likely that regular players are already at the lower video bitrate, resulting in less quality switches.

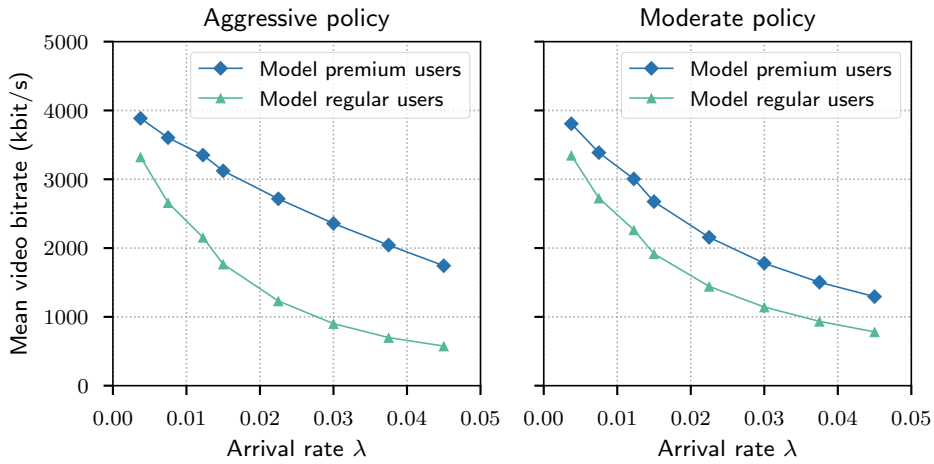


Figure 4.13: Model-based comparison for the mean video bitrate for the aggressive (left) and moderate (right) premium user policies.

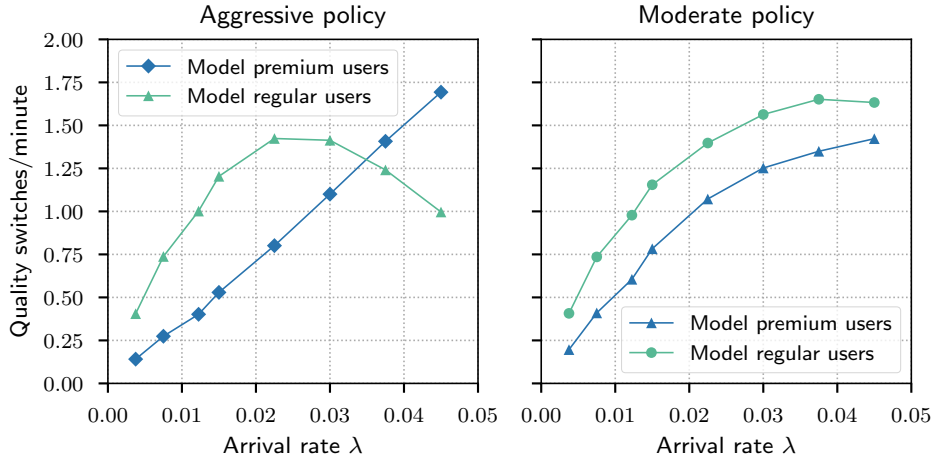


Figure 4.14: Model-based comparison for the quality switches rate for the aggressive (left) and moderate (right) premium user policies.

The aggressive policy yields the highest bitrate for premium users and is the most different compared to the bitrate fair policy. In the experimental runs, a video stream with the same characteristics as in the model-based comparison is used. Figure 4.15 shows that the modeled mean video bitrates are very close to the mean bitrates measured in the experiments. Similarly, the model-based quality switches rates show to be highly accurate when comparing them to the number of quality switches in the streaming testbed, as shown in Figure 4.16.

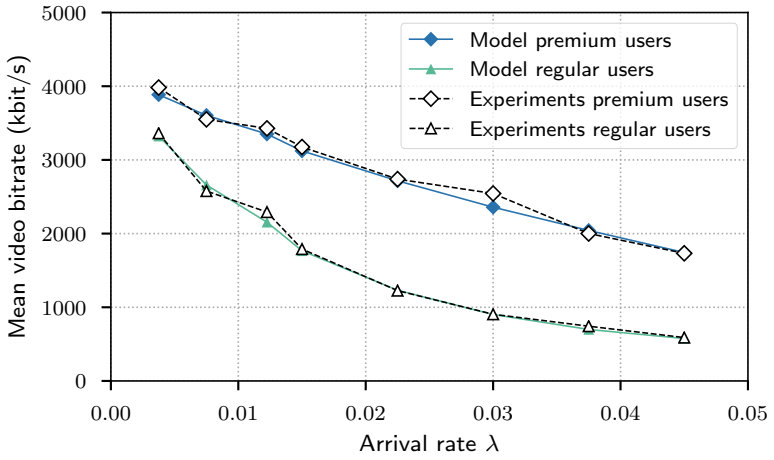


Figure 4.15: Model-based mean bitrate versus mean bitrate measured in experiments for the aggressive premium user policy.

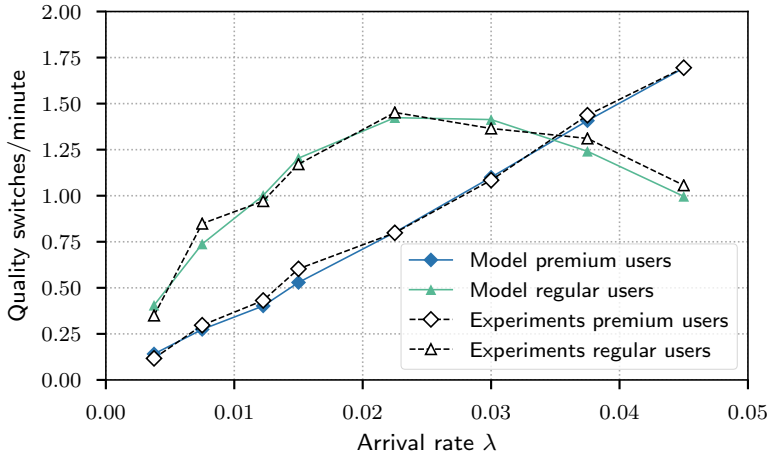


Figure 4.16: Model-based quality switches rates versus the quality switches measured in experiments for the aggressive premium user policy.

4.4 Discussion

HAS players that are capable to cooperate with the Control Element share the available bandwidth in a network based on the instructions that they receive from the Control Element. The Control Element divides the available bandwidth based on the number of HAS player, their characteristics, and the current sharing policy. In this chapter, we presented a method for analyzing the streaming performance of different sharing policies. The presented method allows the analysis of policies that classify different players (e.g., device characteristics and priority) in different groups. Experiments comparing the model-based performance estimations with the actual performance of the Control Element show that the model is accurate when estimating the streaming bitrate and number of quality switches. Using the model, we constructed and analyzed policies for devices with different forms factors (i.e., screen size), groups that play streams with different characteristics, and policies for premium users.

The advantage of the model is that it is a fast method to analyze bandwidth sharing policies for the Control Element, especially compared to the time-expensive equivalent of experimental setups. When limiting the number of different groups in the model to three and the maximum number of players to 500, the model provides estimations of the mean bitrate and number of quality switches in the order of seconds, while using a standard consumer laptop.

Scalability of the model is mostly influenced by the number of groups. Because the states in the model are specified as a vector of the number of players in each group, with the number of groups determining the length of the vector, the size of the state space has an upper bound of the number of players to the power of the number of groups. Adding an extra group will exponentially increase the size of the state space.

Computing the expected streaming bitrate consist of quick operations and scales easily. For estimating the number of quality switches, the expensive uniformization step is required, limiting the number of HAS players and groups. Nevertheless, the model shows a large operating space with reasonable boundaries, as demonstrated using the evaluations in this chapter. Furthermore, the model-based evaluations reveal that sharing policies do effect the number of quality switches, but also show the number of switches does not exceed three switches per minute in the evaluated scenarios. This means that the number of switches is low enough not to significantly influence the streaming experience [120]. In most networks, estimating the streaming bitrate will be sufficient, leaving the analysis of quality switches to situations of doubt.

Besides the number of groups, the complexity of the sharing policy also affects the computing time. The sharing policy has to be applied for each state in the model. However, it is important to realize that only sharing policies that can applied in real-time by the Control Element are interesting for analysis. The Control Element needs to be able to apply the sharing policy at each change in the network. Reacting too slow will limit the effectiveness of the Control Element. Therefore, we do not foresee that complex sharing policies will be the limiting factor when applying the method presented in this chapter.

Given the flexibility and high accuracy of the model, as demonstrated in multiple validation experiments, we deem the model a valuable complement to experimental evaluations.

5 | Optimizing HAS in Mobile Networks

This chapter focusses on the use of the Control Element in mobile networks. We look into the question if having centralized knowledge of both network and video players can be leveraged for more efficient delivery of video content. In mobile networks, the channel quality from base station to client determines the transmission efficiency. We leverage this aspect of mobile networks in a strategy that grows the video buffer when network transmissions are efficient, and shrinks the video buffer when network efficiency is low. The Control Element provides the interface between the HAS players and the mobile network that allows for optimal scheduling of HAS segment downloads and efficient bandwidth use. Simulation-based experiments show that our strategy reduces the load on the mobile network, while keeping the quality of the video stream constant.

This chapter is based on publications [K1][K2]

5.1 Introduction

The available bandwidth that the Control Element divides is constrained by the total capacity of a network connection. In wired networks, the capacity often stays constant over time. Up to a large extend, this also holds for Wi-Fi networks, especially when the Wi-Fi network is not the bottleneck network link. In contrast to those "fixed" networks, the network capacity of mobile (cellular) networks is strongly dependent on the quality of the radio signal. When the channel quality of the mobile signal gets lower, the effective throughput decreases, because a modulation scheme with more redundancy will be used. The channel quality in mobile networks heavily fluctuates, potentially causing high variations in throughput.

Because of the variations in the effective throughput, assigning a fixed share of network resources – so-called resource blocks – does not guarantee a certain throughput, nor that the throughput remains constant over time. This implies that the Control Element cannot guarantee the desired video quality and reduce switches in the same way as it would for fixed networks. To be able to divide network resources on the semantic level of desired video quality, the cost (in resource blocks) for a given video quality first has to be estimated.

In addition, taking a naive approach of allocating resources to match the target bitrate at all times is costly and inefficient. When a mobile client is in a low channel quality zone, the costs of maintaining a certain video bitrate are excessively high. The total resources required for delivering a video stream can be lowered when most of the stream is downloaded in good channel quality zones, and the HAS player relies on the video buffer in low channel quality zones. To ensure video playback while lowering the costs in the network, predictions on channel quality are needed, and the HAS player has to manage its video buffer accordingly.

To be able to predict the channel quality, the evolution of channel quality over time first has to be understood. An extensive set of LTE network traces would be of great value. To the best of our knowledge, such an extensive dataset does not exist. Therefore, this chapter starts with the collection of channel quality traces from LTE networks using an Android application. The remainder of the chapter focusses on the formulation of a Markov model based on the channel quality traces. This model is then used to formulate a buffering strategy for which the parameters are optimized using a Markov Decision Process (MDP). As such, the contributions in this chapter answer the fourth research question in this thesis:

Research question 4: *How can modeling be used in the Control Element to achieve a better streaming experience and more efficient delivery of video streams in mobile networks?*

5.2 Collection of Channel Quality Traces

The buffering strategy that is presented in this chapter targets peaks and dips in LTE channel quality. Downloading video into the buffer when channel quality is high, and playing video from the buffer when channel quality is low, can relieve load from the network. To optimize this strategy and select the optimal match between channel quality and buffer fill rate, we rely on a model that predicts LTE channel quality. The model is constructed from LTE channel quality traces that we collected using a smartphone. This section describes the application for collecting the dataset and gives an overview of the properties of the dataset.

5.2.1 Android Application for Measuring LTE Channel Quality

A smartphone application was developed for the Android platform. The application targets Android version 8.1, and uses the Telephony API from SDK version 26 [41]. The Telephony API from Android provides access to low-level LTE channel quality parameters, such as Reference Signal Received Power (RSRP) and Reference Signal Received Quality (RSRQ). Time-series of those parameters provide insights into the evolution of the channel quality between eNodeB and User Equipment (UE). Channel quality measurements were stored in comma-separated files either locally on the smartphone, or remotely on a server.

Each trace was annotated via the user interface of the application. The user interface provided spinner menus for specifying the area, environment, and activity of the users. For the area menu, users could select the following values: *'I am (almost) alone'*, *'There are a few people around me'*, *'I am in a small crowd (city center, etc.)'*, or *'I am in a big crowd (stadion, etc.)'*.

For the environment, definitions close to [140] were used, and numbers were added for further clarification. Users could select the following environments: *'middle of nowhere'*, *'village (1.000 people or less)'*, *'town (1.000 - 20.000 people)'*, *'large town (20.000 - 100.000 people)'* ,*'city (100.000 - 300.000 people)'*, *'large city (300.000 - 1.000.000 people)'*, or *'metropolis (more than 1.000.000 people)'*.

For a better estimation of moving speed and patters, users were asked to select one of the following activities: *'sitting/standing'*, *'walking'*, *'running'*, *'biking'*, *'biking (fast)'*, *'in a car'*, *'in a bus'*, *'in a train'*, or *'on a plane'*.

The area, environment, and activity fields were mandatory. An optional free textfield was provided for further notes. In addition to the user-provided meta-data, the application automatically determined the location, phone model, and network provider information. Table 5.1 lists all meta-data parameters that were collected.

Variable	input from	datatype/format
`recordingStart`	app	time (mm:ss.SSS)
`recordingEnd`	app	datetime (yyyy-mm-dd hh:mm:ss.SSS)
`recordingActivity`	user	string
`recordingArea`	user	string
`recordingEnvironment`	user	string
`note`	user	string
`city`	app	string
`region`	app	string
`country`	app	string
`phoneModel`	app	string
`networkProvider`	app	string

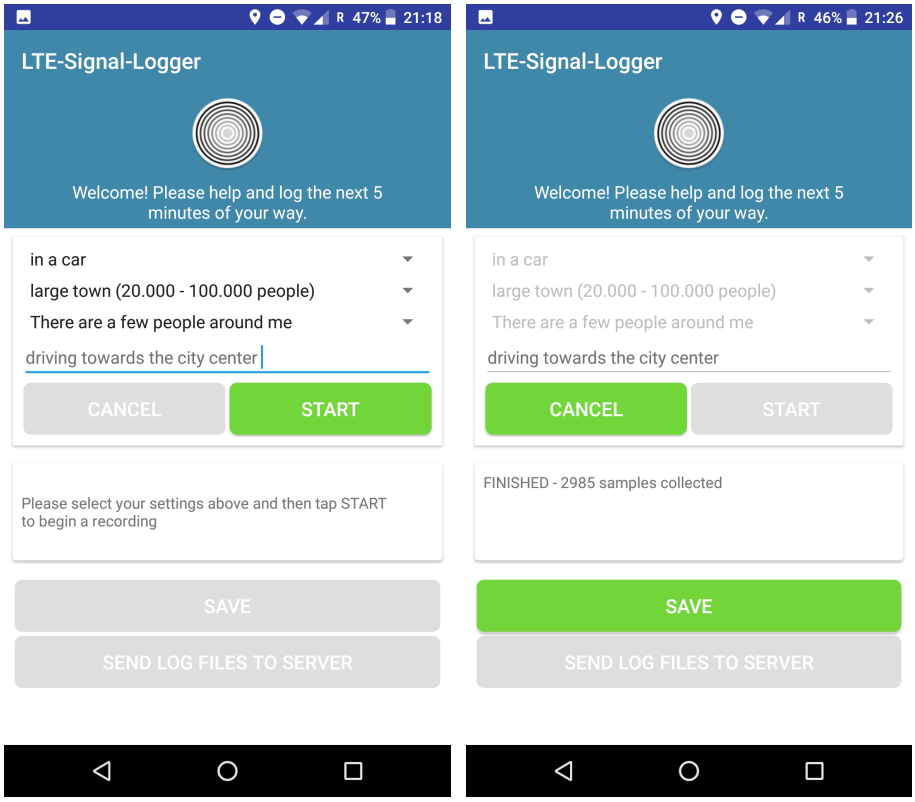
Table 5.1: Header data structure in a trace file.

Once all meta-data was provided by the user, the START button became active, as shown in Figure 5.1a. Using this button, users could start a five-minute recording of LTE network data. Although the data recorder was implemented as Android service [10], the screen had to be on during a recording. Android switches to power saving mode when the screen is turned off. Power saving mode significantly lowers the update interval for parameters from the Telephone API. For the phones that were used during the recordings, the automatic sleep timer was disabled, ensuring that the screen remained on during the recordings.

After completing a recording, users could save the trace file to local storage, as shown in Figure 5.1b. In addition, trace files could be uploaded to a server. A server with a simple HTTP endpoint was used to collect single traces or batches of multiple traces that were stored on the phone.

5.2.2 LTE Network Data Recording

LTE channel quality was recorded using five-minute measurements. Table 5.2 lists the parameters and statistics that were collected. Depending on the network chipset in the smartphone, some additional parameters from the Telephony-API could be recorded. For this dataset, two Google/LG Nexus 5X [80] smartphones were used. The Nexus 5X's chipset does not provide the *band*, *rssnr*, *rssi*, *nid*, *cqi*, *lcid*, *ta*, *snr*, and *signalStrength* parameters. The software, which is provided as open-source in [90], will record the additional parameters when used with suitable hardware.



(a) **(b)**

Figure 5.1: Screenshots of the Android application to record LTE network traces. Providing meta-data (left) and saving the trace to local storage (right).

In total, 546 traces were collected. From those traces, 377 traces contained purely LTE network data. In other traces, the smartphones switched back to 3G/2G networks due to a too low LTE signal. For the strategy in this chapter, the focus is on the 377 LTE network traces, given that LTE is a minimum requirement for high quality video streaming. Older generation networks do not provide the necessary throughput.

Most of the network traces were collected in The Netherlands (120 traces), Germany (205 traces), and in California, United States (50 traces). 4G/LTE SIM cards from the following cellular carriers were used: Vodafone (Netherlands), T-Mobile (Germany), and T-Mobile (USA). An overview of countries and network providers is given in Table 5.3.

Variable	datatype	min	max	mean	median	sd
row	int	1	2993	2986.523	2987	1.8013
currentTime	datetime	format: YYYY-MM-DD HH:MM:SS.mmm				
longitude	double	-122.4154	13.46374	-7.6503	12.05898	44.8977
latitude	double	37.34431	52.55427	48.4865	48.91395	4.6065
speed	double	0	44.64	13.9471	10.37255	13.1583
networkType	string	networkType ∈ {HSPA, HSPAP, LTE, UNKNOWN, NONE, EDGE, EDGE, UMTS, GPRS}				
mcc	int	see Table 5.3				
mnc	int	see Table 5.3				
asu	int	0	88	44.8105	44	12.2644
power	double	4.940656e-324	1e-08	0	1e-12	0
dbm	dbm	-140	-52	-95.1895	-96	12.2644
level	int	1	4	3.2363	3	0.8775
rsrq	int	-20	-3	-9.2718	-9	2.9777
rsrp	int	-140	-52	-95.1895	-96	12.2644
tac	int	265	65535	21295.4	22006	14994.07
enb	int	60	16777215	106827	100125	450037.8
eci	int	-1	230424321	24273109	25632011	15055960
pci	int	0	501	246.3232	250	134.0798
earfcn	int	1025	6400	3716.491	2300	2291.609

Table 5.2: Data structure of a trace file.

MCC	country	MNC	network provider	#samples	#rec.
204	NL	4	Vodafone Libertel	143278	48
204	NL	16	T-Mobile B.V.	214764	72
222	IT	10	Vodafone	5970	2
262	DE	1	T-mobile/Telekom	411755	138
262	DE	2	Vodafone D2	200127	67
310	US	260	T-Mobile	149218	50

Table 5.3: Traces by country (MMC) and network provider (MNC).

Figure 5.2 shows a summary of the data set by area, activity, and environment. To give a better overview, traces with a low number of samples are not displayed and related factors are shown as one group (e.g. village and middle of nowhere are summarized as one group called rural). Most of the traces are recorded in a car or while walking. Most environments have traces while standing still or sitting. Only a few traces were collected in a bus.

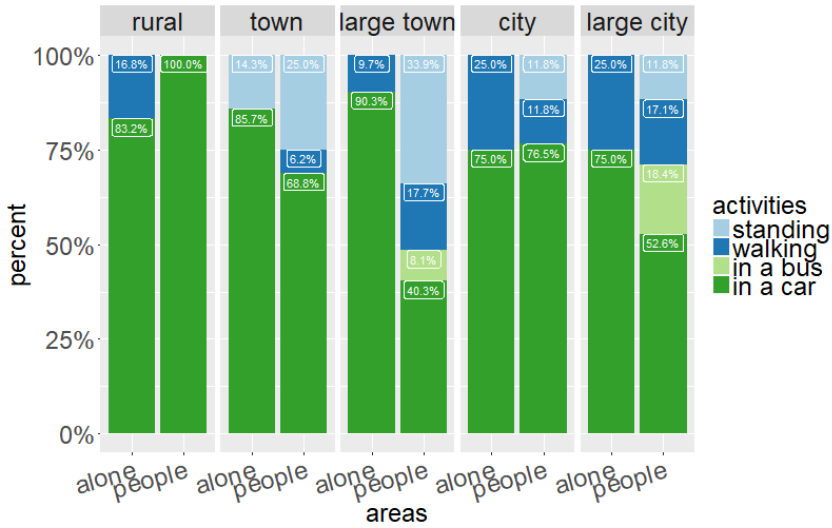


Figure 5.2: Overview of traces by area, environment, and activity.

Although the dataset contains a large number of variables, the remainder of this chapter focusses mainly on the RSRQ. RSRQ provides an indication of the channel conditions between base station and client. Looking at the distribution of RSRQ values, it can be noticed in Figure 5.3a that the data is not normally distributed. There is a slight negative skew, meaning that the channel quality is better than

average (-3 indicates excellent channel conditions, -20 indicates poor quality). Figures 5.3b, 5.4a, and 5.4b show box-plots with the lower and upper quartile at 25% and 75%. These figures show that there are only slight differences between the factors in different groupings.

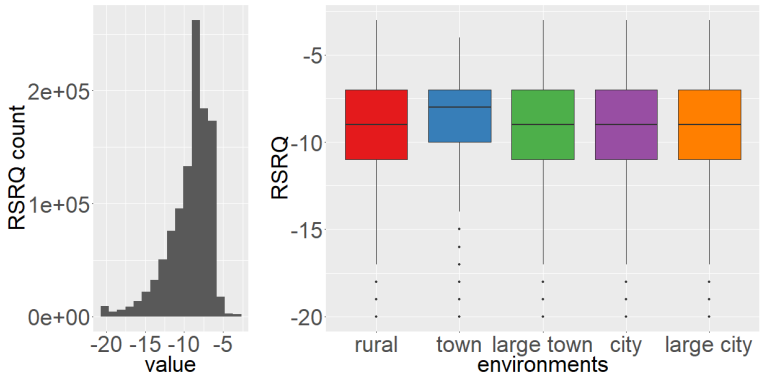


Figure 5.3: Distribution of RSRQ values. All RSRQ values (left) and RSRQ values per environment (right).

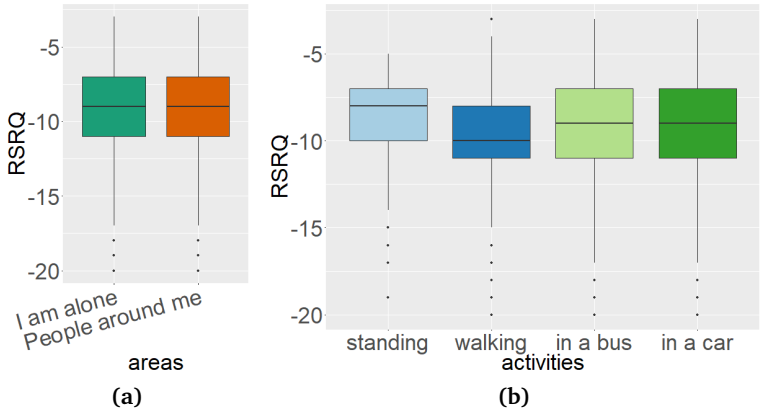


Figure 5.4: Distribution of RSRQ values. RSRQ values per area (left) and per activity (right).

Figure 5.5 shows the speed distributions in the traces. The speeds in buses are comparably low, where the low quartile is at zero because of frequent stops at bus stops. The median for driving in a car is around 80 km/h. The distribution of speeds shows some outliers. These outliers are most likely the result in inaccuracies of the GPS measurements with the smartphone.

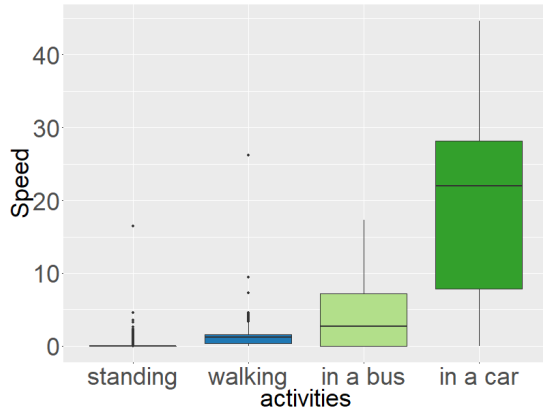


Figure 5.5: Distribution of the traces by speed (in m/s) and environment.

5.3 Modeling Channel Quality Behavior

Data transmission efficiency in mobile networks depends on the channel quality. The strategy that is presented in this chapter anticipates on this by downloading more video content into the buffer when efficiency is high, and play video from the buffer when efficiency decreases. To determine when to start growing/consuming the video buffer, the evolution of channel quality has to be understood. Accordingly, the following two steps are taken:

A) fit a Markov chain on the collected traces and determine data transmission efficiency for each state;

B) predict data transmission efficiency for future video segments.

5.3.1 Modeling Channel Quality and Data Rate Behavior

Through analysis of the channel quality traces it is obtained how RSRQ evolves over time. Based on this information, a Markov chain is constructed to fit this process. The state space $\mathcal{S}$ is defined as eighteen states representing the RSRQ reporting range in steps of one decibel. Transitions in the Markov chain express changes in RSRQ. We observed in our traces that RSRQ can change between any two levels. Therefore, a fully connected Markov chain is used. Nevertheless, small changes in RSRQ are more probable. The transition probability matrix P is obtained through a straightforward translation from the number of channel quality changes we have observed in the traces to the transition probabilities. We opt for a transition interval of 1/3 second. The two-second intervals from the channel quality traces are too coarse to express the download process of DASH video segments at different

buffering speeds. A too small transition interval would cause long execution times when deriving the strategy by solving the MDP that is described in the next section.

For each state in the Markov chain, the data transmission efficiency that corresponds with the RSRQ is computed. A process that is similar to how the modulation settings (which determine the efficiency) in LTE networks are established is followed. In the first step, RSRQ is mapped to Signal to Noise and Interference Ratio (SINR). A theoretical mapping between RSRQ and SINR exists, and is approximated using the Equation 5.1 [55, 65].

$$SINR = \frac{1}{\frac{1}{12-RSRQ} - \rho} \tag{5.1}$$

The load in the serving cell is denoted by ρ. We use $\rho = 1/6$, which indicates a lightly loaded cell. Given the SINR, a lookup table to obtain the Channel Quality Indicator (CQI) and corresponding data rate is used. The data rate is the fraction of bits in a Resource Block (RB) that remains after demodulation (i.e. the actual data). Table 5.4 lists the mapping from RSRQ to the data rate. The function $d(x)$ applies this mapping and provides the effective data rate for state x.

5.3.2 Predicting Transmission Efficiency

Given the RSRQ Markov chain and the mapping to effective data rate, predictions can be made on how the data rate will evolve. Those predictions will be used to estimate how many resources will be required to download the next video segment at a certain speed (i.e. grow, maintain, or shrink the video buffer). Depending on the download speed, downloading a video segment may to take longer than one interval in the model. The multi-step transition probabilities are computed to determine how RSRQ and data rate change while downloading this video segment. Given transition probability matrix P, the n-step transition probabilities are obtained by multiplying P. The probability that our RSRQ model transitions from state x to y in n steps is denoted as $P^n_{x,y}$.

Driven by the changes in RSRQ that can occur during one segment download, the effective data rate may also vary. To get the expected network resources needed for one video segment the data rate during that download has to be estimated. Therefore, we average the data rates for each possible path between two states, weighing each path by its probability of taking it.

RSRQ (dB)	SINR (dB)	CQI	Eff. data rate
-20	-9.12	0	0.000
-19	-8.10	0	0.000
-18	-7.07	0	0.000
-17	-6.03	1	0.026
-16	-4.98	1	0.026
-15	-3.92	2	0.039
-14	-2.85	2	0.039
-13	-1.75	3	0.063
-12	-0.06	3	0.063
-11	0.54	4	0.101
-10	1.76	4	0.101
-9	3.05	5	0.147
-8	4.45	6	0.198
-7	6.00	6	0.198
-6	7.82	7	0.248
-5	10.13	9	0.404
-4	13.70	10	0.459
-3	INF	15	0.930

Table 5.4: Mapping RSRQ to effective data rate.

We compute the expected data rate when transitioning from state x to y in n steps using Equation 5.2.

$$D^n_{x,y} = \frac{1}{n}\sum_{i=1}^{n}\sum_{z\in\mathcal{S}}\frac{\lceil P^{n-i}_{z,y}\rceil P^i_{x,z}}{\sum_{w\in\mathcal{S}}\lceil P^{n-i}_{w,y}\rceil P^i_{x,w}}d(z). \tag{5.2}$$

For each of the n steps, we compute the expected data rate in that step, which is the average data rate of all states, weighted by the probabilities that a route from x to y traverses a state in the given step. However, not all paths provide a viable route $x \to y$, especially when the number of steps becomes small. Therefore, transitions that after a transition have enough steps left to reach destination state y are considered. For each sub-state z in Equation (5.2), we only include z when the probability of transitioning $z \to y$ in $n-i$ steps is non-zero. Since not all states are included, the weights are adjusted accordingly.

5.4 Channel Quality Based Buffer Strategy

The number of network resources that are required for downloading one segment can be estimated with $D^n_{x,y}$ and the current channel conditions. How many intervals are needed to download one segment depends on how fast it should be downloaded. Assuming a segment with a duration of two seconds, downloading it in two seconds would maintain the buffer level. Downloading faster increases the buffer, while downloading slower than two seconds (or not at all) would shrink the buffer. We define six different buffering actions (denoted as *Bxxx*) for our Channel Quality based Buffering Strategy (CQBS):

- **Growing the buffer**: Downloading video segments two (*B200*) or three (*B300*) times faster than playing segments out.
- **Maintaining the buffer**: Download one video segment takes the same time as playing one segment (*B100*). This action will be chosen over *B200/300* when the buffer reaches its maximum level.
- **Shrinking the buffer**: Relieving the network when bandwidth becomes expensive. We download video segments either at $\frac{1}{3}$ (*B033*) or $\frac{1}{2}$ (*B050*) of the playback speed.
- **Not downloading**: Data transmissions are sometimes not possible because channel quality it too low. During those periods, no video segments will be downloaded and no resources are required (*B000*).

Depending on the buffering action, downloading one two-second segment will take between two and eighteen intervals of 1/3 second. The channel quality of the client and the current buffer fill level determine the best buffer action at each time. This problem, which is the core of CQBS, is formulated as an MDP. Let $(\mathcal{S}', A, P', R, \gamma)$ be our MDP. The state space $\mathcal{S}'$, describes the state of a single video player instance. A state is defined as a couple (c, b) combining the current channel quality (indicated by RSRQ) with the current buffer level. Buffer fill levels are modeled as the number of video units with a duration 1/3 second, aligned with the transitions intervals in our Markov chain. As such, one video segment of two seconds consists of six video units. The buffer is limited to 60 seconds.

The set of actions A covers the six buffering actions, *B000* to *B300*. Based on the buffering action, and the download time of one segment given that action, certain transitions within $\mathcal{S}'$ are possible. For example, downloading a two-second segment at speed *B200* takes three steps. While downloading six video units, the video player plays out three units. The buffer grows with three units. For buffering action *B200*, transitions are possible to all states in $\mathcal{S}'$, such that $b \rightarrow b + 3$. The probabilities for these transitions are obtained from the 3-step transition probability

matrix of the RSRQ Markov chain, $P^3_{x,y}$. Following this principle, the transition matrix P' is build for each combination of RSRQ and buffer level.

The rewards for each transition, R, are in general linked to the expected cost in the network. For each action and transition, the expected load on the network is computed as the percentage of resource blocks required by the video player. Equation 5.3 defines the calculation of the expected load. In Equation 5.3, B is the video bitrate, $T_{segment}$ the duration of a video segment in seconds, Rbs the number of resource blocks that is available per second, and n the number of steps that the segment download takes.

$$ld(x,y,n) = \frac{B \cdot 3T_{\text{segment}}}{D^n_{x,y}} \cdot \frac{1}{\frac{1}{3}Rbs \cdot n} \tag{5.3}$$

Whether the load is used in the reward depends on the following three step process:

1. Check if the buffering action is possible. Only actions that don't cause an overload on the system (an overload being $ld(x,y,n) > 1$) are accepted. Also the *B000* action is restricted to only being used when channel quality is too low for data transmission. A high negative reward (-10000) is used to ensure invalid actions.
2. Check if the buffering action would lead to an empty buffer. In case the buffer becomes empty, a penalty of -50 times the duration of the freeze (in ticks) is assigned as a reward for the transitions.
3. When 1) and 2) don't apply, the inverse load $(1 - ld(x,y,n))$ is used as reward.

The MDP is solved using the value iteration algorithm with discount factor $\gamma = 0.999$ from the Python-based mdptoolbox [34]. The resulting policy is the optimal buffering strategy. For each channel quality and buffer level the strategy provides the buffering action (*B000* to *B300*) to take. A visualization of our strategy is shown in Figure 5.6. The lighter the color is in the visualization the faster should be downloaded. The strategy reflects the intuition behind CQBS. When the network resources become expensive, the buffering actions that shrink the buffer are selected. Only when the buffer level is low, the buffer has to grow, or has to be maintained as a minimum. As such, the strategy avoids the penalty for an empty buffer. When network traffic becomes cheaper, the strategy lets the buffer grow. The maximum fill level of the buffer grows with the channel quality. This means that the strategy stops filling the buffer at some point. Only when the channel quality further increases the strategy continues to grow the buffer.

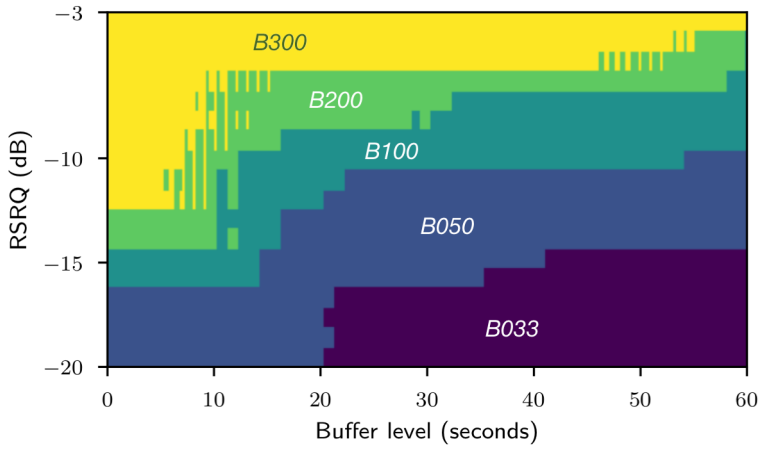


Figure 5.6: Buffering strategy for constant video quality (1458 Kbit/s), best buffering action for each channel quality and buffer level.

Figure 5.7 shows an example streaming session with the channel quality and the buffer level overlaid in the same plot. Initially, CQBS grows the buffer when channel quality is good. Around, $t = 60$, the signal quality decreases and more video from the buffer is used. The buffer level is restored when signal quality restores. Around $t = 350$, the signal quality becomes excellent. This triggers CQBS to fill the buffer to a higher level than it did before.

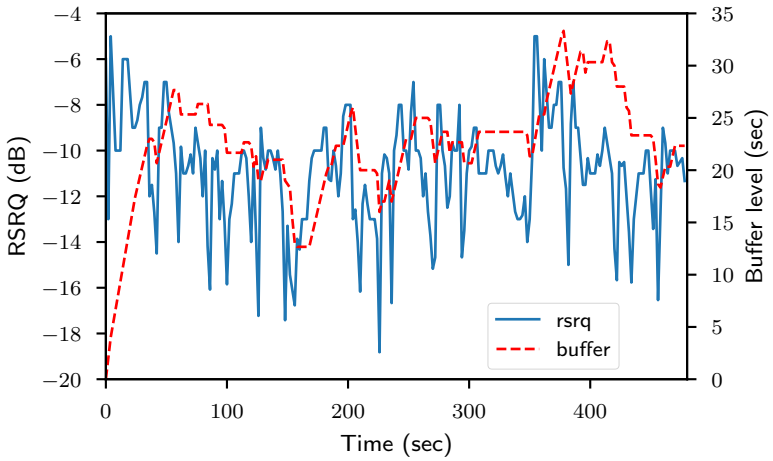


Figure 5.7: Example operation of CQBS: interaction between channel quality and buffer level.

5.5 Performance Evaluation

In this section, the simulation results that compare CQBS to two traditional adaptation algorithms and a naive constant video quality strategy are presented. First, the simulation setup is described. Then, the different algorithms in simulations where channel quality behavior is retrieved from the Markov chain from Section 5.3 are compared. Last, CQBS is cross-validation against the real-word channel quality traces to access the practical performance.

5.5.1 Simulation Setup

The performance of CQBS is evaluated from the perspective of a single client (i.e. how many resources would be required to execute the strategy). We assume that the video streaming client gets the resources assigned that it needs. We use a custom discrete event simulator to simulate an LTE network and a video streaming application. For simplicity, the simulation covers a single video node that is associated to one LTE base station. The LTE base station is configured with 15 MHz bandwidth and has 75,000 RBs available per second. In this section, the network load is denoted as the fraction of those RBs that are used by the video streaming application. The channel quality (and its changes) between the LTE base station and mobile client are either based on the RSRQ Markov chain from Section 5.3 or are obtained from the real-world channel quality traces.

The video player that runs on the client node simulates the download of a DASH video stream. A video clip is taken from the movie Sintel [123] and encoded in ten representations listed in Table 5.5. The video is segmented for DASH with a segment size of two seconds. The video player informs the LTE base station about it its current buffer level and channel quality. The player signals the base station every time before downloading the new video segment. The base station executes our CQBS strategy by combining the buffer level with the current channel quality, selecting the best buffer action and allocating appropriate resources.

For evaluation, we compare four different adaptation and buffering strategies:

1. **Conventional:** The conventional algorithms uses the download speed of previous video segments to estimate future bandwidth. The video quality will be the highest bitrate that is below this estimation.
2. **BOLA:** An implementation of the Buffer Occupancy-based Lyapunov Algorithm from Spiteri et al. [126]. We use a version that reduces video quality oscillations, comparable to the reference implementation in the DASH.js player [37].

Quality level	**1**	**2**	**3**	**4**	**5**
Bitrate (kbit/s)	296	395	493	732	971
Resolution	240p	240p	360p	360p	480p
Quality level	**6**	**7**	**8**	**9**	**10**
Bitrate (kbit/s)	1.458	1.934	2.878	3.779	5.544
Resolution	480p	720p	720p	1080p	1080p

Table 5.5: Video bitrates used in the simulations.

3. **Static:** A naive implementation that produces static video quality. Until the buffer is full, resources for twice the video bitrate are allocated for quickly growing the buffer and preventing freezes. When the buffer is full, resources matching the video bitrate are allocated. When the video bitrate exceeds the networks capacity (e.g. when signal quality is very low), the maximum amount of resources are allocated.
4. **CQBS:** Adjusts its buffer based on the current channel quality and buffer level. Resources between one third- and three times the video bitrate are allocated, depending on the selected buffering action. When the video buffer drops below four seconds (i.e. two video segments), and the channel quality does not permit streaming in the target quality, the video player temporarily lowers its video quality, as part of a failover mechanism to ensure uninterrupted playback.

We want to compare the four algorithms for different video quality levels. The conventional algorithm and BOLA always try to get the highest possible video quality, for which they require many network resources. To create a fair comparison between the algorithms – where fair means that the average video quality is the same for each algorithm – the network resources for the conventional algorithm and BOLA are capped as specified in Table 5.6. The resource limits are specified as the maximum fraction of RBs that can be used by a video streaming client.

5.5.2 Performance Comparison Based on the Channel Quality Behavior Model

In the first part of the evaluation, channel quality behavior is simulated based on random walks in the RSRQ Markov chain from Section 5.3. The CQBS strategy is based on the same Markov model, and thus it is based on accurate probabilities how RSRQ can change. As such, CQBS will perform optimal, allowing us to check the concept behind our strategy. Per setting, 25.000 channel quality traces are

Target video quality level	**3**	**4**	**5**	**6**
Conventional adaptation limit	0.063	0.095	0.140	0.200
BOLA limit	0.045	0.072	0.110	0.156
Target video quality level	**7**	**8**	**9**	**10**
Conventional adaptation limit	0.280	0.400	0.650	1.00
BOLA limit	0.216	0.305	0.435	1.00

Table 5.6: Network resource limits as a fraction of the total available RBs.

generated. For each trace, a 10-minute HAS streaming session is simulated, which we will call an instance. For each instance, we measure the video quality (levels ranging from 1 to 10) and load on the network. A video quality level is used instead of the video bitrate. The encoding settings of our video are chosen such that every step gives a comparable increase in quality. However, the bitrate that is needed for each step increases with the video quality. Averaging over video bitrate would give skewed results.

The evaluation starts by looking into the differences in video quality between the four algorithms. The target video quality is level five. For each instance the mean video quality is computed. The distribution of mean video qualities is shown in Figure 5.8. The boxes in Figure 5.8 indicate the four quartiles. The red line in the box is the median. For the static and CQBS strategies, Figure 5.8 reveals that the overall video quality matches the target quality in almost all instances. The exceptional instances suffered from either low channel quality at the startup or long periods of low channel quality. As result, the fallback mechanism temporarily lowered the video quality. For the conventional and BOLA algorithms the mean video quality lies within one and a half video quality levels around the target.

The video quality within one instance is also not constant for the conventional and BOLA algorithms. These adaptation algorithms adapt the video quality to match the network conditions. Because the network conditions change (as a result of the channel condition chaning), the video quality changes as well. The cumulative distribution of video quality within instances is shown in Figure 5.9. The figure shows that the video qualities are spread out over all levels, as can be seen by comparing the dashed lines to the solid lines in Figure 5.9.

When using the conventional and BOLA algorithm, the video is streamed in the target quality only a relatively small fraction of the time. The broad spread of video qualities during each run indicates that the video quality changes from time to time.

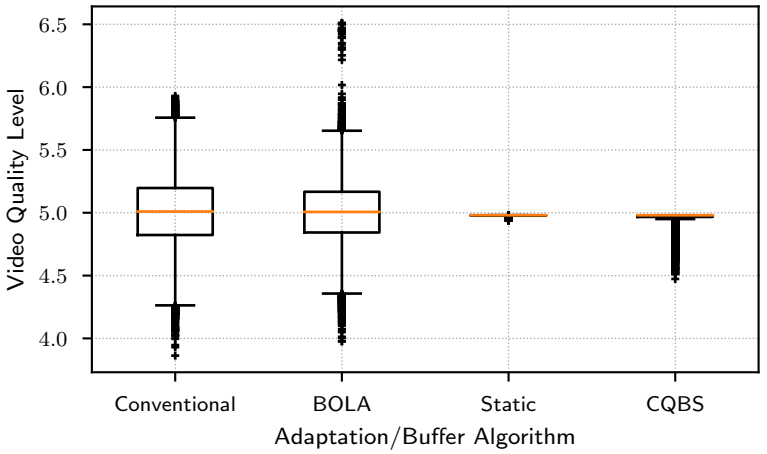


Figure 5.8: Distribution of mean video quality per instance, target video quality level five.

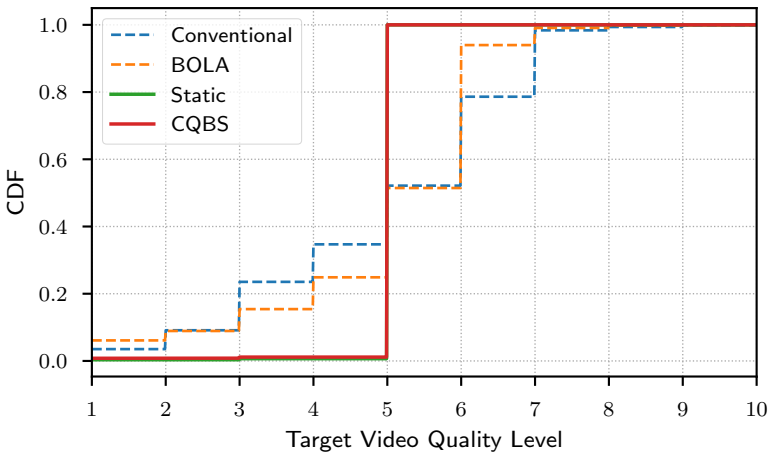


Figure 5.9: Cumulative distribution of video quality within the instances, target video quality level five.

In Figure 5.10, we observe that both the conventional algorithm and BOLA show many video quality switches. The conventional algorithm performs worst, resulting in more than 12 quality switches per minute for most of the target quality levels. BOLA perform better, but the switching frequency is still high. A big difference can be observed from target quality level 10. In this case, all algorithms had the full spectrum available for streaming. In the case of BOLA, it meant that it could

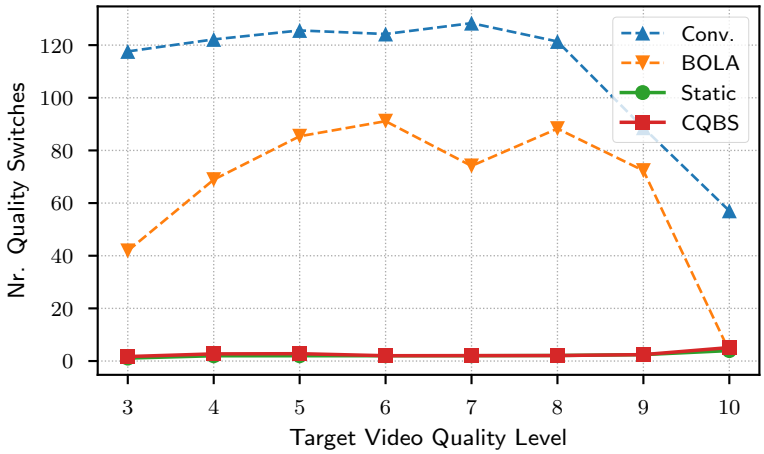


Figure 5.10: Number of quality switches as a function of video quality.

maintain large buffers. Due to the nature of BOLA, an almost full buffer will result in video segments of the highest quality. Even when the signal quality decreased, the buffer levels were still sufficient for BOLA to request video quality level 10. As the result, BOLA worked similar (except during the startup phase) to the naive static strategy. This resulted in little quality switches, but to an increased network load as we will discuss next in this section.

The high number of quality switches when using the traditional algorithms may be perceived by the user as annoying. In comparison, the naive static quality strategy and CQBS maintain an almost perfectly constant video quality. Although providing a constant video quality may be more beneficial for the end-user, it could increase cost for the network operator. Figure 5.11 shows the distribution of network load (percentage of RBs that were used for video streaming) per instance. For most of the instances, the network load for the conventional algorithm and BOLA is comparable. Nevertheless, BOLA shows some instances that have relatively high load. Comparing the medians of the traditional algorithms with the naive static quality strategy, we observe and increase in network load of 25%. This increase in network load takes up resources that cannot be used by other clients in the same cell. The increase makes a difference over ten minutes, from up to 2 Mbit/s bandwidth for clients with good signal quality, to about 330 Kbit/s for an average client. This bandwidth cannot be used anymore for several Web browsers, a high quality music stream, or a low quality video stream.

The CQBS strategy is designed to reduce the load on the network. It consumes video from the buffer when network bandwidth is expensive. When channel quality

becomes better and bandwidth comparatively cheap, it re-fills the video buffer. Figure 5.11 shows the impact of this strategy on the network load. On average, the use of network resources by CQBS is on the same level as the traditional adaptation algorithms. In 23% to 64% of the cases, CQBS requires less resources than the most efficient instance from the conventional algorithm and BOLA.

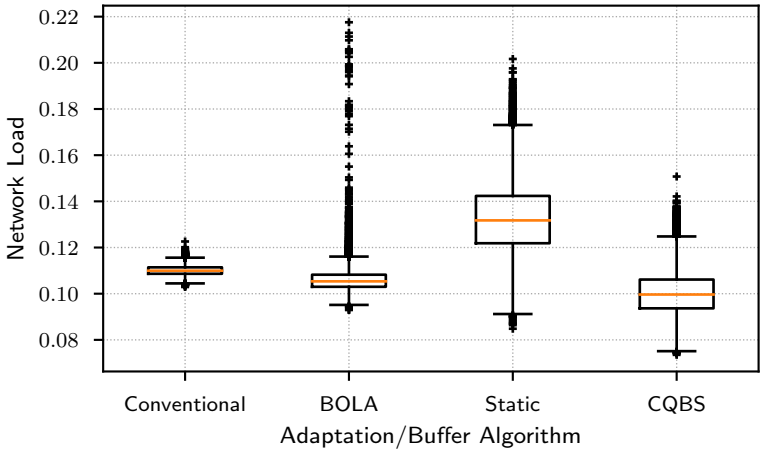


Figure 5.11: Distribution of overall network load per instance, target quality level 5.

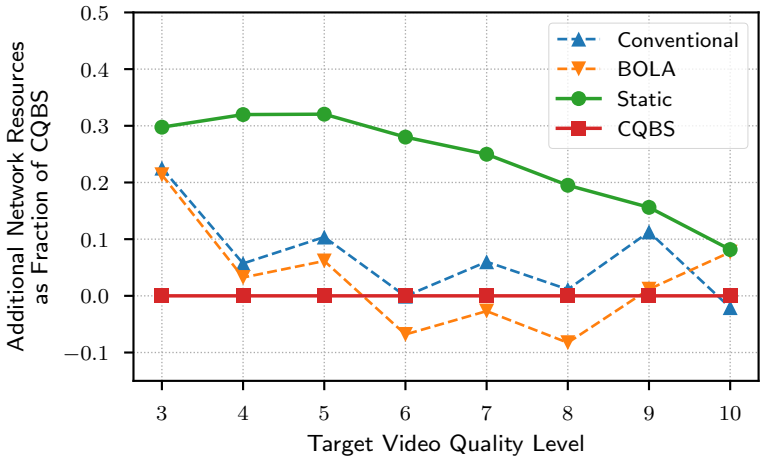


Figure 5.12: Additional usage of network resources relative to the resources used by CQBS.

Figure 5.12 shows the difference in network usage per video quality level for the four different implementations. The differences are shown relative to the network resources used by CQBS. CQBS significantly reduces the network load compared to non-optimized static quality implementation. On average, the network load is 19% lower when using CQBS. When comparing CQBS to the conventional algorithm and BOLA, we see overall a similar network load for the three strategies.

For quality level 9, the conventional algorithm yields a high network load. The conventional algorithm bases the quality of future video segments on the network throughput in the near past. Past performance is thus not a good indicator for the future. In this case, the conventional algorithm overestimated the quality and had to recover the buffer afterwards. The reason for the higher load for BOLA at quality level 10, is given earlier this section.

5.5.3 Validation against Real-world Traces

In the second part of the evaluation, the CQBS algorithm is validated against the real-world LTE traces that we have collected. We use the cross-validation technique, where the CQBS strategy was trained on 80% of the traces and then tested against the remaining 20% of the LTE traces, where the fraction of traces that is used for validation rotates. This sums up to 377 instances per setting. The DASH player streams a 8:20 minutes clip from the movie Sintel using the same encoding settings as before. The LTE channel quality traces have a duration of five minutes. Therefore, a trace is repeated to cover the full length of the video.

In general, the performance of CQBS and the other algorithms is similar when using the real-world traces instead of the model-based traces. Therefore, the focus will be on the differences compared to the validation experiments above, starting with the stability of the streams' qualities. Figure 5.13 shows the distribution of mean video quality for each instance. Given target quality level 7, the constant video strategies perform well, having almost all instances at the target video quality. For the conventional algorithm and BOLA the spread of average video quality is larger. With video resolutions ranging between SD (360p) and Full-HD (1080p) the differences between instances are large, indicating that reserving a fixed portion of network resources cannot provide guarantees on the video quality.

With regards to the number of quality switches as shown in Figure 5.14, the conventional algorithm and BOLA both produce a high number of quality switches, similar to using the model-based traces (the video was slightly shorter and the number of switches is consistently lower). In contrast, the CQBS strategy produces a higher switching frequency with the real-world traces, especially for the high target qualities. In CQBS, quality switches only occur when channel quality is bad

and the buffer level is too low to guarantee uninterrupted streaming. This indicates that buffers levels get lower more often because the duration and severity of low channel quality is underestimated by our model. The model is thus not entirely accurate at low channel qualities. However, even though there is a slight increase in quality switches, CQBS still greatly outperforms the traditional algorithms.

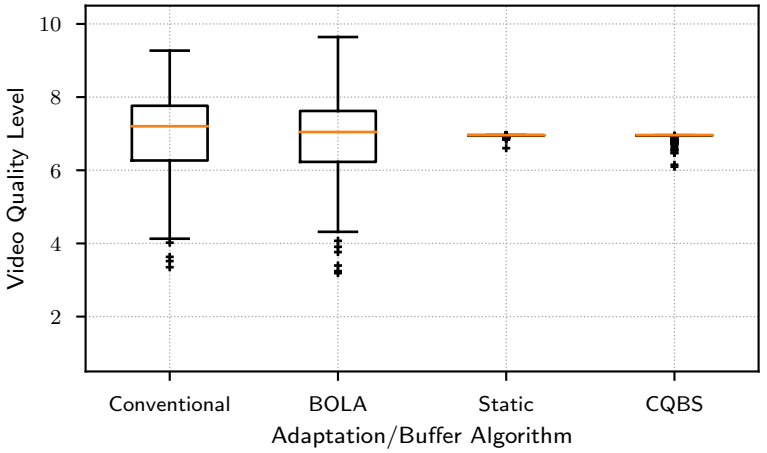


Figure 5.13: Distribution of mean video quality per instance, target video quality level seven.

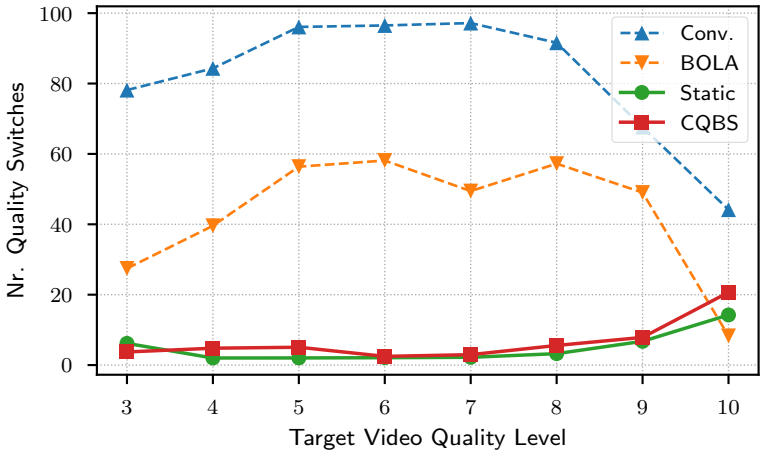


Figure 5.14: Mean number of quality switches as a function of video quality.

In Figure 5.15, the differences in network resource usage relative to CQBS for the target quality levels are shown. Compared to the model-based traces, the network loads for the real-world LTE traces show small differences. There is a slightly smaller difference in network load between the naive static strategy and CQBS. The naive static strategy seems to perform slightly more efficient given the real-world traces. Nevertheless, the CQBS strategy is able to decrease the network load by on average 15%. In terms of network load, CQBS performs slightly better than the conventional algorithm, and on average as good as BOLA.

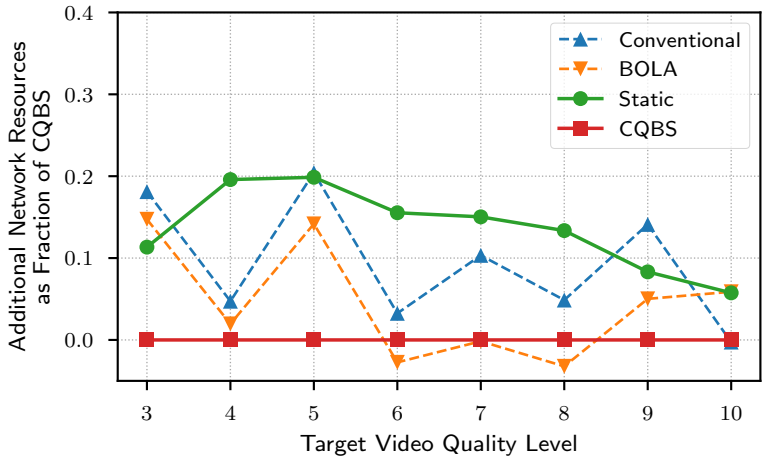


Figure 5.15: Additional usage of network resources relative to the resources used by CQBS.

Although BOLA performs overall best when it comes to network load, we did observe a high number of freezes when using BOLA. In Figure 5.16, it can be seen that BOLA has a structural problem and that freezes occur in most of the instances. In a large number of runs, the number of freezes can even be considered as very high. A high number of freezes is disastrous for the user experience, and will lead to abandonment of the video stream [43]. The other algorithms, including CQBS, do no have playback interruptions in almost all the runs. Only in exceptional cases freezes might occur. This shows that even though CQBS actively reduces the buffer level, it does not compromise the continuity of the stream.

Overall, we can conclude that CQBS can maintain a constant video quality. It outperforms the traditional adaptation algorithms that show fluctuations in video quality as a result of variations in LTE channel quality. Compared to high number of freezes in BOLA, CQBS is able to stream without interruptions in almost all runs.

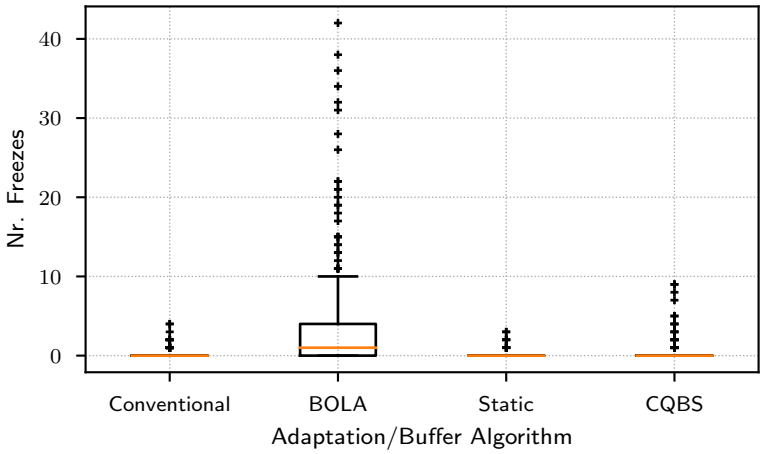


Figure 5.16: Distribution of the number of video freezes per instance, target quality level seven.

As such, CQBS provides the best experience to the user. When looking at network load, CQBS performs on par with the traditional algorithms, but outperforms the non-optimized static quality strategy.

5.6 Discussion

Mobile networks are a different environment compared to wired, or even Wi-Fi, networks. One significant difference is that mobile networks operate with a high level of management, which is required for admission into the network, charging traffic to the data plan, and to handle mobility. The advantage of such a managed network architecture is that mechanisms for fine grained QoS are available and integrated in each aspect of the network (i.e., the backhaul, core network, and radio access networks). Through QoS bearers in LTE, and QoS flows in future 5G networks, policy and charging functions in the network can provide QoS for a client or to specific flows (e.g., video streams).

However, the allocation of network resources to realize QoS is tightly linked to the channel quality between the UE and the mobile base station. Depending on the channel quality, providing QoS might be a costly operation and may even decrease the overall performance of the mobile network. In extreme cases with poor channel quality and a high network load from other users, QoS may be refused. To be able to offer QoS, and offer QoS to as many users as possible, it is important to configure QoS in such a way that it puts the smallest demand on the network as possible.

In this light, the works from Ramamurthi et al. [109] and Zahran et al. [147, 146] might be too limiting. The focus in these works is on providing a boost in network throughput to HAS players for which the buffer level is critically low. As such, they operate consistently with the "network assistance" SAND mode in the 3GPP specifications [1]. However, video buffers are typically running low when network conditions are poor (except in the situation of the start of a new stream). In mobile network, this happens when the channel quality is poor or when many users access the same network. Although providing QoS for HAS players with low buffer levels prevents those players from video freezes, and it is thus the right action to take for ensuring a good QoE, the action itself may be expensive in terms of network resources. The CQBS strategy presented in this chapter addresses this issue. CQBS not only provides a boost in network throughput when playout buffer are low, it also provides a boost when network resources are relatively cheap. Following this strategy, the risks of needing expensive throughput boosts is reduced, freeing network resources in the network to be used by other users.

To implement the CQBS strategy, the integration between HAS player, the Control Element and the network, is essential. The network needs to know the state of the player (i.e., the current buffer level and expected video quality) when configuring QoS and scheduling network resources. From the simulations, we learned that this information needs to be actual for the strategy to be effective. Therefore, the HAS players provide the information on every segment requests. Buffer estimations (e.g., using BUFFEST [77]) might be too slow or inaccurate, motivating the need for an explicit information channel between the HAS player and the Control Element.

In addition to the information flowing from the HAS player to the Control Element, feedback from the Control Element to the HAS player cannot be omitted. HAS players need to know why they receive additional QoS, and what the intension of the Control Element is for providing the extra throughput. Regular HAS players would increase the video quality, where an informed HAS player uses the extra throughput for filling the playout buffer. We argue that only handing QoS and not explicitly informing the player (e.g., through DiffServ [21] or active queue management [100, 116, 96, 54]) may trigger the wrong action in HAS players. Because video streaming is more than delivering packets at a certain rate, and QoS mechanisms also have to deal with the intelligence that is now part of the video player, providing QoS is no longer a network level action only. The Control Element operates on both the network level and application level, providing suitable support for HAS-based video streaming in mobile networks.

6 | Conclusion

With a share of around 70% of the global Internet traffic, video streaming is one of the most popular online applications [117]. Nowadays, there is a large catalogue of high quality video content available and it is continuously expanding. The availability of high video quality content for on-demand streaming has become the standard and it has shaped the expectations of the user towards the delivery of this content. Networks have to be able to deliver the content to the end-user, providing good performance and allowing multiple users to share a network connection for video streaming.

However, the delivery performance of HTTP Adaptive Streaming (HAS), the dominant streaming technology on the Internet, does not match the users' expectations. We have performed an extensive set of experiments where we covered multiple types of networks, evaluated networks with multiple players and background traffic, and studied algorithms in the reference player. Based on those experiments, we confirm the findings in the literature and show how current solutions perform sub-optimally in shared networks. The performance problems are caused by a too low throughput available to the HAS players, highly varying throughput, and mistakes made by the HAS players while estimating the throughput. The results are video freezes, lower video quality, many quality switches, and unfair sharing of network resources.

The main contribution of this thesis is a centralized control and management element in the network, the Control Element, for which we show that it significantly improves the streaming performance of HAS in shared networks. The Control Element guides HAS players when they select a video quality and it establishes traffic control to ensure fair sharing of resources. The Control Element further improves video streaming by allowing differentiation of players. Players can be treated differently, for instance, based on the type of device, type of user, or type of stream. The Control Element can assign some players more bandwidth than others when it is needed. Without the Control Element, adaptation algorithms follow the

approach that higher video quality is always better and always greedily strive to get this higher video quality. Greedy adaptation algorithms take bandwidth from other players that may need more resources. This behavior of the adaptation algorithms highlights the difference between the local optimization perspective of HAS players and the global optimization perspective that the Control Element envisions.

Local optimizations are selfish and will harm the fairness and overall performance of video players in a shared network. With the Control Element, specific sharing policies can be executed. As such, fairness and streaming performance are defined in a policy. In several experiments, we have shown that the Control Element correctly executes those policies. With the Control Element, HAS players that need more bandwidth than others will get it, and as a result, will stream in a higher video bitrate.

By specifying sharing policies that are in alignment with the user requirements, the Control Element makes HAS-based video streaming perform as expected. However, different networks and users require different sharing policies. Therefore, this thesis also focusses on the impact of sharing policies on the streaming performance. We defined four sharing policies: bitrate fairness, device characteristics aware, and two policies with premium-user support. Using accurate performance models, we analyze the impact of different sharing policies on the video quality and the changes in video quality over time. These two factors actively contribute to the streaming experience of the user. The model-based results show how the different classes of players benefit from the coordination of the Control Element.

Given the Control Element and its versatility to execute different sharing policies, this thesis improves video streaming in two ways: First, the Control Element improves the general performance of HAS compared to the standalone and entirely client-based solutions. By eliminating the competition for bandwidth, the adverse side-effects of this competition are removed. Secondly, the Control Element matches the streaming performance to a policy. When the policy reflects the requirements of the users, the streaming performance will also match. As such, this thesis validates the core hypothesis of the thesis:

A network-based Control Element that guides HTTP adaptive video streaming clients improves the video quality and fairness in shared networks.

6.1 Revisiting the Research Questions

This thesis validates the core hypothesis by looking into the integration of the Control Element in the HAS architecture, the mechanisms that the Control Elements offers to HAS players and the networks it is controlling, the bandwidth sharing

policies in the Control Element, and the use of the Control Element to improve efficiency of network resource usage in mobile networks.

6.1.1 Integrating the Control Element in the HAS Architecture

Client-based streaming technologies, such as HAS, have significantly contributed to the success of online video streaming services. By moving the intelligence from the server to the player and the use of HTTP for transporting the video segments, HAS is a technology that excels in scalability and ease of deployment. Using HAS, content providers can distribute their content using HTTP-based delivery networks, leveraging technologies such as replication and caching. These technologies, in combination with HAS, enable content providers to scale their services to the masses. Solutions for improving video streaming performance thus should keep the advantageous aspects of HAS: use of HTTP servers and the HAS formats. Hence, the first research question poses the challenge of designing an optimization platform that uses HAS as the basis:

Research question 1: *How can a Control Element be designed to seamlessly integrate into existing HAS streaming infrastructures?*

Chapter 2 answers this question by reporting the design of the Control Element. The role of the Control Element is to guide HAS players that share a bottleneck network connection. The Control Element ensures that the available bandwidth is distributed evenly over the active HAS players and other traffic on the network link. The Control Element follows a sharing policy in doing so.

Our first implementation of the Control Element is a proxy server, which is transparently deployed in the network. The proxy server executes a sharing policy by rewriting the URLs in HTTP requests coming from a HAS player. Requests for video segments in a non-optimal video quality are changed into requests for the same segments in the optimal quality. Because the proxy server is transparent towards the HAS player and server, it does not require any changes to the basis of HAS. As such, the proxy server shows to be a simple, yet effective, implementation of the Control Element.

The proxy server satisfies the performance requirements and it is a solution that integrates well with HAS. However, there are other concerns outside of performance and integration with HAS. In a time where users are getting more concerned about their privacy, and the users without concerns should still be protected, the proxy server can be considered as an invasive solution. The proxy server requires inspecting the traffic, practically allowing the owner of the proxy server to analyze

what someone is watching. Depending on who is the owner (i.e., the place in the network where the proxy server is deployed), this behavior is infringing privacy.

This discussion on privacy challenged us to implement a solution that is as effective as the proxy server but honors the privacy of its users. Therefore, the second implementation of the Control Element that is presented in Chapter 2 features a dedicated control channel between the HAS players and the Control Element. This decision of using out-of-band communication meant that the player had to be altered. Although it might seem that altering the players breaks the integration with HAS, we allowed this change because the servers and the protocols remain unchanged. As such, the parts of HAS relevant for scaling and deploying a video streaming service stay intact.

6.1.2 Mechanisms for Improving Video Streaming

When HAS players share a bottleneck network connection, two performance problems arise. First, the constant competition for bandwidth causes unstable throughput, resulting in temporal lower video quality and frequent changes in video quality. The second performance problem is caused by the fact that HAS players are unaware of each other. HAS players will increase their video quality at the expense of other players that might need the bandwidth more. This process results in an unfair or not optimal sharing of network resources, in a way that it does not match the needs of the users. The root of both problems is that HAS players strive for selfish optimization instead of global optimization.

The Control Element is a central node in the network that knows active HAS players. It can use this knowledge of the active players to guide these players to streaming following a globally optimal division of network bandwidth. Depending on the mechanisms, the guidance from the Control Element is explicit or implicit. On the one hand, for example, the proxy server rewrites the segment requests to match the bitrate in those request to the sharing policy. Likewise, the SDN-based implementation explicitly signals target bitrates to the HAS players. On the other hand, traffic control has an implicit effect on the performance of HAS. The streaming performance increase depends on the mechanisms that are used to assist HAS players. In Chapter 3 of this thesis, an extensive evaluation of the mechanisms in the Control Element is performed to answer the second research question:

Research question 2: *(How) do the mechanisms in the Control Element improve video streaming performance in shared networks in terms of video freezes, video quality, quality switches, and fairness?*

Our evaluations show that all mechanisms contribute to the improvement of the video streaming experience. Both, rewriting the segment requests in the proxy server and signaling HAS players via an out-of-band channel, effectively prevent HAS players from selecting too high video bitrates and taking bandwidth that was supposed to go to other clients. This means that HAS players leave each other enough bandwidth for streaming at the globally optimal video bitrates.

When signaling the HAS player the optimal bitrate, it also increases the confidence of a player. Without knowledge of the optimal bitrate, increasing the video quality is a process of trial and error. Bandwidth estimations may indicate that a higher bitrate might be possible, but there is no guarantee of it. With knowledge of the optimal bitrate, HAS players can quickly converge to the optimal bitrate. This approach reduces the risks of selecting too low bitrates or only slowly increasing the bitrates in order to be cautious.

Based on the evaluations in Chapter 3, we can conclude that providing bitrate guidance is an adequate and sufficient solution in a network that is shared only by HAS players. When there is other traffic in the network, this traffic also has to adhere to the decisions of the Control Element (i.e., other traffic cannot take the bandwidth assigned to HAS players). Because cross-traffic can come from many applications that are not necessarily adaptive, a similar approach of bitrate guidance is not feasible. Chapter 3 shows that traffic control is a useful measure to separate HAS from cross-traffic, ensuring the bandwidth for HAS.

In fact, the evaluations prove that traffic control should not be omitted in networks with significant cross-traffic. This means that traffic control has to be used in addition to bitrate guidance for the Control Element to reach its optimal performance. Furthermore, in the most common case of shared internet connections, the Control Element should be deployed with all mechanisms activated. Only in networks that are dedicated to video streaming, traffic control can be disabled.

6.1.3 Modeling Sharing Policies

The policy in the Control Element defines how HAS players are supposed to share the available bandwidth. A policy is a definition that describes the division of bandwidth for every possible combination of players. To simplify the specification of sharing policies, we group players with similar characteristics. Groups can, for example, differ in the type of device, type of user, or type of stream. A policy then becomes the specification of a video bitrate for players in each group, taking into account the characteristics and the number of players.

The selection of video bitrates in a policy affects the streaming performance for players in each group. With the formulation of performance models, and by using

these models to study the effects of changes to sharing policies, Chapter 4 answers the third research question:

Research question 3: *How can the behavior of the Control Element be stochastically modeled and evaluated?*

To model the impact of a sharing policy, it is essential to know how many HAS players are active. In a dynamic network, HAS players will start and stop over time. Assuming randomly starting players, and by using the mean video duration as an indicator when describing the stopping process, we can model the number of active players as a Markov process. Through the analysis of the model, observing how often a specific number of players is active, we can estimate the overall number of video players. The model is defined such that video quality can be estimated for each of the different groups, as well as the overall video quality. Besides the video quality, Chapter 4 also looks into how often a sharing policy assigns different bitrates to the players in a group. This is a good indicator of how often the quality in a stream will change.

The model presented in Chapter 4 of this thesis has been validated against experiments with our Control Element. Our validations have proven that the model is highly accurate, and only has a neglectable margin of error. This means that we can reliably use this model to get results when evaluating sharing policies for our Control Element. We have used the model to specify three types of policies: bitrate fair, device-aware sharing, and policies including premium users.

We showed that changing the sharing policy affects the resulting video quality, benefiting the players that require more bandwidth compared to other. The model-based results for the number of changes in video quality mostly provide an indication when a sharing policy is active (i.e., quality switches only happen when the policy in the Control Element prescribes a different bitrate after a change in the number of players). Our results show that the overall number of quality switches is low, far below the threshold for annoyance as it was discovered in [57, 130].

From the findings in Chapter 4, we can conclude that Markov modeling is an effective method when studying bandwidth sharing policies for HAS. It allows to get performance results and deep insights much faster compared to using experimenting in a testbed.

6.1.4 Improving Quality and Efficiency in Mobile Networks

A core role of the Control Element is to assign players a target quality. After signaling the target bitrate to players, the Control Element has to ensure sufficient

resources for the players to stream at the target bitrates. In LTE mobile networks, guaranteeing a throughput means scheduling an appropriate number of resource blocks. Depending on the channel quality between the LTE base station and the client device, the number of resource blocks increases or decreases. This behavior implies that providing a constant throughput when channel quality is poor is excessively expensive. In Chapter 5, we show with simulation experiments that an increase of up to 30% in network resources has to be expected when delivering a constant video quality, compared to adapting the video quality while keeping the number of resource blocks constant.

From these simulations, we learned that adapting the video quality is beneficial concerning efficiency. However, high variations in channel quality will cause variations in video quality, resulting in a lower streaming experience to the user. In Chapter 5, we address the challenge of reducing the load on the network while providing the user with a constant quality streaming experience. Using a model-based optimization approach, Chapter 5 answers the fourth research question:

Research question 4: *How can modeling be used in the Control Element to achieve a better streaming experience and more efficient delivery of video streams in mobile networks?*

We collected a data set of LTE channel quality traces to understand the evolution of channel quality over time. The traces showed a considerable degree of randomness and historical information did not prove useful in predicting future channel quality. For these reasons, we opted for a Markov chain to model the channel quality, where the transitions in the Markov chain were fitted to the data set. Based on the Markov chain we could make short-term predictions about the channel quality.

The predictions were used to formulate and tune the following buffering strategy: increase the buffer when channel quality is excellent; play video from the buffer when channel quality is poor. The optimal buffer actions (grow, maintain, shrink) were obtained by solving the Markov decision process described in Chapter 5. The strategy is to map between the buffer level in the player and the current channel quality, and the best buffering action in the given situation.

In an initial simulation experiment, we found that applying the buffering strategy effectively reduces the load on the network. When using the buffering strategy, the network load is on average comparable to using off-the-shelf adaptation algorithms and guaranteeing a fixed number of resource blocks. From the results, we conclude that adapting the video quality or the buffer fill level are both efficient regarding network resource usage. Adapting the playout buffer has the advantage of a constant

video quality that is presented to the user. With our strategy, mobile operators can provide a better streaming experience by keeping video quality constant, for a reduction of the costs.

The strategy presented in Chapter 5 is an example of an optimization that works both on the network level and the application level. The information flowing from the HAS player to the Control Element and the feedback from the Control Element to the HAS player cannot be omitted, or else the optimization cannot be reached. This optimization stresses again the importance and the potential of a communication channel between applications and network elements.

6.2 Reflection and Outlook

The Control Element applies to HAS, an established Internet application with many users. In this section, we reflect on the impact that our Control Element may have on current and future video based streaming applications.

6.2.1 Impact on Online Video Streaming

The Control Element is most useful in networks with limited bandwidth, such that the Control Element can make an optimal division of bandwidth. In networks that provide high download speeds, selecting a video streaming bitrate is a trivial excise. Both the Control Element and a "stand-alone" HAS player will select the highest video quality representation, because bandwidth for the highest representation would be available. In those high-speed networks, the Control Element has a limited effect. Although in some countries is it possible to get high-speed Internet connections (e.g., in The Netherlands ISPs offer home Internet packages with download speeds up to 500 Mbit/s), such Internet connections are not available to everyone and it may take several years before they become available. For example, in Germany in 2017, about half of the Internet connections have download speeds below 10 Mbit/s [6]. For large areas in the Middle East, Africa, and the Southern Americas, the fraction of Internet connections with download speeds above 10 Mbit/s is lower than 10%. For most networks is it not possible to have two concurrent full-HD video streaming, according to Netflix's recommendation of 5 Mbit/s per streams [95]. This means that for many networks, the Control Element can improve video streaming performance and as such enhance the streaming experience.

Improving the streaming experience is done by deploying the Control Element in a network. Network hardware or software has to be upgraded for the Control Element to become available in a network. Network elements, such as switches or

Wi-Fi routers need to host the Control Element software. For optimal performance, the network elements need some form of traffic control support. Fortunately, the algorithms in the Control Element are not computationally expensive. The number of control messages between the HAS players is low. Furthermore, traffic control already proves to be adequate with as little as two traffic queues. This means that the Control Element can, in theory, be integrated into low computing-powered network hardware, such as household Wi-Fi routers. We showed in this thesis that a Raspberry Pi 2 device already has more than sufficient capabilities.

Besides upgrades to network hardware and depending on the implementation of the Control Element, the HAS players needs to be altered. When using the SDN-based implementation, players maintain a separate communication channel with the Control Element. Current players from large content providers do not have support for this communication channel yet. However, with the standardization efforts of SAND [132], which provides a comparable communication channel to the Control Element, support is expected to grow.

Furthermore, the Control Element is an opportunity for network providers that also offer television. Television watching is moving from the traditional single TV to a live streaming platform available on a variety of devices. Currently, providers struggle to reach the users on the different devices. The multicast IPTV-based solutions that have been used until now require dedicated set top boxes and managed network connections. Moving the TV services to a HAS-based solution cannot be done without a loss in performance, or a guarantee of performance, as providers offered until now. Additionally, when providing the streams at low latency, HAS players cannot rely on large buffers to deal with delivery problems. Given that network providers control the whole chain (i.e., from the server, via their own network, to the player), they can alter the HAS player and use the Control Element in their networks. With the Control Element, video quality can be improved, and specific quality levels can be guaranteed for different devices. As such, the Control Element may act as an enabler for modernizing the TV delivery infrastructures.

6.2.2 Novel Media Applications

In this thesis, the focus is on a Control Element for online video streaming. Video streaming has been used as an example that shows how transporting large amounts of data using simplified protocols leads to performance deficits. Because video streaming is a visual activity, the impact of poor network performance on the streaming experience is often severe. The time sensitive nature of video streaming and high throughputs for delivering it, make video streaming a challenge. A challenge that is not yet over.

Even though network performance is increasing and developments in network hardware will continue to deliver higher throughputs, the bandwidth requirements for video streaming applications will continue to grow with them. For the past few years, full-HD has been the standard for high quality video streaming. However, as the market pushes forward, the availability and affordability of 4K devices increases. Where 4K is on its way to become the standard, even higher resolution 8K displays are already available for purchase.

Besides resolution, camera and display manufacturers also work on increasing displaying properties. A good example is high dynamic range (HDR) video, which increases the bit depth gradient step precision. While HDR offers a superior viewing experience, video bitrates need to increase to support HDR. Pushing the boundaries of video quality forward, video streaming will remain a challenging application.

In addition to "traditional" video streaming, the emergence of immersive video-based applications can be observed as well. Users can transport themselves in virtual realities (VR), using techniques such as omnidirectional/360 degree video streaming. To increase the experience and immersive feeling, VR applications use head mounted displays (HMDs). In HMDs, the screens are close to the users' eyes, and provide a large field of view. However, having screens close to the eyes stresses the importance of resolution. Video resolutions have to go far beyond 8K before individual pixels cannot be distinguished by the human eye. Given that VR streaming applications also need to account for head motion, not only the area within the field of view has to be downloaded, also the areas around it have to be available. The bandwidth requirements for VR streaming further increase with stereoscopic imaging, enabling users to experience depth in the video. With stereoscopic streams, both eyes get a separate stream presented. Increasing the degrees of freedom will even further push the required network bandwidth. Additionally, network latency will play a role because of the high level of user interactivity.

The discussion on the future of video streaming, either about traditional streaming or about novel immersive media applications, shows that video streaming is nowhere near its boundaries. Networking requirements will continue to grow, especially in immersive applications like VR streaming. Careful management of resources will be essential to provide users with the best experience, further requiring the need for a Control Element as provided in this thesis.

6.3 Closing Thoughts

Network support for online video streaming and novel media applications goes beyond the engineering efforts to increase bandwidth in shared networks. Because

bandwidth is limited and multiple users will access the same pool of resources, it is important to focus on how bandwidth can be shared. Therefore, this thesis studies both mechanisms and sharing policies. When focusing on the mechanisms, we use an empirical approach. For the policies we use a more high-level and theoretical approach. The two different views on the topic of network optimization for video streaming, and the two different approaches to investigate this problem, typically belong to different research disciplines. However, with this thesis, we deliberately created a cross-over between the multimedia systems and teletraffic performance research disciplines. This interdisciplinary approach enabled us to holistically study the problem of optimizing video streaming performance in shared networks. For example, because we worked with video streaming players and traffic management mechanisms, we could identify the key parameters for the performance models. Using the performance models, we could develop sharing policies and evaluate them under many different network loads before implementing them in the Control Element.

Although interdisciplinary research is often stimulated and praised, its process is not without hurdles. Researchers from different disciplines speak different languages. Even when the overall goal is clear (e.g., increase the video streaming quality in shared networks), the methodologies and even the objectives may differ. In multimedia systems research, full prototypes and realistic experiments are highly appreciated. This means that a large amount of time and effort goes into the development of prototypes and experiments. In contrast to this experimental approach in multimedia systems research, teletraffic research takes a theoretical approach, which focusses on modeling of systems and the optimization potential of these models. Analytical elegance, novel modeling and optimization methods, and feasibility of model-based studies have the preference. Abstract simulations often suffice for validating the models.

Bringing researchers from two disciplines together means changing viewpoints. This process requires an investment. Working with people from another discipline means that there is a knowledge deficit on both sides. Time and effort is required to teach the important aspects of each field to bring everyone on the same page. Interdisciplinary research projects can only make true steps when one understands and trusts each other. Interdisciplinary research requires efforts and a time investment. Even though at times it might seem to slow work down, in the long run it is worth it.

Summary

Computer networks are shared by multiple users and devices. Traffic from multiple applications is combined on a single network link, dividing the bandwidth between these applications. Video streaming applications require a relatively large and stable share of the bandwidth to be able to deliver high quality video. The architecture and protocols used by the current dominant streaming technology, HTTP Adaptive Streaming (HAS), have enabled the massive scale of video content delivery as we know it today. Yet they are not suitable for meeting the demanding bandwidth requirements in shared networks. Cross-traffic from other applications and other video streams have a large impact, causing HAS players to lower the video quality, frequently undershoot and overshoot the video quality compared to the available bandwidth, and even interrupt the stream for rebuffering.

Failure to deliver high quality video streams reflects badly on content providers and network operators, who risk the loss of revenue due to unsatisfied users. Video delivery problems distract the user, cause annoyance, and may lead to users abandoning their streams. Many scientific efforts have been performed to improve the streaming experience by optimizing the internals of HAS players. Such optimizations focus on the adaptation algorithms in HAS players to overcome bandwidth estimation errors, but they overlook the crucial role of the network in providing an environment where HAS players can operate without difficulty.

In this dissertation, we validate the hypothesis that *the addition of a network-based control element that guides HTTP adaptive streaming clients improves the video quality and fairness between clients in shared networks*. To this end, we develop, analyze, and optimize mechanisms and policies for bandwidth sharing between HAS clients. We combine empirical and theoretical research methodologies to gain a broad understanding of network optimizations for video streaming. Real implementations and experiments in different networks allow us to evaluate and improve the mechanisms for interaction between HAS clients and the network. We use Markov modeling to understand the impact of sharing policies on the

overall streaming performance, allowing faster results and more insights on the optimization of sharing policies that meet the requirements of the user.

A network-based control element for HAS optimization

No network elements providing the necessary flexibility and functionality exist that allow us to validate our hypothesis. In **Chapter 2**, we introduce the concept and two embodiments of the Control Element. The Control Element gathers information about the network and active HAS players, divides the network bandwidth among the HAS players according to a sharing policy, and translates bandwidth sharing decisions into traffic control rules. The first implementation of the Control Element is a HAS-aware proxy server, focusing on a transparent solution towards HAS clients. We describe the design of a proxy server that obtains HAS related information by monitoring HTTP requests. The proxy server controls HAS clients by modifying certain HTTP requests. The second implementation is a network element that maintains a control channel with HAS clients to obtain status information and provide bitrate recommendations. This implementation uses Software Defined Networking (SDN) techniques to control the network infrastructure. The Control Element improves over the related work, because it creates a network environment where HAS players stream at the optimal bitrate given the current network state.

To prove this concept, we describe a series of experiments that evaluate the Control Element. In **Chapter 3**, we compare the streaming performance of reference HAS implementations in best effort networks with HAS in networks optimized by our Control Element. We assess the two different implementations of the Control Element using wired and wireless testbeds, focusing on video freezes, video quality, changes in video quality, and fairness between players. Furthermore, we investigate the impact of different traffic control techniques on HAS, comparing different types of traffic control queues, varying the size of the queues, varying the number of queues in the network, and varying the number of HAS clients per queue. We show that our solution improves the streaming experience by preventing video freezes, minimizing the number of quality switches, and providing a fair video quality to all players. In experiments with up to 600 concurrent HAS clients, we show scalability and effectiveness of the Control Element in larger networks.

Modeling network-assisted HAS

While the evaluations in Chapter 3 prove the streaming performance increase when using the Control Element, implementation and experimentation is time consuming. In **Chapter 4**, we present a method for understanding and predicting the overall streaming performance given a bandwidth sharing policy in the Control Element. By formulating the starting and stopping of different types of players in a multi-dimensional Markov model that takes HAS characteristics into account, our model

predicts the video quality, the number of video quality switches, and fairness for each type of player. We perform a model-based analysis of sharing policies that include HAS players with different device characteristics and with premium users. In an extensive validation, comparing the model-based results with results from the experiments with the Control Element, we show the high accuracy of our model. This allows us to gain insights on the performance of HAS with the Control Element much faster.

More and more video is consumed via mobile networks. These networks have different characteristics in terms of throughput and resource allocation compared to fixed and Wi-Fi networks. This raises the question, how the Control Element can improve the video quality while also increasing the delivery efficiency. In **Chapter 5**, we present an extensive data set of LTE channel quality traces and formulate a Markov model for predicting the LTE channel quality in the near future. Based on this model, we formulate a strategy that combines resource allocation in the mobile network with buffering in the client, exploiting the fact that data transmissions under good channel quality are more efficient compared transmission under poor channel quality. The strategy aims at proving a stable video quality, while minimizing the needed resources in the network. The parameters for the strategy are obtained by solving a Markov Decision Process. We show that we can stream video in a stable quality and increase the efficiency of the mobile network.

Nederlandse Samenvatting

Computernetwerken worden gedeeld door meerdere gebruikers en apparaten. Het verkeer van verschillende applicaties wordt samengevoegd op een netwerkverbinding waarvan de bandbreedte gedeeld moet worden. Videostreaming vereist een relatief hoog en stabiel deel van de bandbreedte om een hoge videokwaliteit te kunnen leveren. De architectuur en protocollen die worden gebruikt door HTTP Adaptive Streaming (HAS), de meest gebruikte technologie voor videostreaming, hebben sterk bijgedragen aan de massale schaal waarop videostreaming tegenwoordig plaatsvindt. Echter, de HAS architectuur en protocollen zijn ongeschikt voor de hoge eisen die gebruikers aan ze stellen. Verkeer van andere applicaties en andere videospelers in het netwerk hebben een aanzienlijke impact op de prestaties van HAS. Dit kan leiden tot een verlaging van de videokwaliteit, het regelmatig wisselen naar een te hoge of een te lage videokwaliteit ten opzichte van de beschikbare bandbreedte, of het moeten pauzeren van een stream omdat de buffer leeg is.

Het niet kunnen leveren van een hoge videokwaliteit is slecht voor aanbieders van online videostreaming en netwerkoperatoren. Zij riskeren een verlies van inkomsten als gevolg van ontevreden gebruikers. Problemen bij het afleveren van videostreams kunnen de gebruiker afleiden, zijn vervelend, en gebruikers kunnen er zelfs voor kiezen een stream te stoppen. Veel wetenschappelijke inspanningen zijn gedaan om de gebruikerservaring te verbeteren. Veel van deze inspanningen richten zich op het verbeteren van de HAS speler, maar zien de belangrijke rol van het network over het hoofd.

In dit proefschrift valideren we de hypothese: *het toevoegen van een controle-element in het network voor het assisteren van HAS spelers verbetert de videokwaliteit en de eerlijkheid in het netwerk*. Hiertoe ontwikkelen, analyseren, en optimaliseren we mechanismen en strategieën voor het verdelen van bandbreedte tussen HAS spelers. We combineren empirische- en theoretische onderzoeksmethodologieën om tot inzichten te komen hoe netwerkoptimalisatie voor videostreaming werkt. Door te werken met echte implementaties in de experimenten stellen we ons staat om

de verschillende mechanismen grondig te evalueren en te verbeteren. Daarnaast gebruiken we modelleringstechnieken om snel strategieën door te rekenen en voor het begrijpen welke impact een strategie heeft op de algehele prestaties.

Een controle-element in het netwerk voor HAS optimalisatie

Er bestaan geen netwerkelementen die de benodigde flexibiliteit en functionaliteit bieden om onze hypothese te valideren. Daarom presenteren we in **hoofdstuk 2** het concept en twee uitvoeringen van ons controle-element. Het controle-element verzamelt informatie over het netwerk en de actieve videospelers. Op basis van deze informatie en een strategie verdeelt het controle-element de bandbreedte tussen de videospelers. De eerste implementatie van het controle-element heeft de vorm van een proxyserver. Deze transparante oplossing verzamelt informatie over actieve videospelers door het monitoren van HTTP verkeer. De proxyserver kan videospelers sturen door specifieke HTTP verzoeken aan te passen. De tweede implementatie van het controle-element maakt gebruik van Software Defined Networking (SDN) om de netwerkinfrastructuur te configureren. Deze implementatie communiceert direct met de videospelers om informatie te verkrijgen en om instructies uit te delen. Met behulp van het controle-element in een netwerk kunnen videospelers de optimale videokwaliteit leveren gegeven de huidige staat van het netwerk.

Om dit concept te evalueren beschrijven we een serie van experimenten in **hoofdstuk 3**. We vergelijken de prestaties van HAS spelers in gewone netwerken met de prestaties van HAS spelers in een netwerk waarin ons controle-element actief is. Voor beide implementaties van het controle-element kijken we naar interrupties in videostreams, de kwaliteit van de videostreams, hoe vaak een videospeler moet wisselen van videokwaliteit, en de eerlijkheid van de verdeling van bandbreedte. Daarnaast onderzoeken we de impact van verschillende technieken om HAS verkeer te prioriteren, waarbij we het type, het aantal, en de grote van de wachtrijen in een netwerkswitch variëren. We tonen aan dat onze oplossing helpt om pauzes in videostreams te voorkomen, het aantal wisselingen in videokwaliteit drastisch te verlagen en een eerlijke bandbreedteverdeling te maken voor alle videospelers in het netwerk. In experimenten met 600 gelijktijdige videospelers tonen we de schaalbaarheid van onze oplossing.

Modelleren van HAS in netwerken met het controle-element

De evaluaties in hoofdstuk 3 laten zien dat ons controle-element de HAS prestaties verbetert. Echter, het experimenteren met het controle-element is tijdsintensief. In **hoofdstuk 4** presenteren we een methode die ons helpt bij het begrijpen en voorspellen van de prestaties van HAS gegeven een strategie in het controle-element. Door het process van startende en stoppende HAS spelers en een verdelingsstrategie in een model te vatten kunnen we de videokwaliteit, wisselingen van videokwa-

liteit, en de verdeling van bandbreedte tussen verschillende types videospelers voorspellen. Op basis van dit model maken we een analyse van strategieën voor het differentiëren van verschillende apparaten en gebruikers. Daarnaast bevat dit hoofdstuk een uitgebreide validatie om de hoge nauwkeurigheid van ons model aan te tonen. Als zodanig is ons model een effectieve methode om snel tot resultaten en inzichten te komen omtrent het verdelen van bandbreedte tussen videospelers in een gedeeld netwerk.

Steeds meer video wordt geconsumeerd via mobiele netwerken. In vergelijking met vaste netwerken hebben mobiele netwerken andere eigenschappen als het gaat om het toewijzen van bandbreedte. De vraag dringt zich op hoe het controle-element de videokwaliteit en de efficiëntie van videostreaming in mobiele netwerken kan verhogen. In **hoofdstuk 5** presenteren we een uitgebreide dataset met LTE signaalkwaliteit metingen. Met behulp van deze metingen formuleren we een Markov model om de LTE signaalkwaliteit in de nabije toekomst te voorspellen. Op basis van dit model maken we een strategie die de toewijzing van bandbreedte en het bufferen van video in de HAS speler combineert. De strategie kan de efficiëntie van het mobiele netwerk vergroten door extra video te downloaden wanneer de signaalkwaliteit goed is en minder video te downloaden wanneer de signaalkwaliteit slecht is. We bepalen de optimale parameters voor de strategie door middel van het oplossen van een Markov beslisproces. We tonen aan dat onze strategie de videokwaliteit stabiel kan houden en de efficiëntie van het mobiele netwerk kan verhogen.

Publications by the Author

[K1] **J.W. Kleinrouweler**, B. Meixner, J. Bosman, H. van den Berg, R. van der Mei, and P. Cesar. Improving Mobile Video Quality Through Predictive Channel Quality Based Buffering. In *Proceedings of the 30th International Teletraffic Congress*, ITC30, pages 236-244, Vienna, Austria, ITC/IEEE, 2018.

[K2] B. Meixner, **J.W. Kleinrouweler**, and P. Cesar. 4G/LTE Channel Quality Reference Signal Trace Data Set. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 387-392, New York, NY, USA, ACM, 2018.

[K3] **J.W. Kleinrouweler**. Using DASH Assisting Network Elements for Optimizing Video Streaming Quality. In *Proceedings of the 2017 ACM Multimedia Conference*, MM '17, pages 821-825, New York, NY, USA, ACM, 2017.

[K4] **J.W. Kleinrouweler**, S. Cabrero, and P. Cesar. An SDN Architecture for Privacy-Friendly Network-Assisted DASH. *ACM Transactions on Multimedia Computing, Communications, and Applications* 14(3s):44:1-44:22, ACM, 2017.

[K5] **J.W. Kleinrouweler**, B. Meixner, and P. Cesar. Improving Video Quality in Crowded Networks Using a DANE. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '17, pages 73-78, New York, NY, USA, ACM, 2017.
[DASH-IF Excellence in DASH Award; 2nd place]

[K6] **J.W. Kleinrouweler**. Enhancing Over-the-Top Video Streaming Quality with DASH Assisting Network Elements. In *Adjunct Publication of the 2017 ACM International Conference on Interactive Experiences for TV and Online Video*, TVX '17 Adjunct, pages 113-116, New York, NY, USA, ACM, 2017.
[Best Doctoral Consortium Paper Award]

[K7] **J.W. Kleinrouweler**, S. Cabrero, R. van der Mei, and P. Cesar. A Model for Evaluating Sharing Policies for Network-assisted HTTP Adaptive Streaming. *Computer Networks*, 19:234-245, Elsevier, 2016.

[K8] **J.W. Kleinrouweler**, S. Cabrero, R. van der Mei, and P. Cesar. A Markov Model for Evaluating Resource Sharing Policies for DASH Assisting Network Elements. In *Proceedings of the 28th International Teletraffic Congress*, ITC28, pages 112-120, Würzburg, Germany, ITC/IEEE, 2016.

[K9] **J.W. Kleinrouweler**, S. Cabrero, and P. Cesar. Delivering Stable High-quality Video: an SDN Architecture With DASH Assisting Network Elements. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, pages 4:1-4:10, New York, NY, USA, ACM, 2016.
[DASH-IF Excellence in DASH Award]

[K10] **J.W. Kleinrouweler**, S. Cabrero, R. van der Mei, and P. Cesar. Modeling the Effect of Sharing Policies for Network-assisted HTTP Adaptive Video Streaming. *SIGMETRICS Performance Evaluations Review*, 43(2):26-27, ACM, 2015.

[K11] **J.W. Kleinrouweler**, S. Cabrero, R. van der Mei, and P. Cesar. Modeling Stability and Bitrate of Network-assisted HTTP Adaptive Streaming Players. In *Proceedings of the 27th International Teletraffic Congress*, ITC27, pages 177-184, Ghent, Belgium, ITC/IEEE, 2015.

Bibliography

[1] 3GPP TS 26.247 Transparent end-to-end Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH). Technical Specification, 2018.

[2] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. Zhang, M. Varvello, and M. Steiner. Measurement Study of Netflix, Hulu, and a Tale of Three CDNs. *IEEE/ACM Transactions on Networking*, 23(6):1984–1997, 2015.

[3] Adobe Systems Inc. Real Time Messaging Protocol (RTMP) Specification. Specification, 2012. URL: `https://wwwimages2.adobe.com/content/dam/acom/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf` (accessed on February 14, 2020).

[4] A. Ahmad, A. Floris, and L. Atzori. QoE-centric Service Delivery: A Collaborative Approach Among OTTs and ISPs. *Computer Networks*, 110:168–179, 2016.

[5] J. Ahrenholz. Comparison of CORE Network Emulation Platforms. In *Military Communications Conference*, MILCOM '10, pages 166–171, San Jose, CA, USA. IEEE, 2010.

[6] Akamai. State of the Internet - Q1 2017 report, 2017. URL: `https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf` (accessed on February 19, 2019).

[7] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis. What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth? In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV '12, pages 9–14, New York, NY, USA. ACM, 2012.

[8] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems*

Support for Digital Audio and Video, NOSSDAV '13, pages 19–24, New York, NY, USA. ACM, 2013.

[9] S. Akhshabi, A. C. Begen, and C. Dovrolis. An Experimental Evaluation of Rate-adaptation Algorithms in Adaptive Streaming over HTTP. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, pages 157–168, New York, NY, USA. ACM, 2011.

[10] Android Developers. Services overview. Website, 2017. URL: `https://developer.android.com/guide/components/services` (accessed on May 6, 2019).

[11] Apache Documentation Group. The Apache HTTP Server Project. Website, 2018. URL: `https://httpd.apache.org` (accessed on July 30, 2018).

[12] I. Apple. How To Use QuickTime Player. Website, 2018. URL: `https://support.apple.com/en-us/HT201066` (accessed on July 30, 2018).

[13] A. Beben, P. Wiśniewski, J. M. Batalla, and P. Krawiec. ABMA+: Lightweight and Efficient Algorithm for HTTP Adaptive Streaming. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, 2:1–2:11, New York, NY, USA. ACM, 2016.

[14] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. A survey on bitrate adaptation schemes for streaming media over http. *IEEE Communications Surveys & Tutorials*, Early Access, 2018.

[15] A. Bentaleb, A. C. Begen, R. Zimmermann, and S. Harous. SDNHAS: An SDN-enabled architecture to optimize QoE in HTTP adaptive streaming. *IEEE Transactions on Multimedia*, 19(10):2136–2151, 2017.

[16] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Want to Play DASH?: A Game Theoretic Approach for Adaptive Streaming over HTTP. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 13–26, New York, NY, USA. ACM, 2018.

[17] A. Bentaleb, A. C. Begen, and R. Zimmermann. ORL-SDN: Online Reinforcement Learning for SDN-Enabled HTTP Adaptive Streaming. *Transactions on Multimedia Computing, Communications, and Applications*. TOMM, 14(3), 2018.

[18] A. Bentaleb, A. C. Begen, and R. Zimmermann. SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking. In *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, pages 1296–1305, New York, NY, USA. ACM, 2016.

[19] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. Network Assisted Content Distribution for Adaptive Bitrate Video Streaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys '17, pages 62–75, New York, NY, USA. ACM, 2017.

[20] A. Biernacki and K. Tutschku. Performance of HTTP Video Streaming under Different Network Conditions. *Multimedia Tools and Applications*. MTAP '14, 72(2):1143–1166, 2014.

[21] D. Black and P. Jones. Differentiated Services (Diffserv) and Real-Time Communication. Request for Comments, 2015. URL: `https://tools.ietf.org/html/rfc7657` (accessed on July 12, 2018).

[22] B. Blaszczyszyn, M. Jovanovic, and M. K. Karray. Quality of Real-Time Streaming in Wireless Cellular Networks — Stochastic Modeling and Analysis. *IEEE Transactions on Wireless Communications*, 13(6):3124–3136, 2014.

[23] N. Bouten, M. Claeys, B. V. Poecke, S. Latré, and F. D. Turck. Dynamic Server Selection Strategy for Multi-server HTTP Adaptive Streaming Services. In *Proceedings of the 12th International Conference on Network and Service Management*, CNSM '16, pages 82–90, Montreal, QC, Canada, 2016.

[24] N. Bouten, J. Famaey, S. Latré, R. Huysegems, B. De Vleeschauwer, W. Van Leekwijck, and F. De Turck. QoE Optimization Through In-network Quality Adaptation for HTTP Adaptive Streaming. In *Proceedings of the 8th International Conference on Network and Service Management*, CNSM '12, pages 336–342, Las Vegas, NV, USA. IFIP, 2013.

[25] B. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Request for Comments, 1997. URL: `https://tools.ietf.org/html/rfc1633` (accessed on July 12, 2018).

[26] E. F. Camacho and C. B. Alba. *Model Predictive Control*. Springer Science & Business Media, 2013.

[27] N. Carlsson and D. Eager. Ephemeral Content Popularity at the Edge and Implications for On-Demand Caching. *Transactions on Parallel and Distributed Systems*, 28(6):1621–1634, 2017.

[28] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 389–400, New York, NY, USA. ACM, 2013.

[29] F. Chiariotti, S. D'Aronco, L. Toni, and P. Frossard. Online Learning Adaptation Strategy for DASH Clients. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA. ACM, 2016.

[30] Christopher Mueller. MPEG-DASH (Dynamic Adaptive Streaming over HTTP), 2015. URL: `https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/` (accessed on September 15, 2019).

[31] L. D. Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. ELASTIC: A Client-Side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *Proceedings of the 20th International Packet Video Workshop*, PV '13, pages 1–8, San Jose, CA, USA. IEEE, 2013.

[32] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. Design and Experimental Evaluation of Network-assisted Strategies for HTTP Adaptive Streaming. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, 3:1–3:12, New York, NY, USA. ACM, 2016.

[33] G. Cofano, L. D. Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo. Design and Performance Evaluation of Network-assisted Control Strategies for HTTP Adaptive Streaming. *Transactions on Multimedia Computing, Communications, and Applications*. TOMM, 13(3s), 2017.

[34] S. Cordwell. Markov decision process (mdp) toolbox for python. Website. URL: `https://github.com/sawcordwell/pymdptoolbox` (accessed on May 6, 2019).

[35] N. Cranley, P. Perry, and L. Murphy. User Perception of Adapting Video Quality. *International Journal of Human-Computer Studies*, 64(8):637–647, 2006.

[36] S. D'Aronco, L. Toni, and P. Frossard. Price-based Controller for Utility-aware HTTP Adaptive Streaming. *IEEE MultiMedia*, Early Access, 2018.

[37] DASH Industry Forum. A reference client implementation for the playback of MPEG DASH via Javascript and compliant browsers. Website. URL: `https://github.com/Dash-Industry-Forum/dash.js/` (accessed on August 6, 2018).

[38] DASH Industry Forum. Guidelines for Implementation: DASH-AVC/264 Interoperability Points. Online report, 2013. URL: `https://dashif.org/wp-content/uploads/2015/04/DASH-AVC-264-base-v1.03.pdf` (accessed on August 6, 2018).

[39] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback Control for Adaptive Live Video Streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, pages 145–156, New York, NY, USA. ACM, 2011.

[40] A. Detti, B. Ricci, and N. Blefari-Melazzi. Tracker-assisted rate adaptation for MPEG DASH live streaming. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications*, INFOCOM '16, pages 1–9, San Francisco, CA, USA. IEEE, 2016.

[41] A. Developers. android.telephony. Website. URL: `https://developer.android.com/reference/android/telephony/package-summary` (accessed on May 6, 2019).

[42] M. Devera. Hierarchical Token Buckets (HTB). Website, 2003. URL: http://luxik.cdi.cz/~devik/qos/htb/ (accessed on August 6, 2018).

[43] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. *Communications of the ACM*, 56(3):91–99, 2013.

[44] A. E. Essaili, D. Schroeder, D. Staehle, M. Shehada, W. Kellerer, and E. Steinbach. Quality-of-experience driven adaptive HTTP media delivery. In *Proceedings of the International Conference on Communications*, ICC '13, pages 2480–2485, Budapest, Hungary. IEEE, 2013.

[45] J. Esteban, S. A. Benno, A. Beck, Y. Guo, V. Hilt, and I. Rimac. Interactions Between HTTP Adaptive Streaming and TCP. In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV '12, pages 21–26, New York, NY, USA. ACM, 2012.

[46] A. Farshad, P. Georgopoulos, M. Broadbent, M. Mu, and N. Race. Leveraging SDN to provide an in-network QoE measurement framework. In *Proceedings of the Conference on Computer Communications Workshops*, INFOCOM WKSHPS, pages 239–244, Hong Kong, China. IEEE, 2015.

[47] I. Fette and M. A. The WebSocket Protocol. Request for Comments, 2011. URL: https://tools.ietf.org/html/rfc6455 (accessed on July 13, 2018).

[48] M. Gadaleta, F. Chiariotti, M. Rossi, and A. Zanella. D-DASH: A Deep Q-Learning Framework for DASH Video Streaming. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):703–718, 2017.

[49] Geni - exploring networks of the future. Website. URL: https://www.geni.net (accessed on February 21, 2019).

[50] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race. Towards Network-wide QoE Fairness Using Openflow-assisted Adaptive Video Streaming. In *Proceedings of the ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, FhMN '13, pages 15–20, New York, NY, USA. ACM, 2013.

[51] Google. Chrome Web Browser. Website. URL: https://www.google.com/chrome/ (accessed on August 6, 2016).

[52] V. Gueant. iPerf - The TCP, UDP and SCTP Network Bandwidth Measurement Tool. Website. URL: https://iperf.fr/ (accessed on August 6, 2018).

[53] J. Hao, R. Zimmermann, and H. Ma. GTube: Geo-predictive Video Streaming over HTTP in Mobile Environments. In *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, pages 259–270, New York, NY, USA, 2014.

[54] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. Request for Comments, 2018. URL: https://tools.ietf.org/html/rfc8290 (accessed on February 19, 2019).

[55] H. Holma, A. Toskala, and J. Reunanen. *LTE Small Cell Optimization: 3GPP Evolution to Release 13*. John Wiley & Sons, 2016.

[56] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. Assessing Effect Sizes of Influence Factors Towards a QoE Model for HTTP Adaptive Streaming. In *Proceedings of the Sixth International Workshop on Quality of Multimedia Experience*, QoMEX '14, pages 111–116, Singapore, Singapore. IEEE, 2014.

[57] T. Hoßfeld, M. Seufert, C. Sieber, T. Zinner, and P. Tran-Gia. Identifying QoE Optimal Adaptation of HTTP Adaptive Streaming Based on Subjective Studies. *Computer Networks*, 81:320–332, 2015.

[58] hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. Website. URL: https://w1.fi/hostapd/ (accessed on August 6, 2018).

[59] R. Houdaille and S. Gouache. Shaping HTTP Adaptive Streams for a Better User Experience. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 1–9, New York, NY, USA. ACM, 2012.

[60] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, pages 225–238, New York, NY, USA. ACM, 2012.

[61] T.-Y. Huang, R. Johari, and N. McKeown. Downton Abbey Without the Hiccups: Buffer-based Rate Adaptation for HTTP Video Streaming. In *Proceedings of the ACM SIGCOMM Workshop on Future Human-centric Multimedia Networking*, FhMN '13, pages 9–14, New York, NY, USA. ACM, 2013.

[62] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 187–198, New York, NY, USA. ACM, 2014.

[63] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, J. Spaans, and P. Larroy. Linux Advanced Routing & Traffic Control HOWTO. Website, 2012. URL: http://lartc.org (accessed on July 30, 2018).

[64] C. Huitema. Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP). Request for Comments, 2003. URL: https://tools.ietf.org/html/rfc3605 (accessed on July 17, 2018).

[65] C. Ide, R. Falkenberg, D. Kaulbars, and C. Wietfeld. Empirical Analysis of the Impact of LTE Downlink Channel Indicators on the Uplink Connectivity.

In *2016 IEEE 83rd Vehicular Technology Conference*, VTC Spring, pages 1–5, Nanjing, China. IEEE, 2016.

[66] ISO/IEC 23009-1 Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segment formats, 2014.

[67] ISO/IEC 23009-5:2017 Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 5: Server and network assisted DASH (SAND), 2017.

[68] H. Jacob. hostapd: Initial OVS support. Website, 2013. URL: `https://github.com/helmut-jacob/hostapd/commit/c89daae` (accessed on August 6, 2018).

[69] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking*, 22(1):326–340, 2014.

[70] V. Joseph, S. Borst, and M. I. Reiman. Optimal rate allocation for adaptive wireless video streaming in networks with user dynamics. In *Proceedings of the IEEE Conference on Computer Communications*, INFOCOM '14, pages 406–414, Toronto, ON, Canada. IEEE, 2014.

[71] V. Joseph and G. de Veciana. NOVA: QoE-driven optimization of DASH-based video delivery in networks. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 82–90, Toronto, ON, Canada. IEEE, 2014.

[72] P. Juluri, V. Tamarapalli, and D. Medhi. SARA: Segment aware rate adaptation algorithm for dynamic adaptive streaming over HTTP. In *Proceedings of the International Conference on Communication Workshop*, ICCW '15, pages 1765–1770, London, UK, 2015.

[73] J. W. Kleinrouweler, S. Cabrero, R. van der Mei, and P. Cesar. Modeling Stability and Bitrate of Network-Assisted HTTP Adaptive Streaming Players. In *Proceedings of the 27th International Teletraffic Congress*, ITC27, pages 177–184. ITC/IEEE, 2015.

[74] J. W. Kleinrouweler, S. Cabrero, and P. Cesar. Delivering Stable High-quality Video: An SDN Architecture with DASH Assisting Network Elements. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, 4:1–4:10, New York, NY, USA. ACM, 2016.

[75] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan. CellSlice: Cellular wireless resource slicing for active RAN sharing. In *Proceedings of the Fifth International Conference on Communication Systems and Networks*, COMSNETS '13, pages 1–10, Bangalore, India. IEEE, 2013.

[76] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[77] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan. BUFFEST: Predicting Buffer Conditions and Real-time Requirements of HTTP(S) Adaptive Streaming Clients. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys'17, pages 76–87, New York, NY, USA. ACM, 2017.

[78] J. Kua, G. Armitage, and P. Branch. The Impact of Active Queue Management on DASH-Based Content Delivery. In *Proceedings of the 41st Conference on Local Computer Networks*, LCN '17, pages 121–128, Dubai, United Arab Emirates. IEEE, 2017.

[79] S. Lederer. Why YouTube & Netflix use MPEG-DASH in HTML5. Website, 2015. URL: `https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/` (accessed on July 17, 2018).

[80] LG. Lg nexus 5x. Website, 2017. URL: `https://www.lg.com/nl/smartphones/lg-Nexus-5X-H791F-smartphone` (accessed on May 6, 2019).

[81] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[82] Z. Li, A. C. Begen, J. Gahm, Y. Shan, B. Osler, and D. Oran. Streaming Video over HTTP with Consistent Quality. In *Proceedings of the 5th ACM Multimedia Systems Conference*, MMSys '14, pages 248–258, New York, NY, USA. ACM, 2014.

[83] TP-Link. TL-WDN4200 | N900 Wireless Dual Band USB Adapter | TP-Link. Website. URL: `https://www.tp-link.com/en/products/details/TL-WDN4200.html` (accessed on August 6, 2018).

[84] Linux Foundation Collaborative Projects. Open vSwitch. Website. URL: `http://www.openvswitch.org` (accessed on August 6, 2018).

[85] C. Liu, I. Bouazizi, and M. Gabbouj. Rate Adaptation for Adaptive HTTP Streaming. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, pages 169–174, New York, NY, USA. ACM, 2011.

[86] C. Liu, I. Bouazizi, M. M. Hannukselab, and M. Gabbouj. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Signal Processing: Image Communication*, 27(4):288–311, 2012.

[87] A. Mansy, M. Fayed, and M. Ammar. Network-layer Fairness for Adaptive Video Streams. In *Proceedings of the IFIP Networking Conference*, IFIP Networking '15, pages 1–9, Toulouse, France. IEEE, 2015.

[88] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, New York, NY, USA. ACM, 2017.

[89] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Compututer Communications Review*, 38(2):69–74, 2008.

[90] B. Meixner, J. W. Kleinrouweler, and P. Cesar. 4G/LTE Channel Quality Reference Signal Trace Data Set. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 387–392, New York, NY, USA. ACM, 2018.

[91] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. Adaptation Algorithm for Adaptive Streaming over HTTP. In *Proceedings of the 19th International Packet Video Workshop*, PV '12, pages 173–178, Munich, Germany. IEEE, 2012.

[92] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. QDASH: A QoE-aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, pages 11–22, New York, NY, USA. ACM, 2012.

[93] H. Nam, K. Kim, J. Y. Kim, and H. Schulzrinne. Towards QoE-aware video streaming using SDN. In *Proceedings of the Global Communications Conference*, pages 1317–1322, Austin, TX, USA. IEEE, 2014.

[94] Netflix. Protecting Netflix Viewing Privacy at Scale. Netflix Technology Blog, 201. URL: `https://medium.com/netflix-techblog/protecting-netflix-viewing-privacy-at-scale-39c675d88f45` (accessed on July 12, 2018).

[95] Netflix Help Center. Internet Connection Speed Recommendations, 2019. URL: `https://help.netflix.com/en/node/306` (accessed on February 19, 2019).

[96] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar. Controlled Delay Active Queue Management. Internet Draft, 2016. URL: `https://tools.ietf.org/html/draft-ietf-aqm-codel-03` (accessed on February 19, 2019).

[97] Node.js. Website. URL: `https://nodejs.org/en/` (accessed on August 13, 2018).

[98] Open Networking Foundation. Software Defined Networking (SDN). Website, 2016. URL: `https://www.opennetworking.org` (accessed on July 13, 2018).

[99] D. P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.

[100] R. Pan, P. Natarajan, F. Baker, G. White, B. VerSteeg, M. Prabhu, C. Piglione, and V. Subramanian. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. Internet Draft, 2016. URL: `https://tools.ietf.org/html/draft-ietf-aqm-pie-06` (accessed on February 19, 2019).

[101] R. Pantos and W. May. HTTP Live Streaming. Request for Comments, 2017. URL: `https://tools.ietf.org/html/rfc8216` (accessed on May 6, 2019).

[102] S. Petrangeli, J. Famaey, M. Claeys, S. Latré, and F. De Turck. QoE-Driven Rate Adaptation Heuristic for Fair Adaptive Video Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications*. TOMM, 12(2):28:1–28:24, 2015.

[103] S. Petrangeli, J. V. D. Hooft, T. Wauters, and F. D. Turck. Quality of Experience-Centric Management of Adaptive Video Streaming Services: Status and Challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(2s):31:1–31:29, May 2018.

[104] S. Petrangeli, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck. Software-defined network-based prioritization to avoid video freezes in HTTP adaptive streaming. *International Journal of Network Management*, 26(4):248–268, 2016.

[105] POX is a Networking Software Platform Written in Python. Website. URL: `https://github.com/noxrepo/pox` (accessed on August 6, 2018).

[106] W. Pu, Z. Zou, and C. W. Chen. Video adaptation proxy for wireless Dynamic Adaptive Streaming over HTTP. In *Proceedings of the International Packet Video Workshop*, PV '12, pages 65–70. IEEE, 2012.

[107] Y. Qi and M. Dai. The Effect of Frame Freezing and Frame Skipping on Video Quality. In *Proceedings of the 2006 International Conference on Intelligent Information Hiding and Multimedia*, pages 423–426, December 2006.

[108] B. Rainer, S. Lederer, C. Müller, and C. Timmerer. A seamless Web integration of adaptive HTTP streaming. In *Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 1519–1523, Bucharest, Romania. IEEE, 2012.

[109] V. Ramamurthi and O. Oyman. Video-QoE aware radio resource allocation for HTTP adaptive streaming. In *Proceedings of the International Conference on Communications*, ICC '14, pages 1076–1081, Sydney, NSW, Australia. IEEE, 2014.

[110] V. Ramamurthi, O. Oyman, and J. Foerster. Video-QoE aware resource management at network core. In *Proceedings of the Global Communications Conference*, GLOCOM '14, pages 1418–1423, Austin, TX, USA. IEEE, 2014.

[111] Raspberry Pi Foundation. Raspberry Pi 2 Model B. Website. URL: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/ (accessed on August 6, 2018).

[112] Raspberry Pi Foundation. Raspberry Pi 3 Model B. Website. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (accessed on August 13, 2018).

[113] Regeling Gebruik van Frequentieruimte Zonder Vergunning en Zonder Meldingsplicht. Website, 2015. URL: http://wetten.overheid.nl/BWBR0036378/2016-12-28 (accessed on August 6, 2018).

[114] R. Ricci, E. Eide, and C. Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *The magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[115] D. C. Robinson, Y. Jutras, and V. Craciun. Subjective Video Quality Assessment of HTTP Adaptive Streaming Technologies. *Bell Labs Technical Journal*, 16(4):5–23, March 2012.

[116] R. Al-Saadi and G. Armitage. Dummynet AQM v0.2 – CoDel, FQ-CoDel, PIE and FQ-PIE for FreeBSD's ipfw/dummynet framework. CAIA Technical Report, 2016. URL: http://caia.swin.edu.au/reports/160418A/CAIA-TR-160418A.pdf (accessed on February 19, 2019).

[117] Sandvine, Inc. Global internet phenomena report - October 2018, 2018. URL: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf (accessed on September 15, 2019).

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

[119] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and M. Stiemerling. Real-Time Streaming Protocol Version 2.0. Technical report, December 2016.

[120] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld, and P. Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys Tutorials*, 17(1):469–492, 2015.

[121] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown. MPLS-TE and MPLS VPNS with Openflow. *SACM SIGCOMM Computer Communication Review*, 41(4):452–453, 2011.

[122] K. D. Singh, Y. Hadjadj-Aoul, and G. Rubino. Quality of Experience estimation for Adaptive HTTP/TCP video streaming using H.264/AVC. In *CCNC - IEEE Consumer Communications & Networking Conference*, Las Vegas, United States, January 2012.

[123] Sintel, the Durian Open Movie Project. Website. URL: `https://durian.blender.org` (accessed on August 6, 2018).

[124] R. K. Sitaraman. Network performance: Does it really matter to users and by how much? In *2013 Fifth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10, January 2013.

[125] A. Sobhani, A. Yassine, and S. Shirmohammadi. A Video Bitrate Adaptation and Prediction Mechanism for HTTP Adaptive Streaming. *ACM Transactions on Multimedia Computing, Communications, and Applications*. TOMM, 13(2), March 2017.

[126] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, San Francisco, CA, USA. IEEE, 2016.

[127] K. Spiteri, R. Sitaraman, and D. Sparacio. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 123–137, New York, NY, USA. ACM, 2018.

[128] T. Stockhammer. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, pages 133–144, New York, NY, USA. ACM, 2011.

[129] B. Taani and R. Zimmermann. Spatio-Temporal Analysis of Bandwidth Maps for Geo-Predictive Video Streaming in Mobile Environments. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, pages 888–897, New York, NY, USA. ACM, 2016.

[130] S. Tavakoli, S. Egger, M. Seufert, R. Schatz, K. Brunnström, and N. García. Perceptual Quality of HTTP Adaptive Streaming Strategies: Cross-Experimental Analysis of Multi-Laboratory and Crowdsourced Subjective Studies. *IEEE Journal on Selected Areas in Communications*, 34(8):2141–2153, August 2016.

[131] TCPDUMP/LIBPCAP public repository. Website, 2018. URL: `https://www.tcpdump.org` (accessed on August 6, 2018).

[132] E. Thomas, M. van Deventer, T. Stockhammer, A. Begen, and J. Famaey. Enhancing MPEG dash performance via server and network assistance. *IET Conference Proceedings*, 2015.

[133] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 109–120, New York, NY, USA. ACM, 2012.

[134] U.S. Navel Research Laboratory. Common Open Research Emulator (CORE). Website. URL: https://www.nrl.navy.mil/itd/ncs/products/core/ (accessed on July 30, 2018).

[135] J. D. Vriendt, D. D. Vleeschauwer, and D. Robinson. Model for estimating QoE of video delivered using HTTP adaptive streaming. In *Proceedings of the International Symposium on Integrated Network Management*, IM '13, pages 1288–1293, Ghent, Belgium. IFIP/IEEE, 2013. ISBN: 978-3-901882-50-0.

[136] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. *ACM Transactions on Multimedia Computing, Communications, and Applications*. TOMM '08, 4(2):16:1–16:22, 2008.

[137] C. Wang, H. Kim, and R. Morla. QoE Driven Server Selection for VoD in the Cloud. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 917–924, June 2015.

[138] C. Wang, A. Rizk, and M. Zink. SQUAD: a spectrum-based quality adaptation for dynamic adaptive streaming over HTTP. In *Proceedings of the 7th International Conference on Multimedia Systems*, MMSys '16, New York, NY, USA. ACM, 2016.

[139] Q. Wang, K. Xu, R. Izard, B. Kribbs, J. Porter, K.-C. Wang, A. Prakash, and P. Ramanathan. GENI Cinema: An SDN-Assisted Scalable Live Video Streaming Service. In *Proceedings of the 2014 IEEE 22Nd International Conference on Network Protocols*, ICNP '14, pages 529–532, Washington, DC, USA. IEEE Computer Society, 2014.

[140] Wikipedia, The Free Encyclopedia. Settlement hierarchy. Website, 2017. URL: https://en.wikipedia.org/wiki/Settlement_hierarchy (accessed on May 6, 2019).

[141] P. Wisniewski, A. Beben, J. M. Batalla, and P. Krawiec. On delimiting video rebuffering for stream-switching adaptive applications. In *2015 IEEE International Conference on Communications*, ICC '15, pages 6867–6873, London, UK. IEEE, 2015.

[142] X. Xie, X. Zhang, S. Kumar, and L. E. Li. pistream: Physical layer informed adaptive video streaming over LTE. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 413–425, New York, NY, USA. ACM, 2015.

[143] P. K. Yadav, A. Shafiei, and W. T. Ooi. QUETRA: A Queuing Theory Approach to DASH Rate Adaptation. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, pages 1130–1138, New York, NY, USA. ACM, 2017.

[144] J. Yao, S. S. Kanhere, and M. Hassan. Improving QoS in High-Speed Mobility Using Bandwidth Maps. *IEEE Transactions on Mobile Computing*, 11(4):603–617, 2012.

[145] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *SIGCOMM Computer Communication Review*, 45(4):325–338, 2015.

[146] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan. ASAP: Adaptive Stall-Aware Pacing for Improved DASH Video Experience in Cellular Networks. *Transactions on Multimedia Computing, Communications, and Applications*. TOMM, 14(3s), 2018.

[147] A. H. Zahran, J. J. Quinlan, K. K. Ramakrishnan, and C. J. Sreenan. SAP: Stall-Aware Pacing for Improved DASH Video Experience in Cellular Networks. In *Proceedings of the 8th ACM on Multimedia Systems Conference*, MMSys '17, pages 13–26, New York, NY, USA. ACM, 2017.

[148] C. Zhou, C. Lin, and Z. Guo. mDASH: A Markov Decision-Based Rate Adaptation Approach for Dynamic HTTP Streaming. *IEEE Transactions on Multimedia*, 18(4):738–751, 2016.

[149] C. Zhou, C. Lin, X. Zhang, and Z. Guo. TFDASH: A Fairness, Stability, and Efficiency Aware Rate Control Approach for Multiple Clients Over DASH. *Transactions on Circuits and Systems for Video Technology*, 29(1):198–211, 2019.

[150] M. Zink, J. Schmitt, and R. Steinmetz. Layer-Encoded Video in Scalable Adaptive Streaming. *IEEE Transactions on Multimedia*, 7(1):75–84, 2005.