UNIVERSITÄT
DES
SAARLANDES

Towards Enabling Cross-layer Information Sharing to Improve Today's Content Delivery Systems

Dissertation zur Erlangung des Grades des Doktors der Ingenieurwissenschaften

der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

vorgelegt von

Mirko R. Palmer

Saarbrücken, 2022

Kolloquium

Datum:	02. März 2023
Dekan der Fakultät MI:	Univ.-Prof. Dr. Jürgen Steimle

Prüfungsausschuss

Der Vorsitzende:	Prof. Dr. Ingmar Weber
Die Berichterstatter:	Prof. Dr. Anja Feldmann
	Prof. Dr. Balakrishnan Chandrasekaran
	Prof. Dr. Oliver Hohlfeld
Der akademische Mitarbeiter:	Dr. Jialong Li

This work was supported in part by the SupraCoNex (16KIS1194) project.

Abstract

Content is omnipresent and without content the Internet would not be what it is today. End users consume content throughout the day, from checking the latest news on Twitter in the morning, to streaming music in the background (while working), to streaming movies or playing online games in the evening, and to using apps (e.g., sleep trackers) even while we sleep in the night. All of these different kinds of content have very specific and different requirements on a transport—on one end, online gaming often requires a low latency connection but needs little throughput, and, on the other, streaming a video requires high throughput, but it performs quite poorly under packet loss. Yet, all content is transferred opaquely over the same transport, adhering to a strict separation of network layers. Even a modern transport protocol such as Multi-Path TCP, which is capable of utilizing multiple paths, cannot take the (above) requirements or needs of that content into account for its path selection. In this work we challenge the layer separation and show that sharing information across the layers is beneficial for consuming web and video content. To this end, we created an event-based simulator for evaluating how applications can make informed decisions about which interfaces to use delivering different content based on a set of pre-defined policies that encode the (performance) requirements or needs of that content. Our policies achieve speedups of a factor of two in 20% of our cases, have benefits in more than 50%, and create no overhead in any of the cases. For video content we created a full streaming system that allows an even finer grained information sharing between the transport and the application. Our streaming system, called *VOXEL*, enables applications to select dynamically and on a frame granularity which video data to transfer based on the current network conditions. *VOXEL* drastically reduces video stalls in the 90th-percentile by up to 97% while not sacrificing the stream's visual fidelity. We confirmed our performance improvements in a real-user study where 84% of the participants clearly preferred watching videos streamed with *VOXEL* over the state-of-the-art.

Zusammenfassung

Inhalte sind allgegenwärtig und ohne Inhalte wäre das Internet nicht das, was es heute ist. Endbenutzer konsumieren Inhalte von früh bis spät—es beginnt am Morgen mit dem Lesen der neusten Nachrichten auf Twitter, dem online hören von Musik während der Arbeit, wird fortgeführt mit dem Schauen von Filmen über Online-Streaming Dienste oder dem spielen von Mehrspieler Online Spielen am Abend, und sogar dem, mit dem Internet synchronisierten, Überwachens des eigenen Schlafes in der Nacht. All diese verschiedenen Arten von Inhalten haben sehr spezifische und unterschiedliche Ansprüche an den Transport über das Internet—auf der einen Seite sind es Online Spiele, die eine sehr geringe Latenz, aber kaum Durchsatz benötigen, auf der Anderen gibt es Video-Streaming Dienste, die einen sehr hohen Datendurchsatz benötigen, aber, sehr nur schlecht mit Paketverlust umgehen können. Jedoch werden all diese Inhalte über den selben, undurchsichtigen, Transportweg übertragen, weil an eine strikte Unterteilung der Netzwerk- und Transportschicht festgehalten wird. Sogar ein modernes Übertragungsprotokoll wie MPTCP, welches es ermöglicht mehrere Netzwerkpfade zu nutzen, kann die (oben genannten) Anforderungen oder Bedürfnisse des Inhaltes, nicht für die Pfadselektierung, in Betracht ziehen. In dieser Arbeit fordern wir die Trennung der Schichten heraus und zeigen, dass ein Informationsaustausch zwischen den Netzwerkschichten von großem Vorteil für das Konsumieren von Webseiten und Video Inhalten sein kann. Hierzu haben wir einen Ereignisorientierten Simulator entwickelt, mit dem wir untersuchten wie Applikationen eine informierte Entscheidung darüber treffen können, welche Netzwerkschnittstellen für verschiedene Inhalte, basierend auf vordefinierten Regeln, welche die Leistungsvorgaben oder Bedürfnisse eines Inhalts kodieren, benutzt werden sollen. Unsere Regeln erreichen eine Verbesserung um einen Faktor von Zwei in 20% unserer Testfälle, haben einen Vorteil in mehr als 50% der Fälle und erzeugen in keinem Fall einen Mehraufwand. Für Video Inhalte haben wir ein komplettes Video-Streaming System entwickelt, welches einen noch feingranulareren Informationsaustausch zwischen der Applikation und des Transportes ermöglicht. Unser, *VOXEL* genanntes, System ermöglicht es Applikationen dynamisch und auf Videobild Granularität zu bestimmen welche Videodaten, entsprechend der aktuellen Netzwerksituation, übertragen werden sollen. *VOXEL* kann das stehenbleiben von Videos im 90%-Perzentil drastisch, um bis zu 97%, reduzieren, ohne dabei die visuelle Qualität des übertragenen Videos zu beeinträchtigen. Wir haben unsere Leistungsverbesserung in einer Studie mit echten Benutzern bestätigt, bei der 84% der Befragten es, im vergleich zum aktuellen Stand der Technik, klar bevorzugten Videos zu schauen, die über *VOXEL* übertragen wurden.

Acknowledgements

Doing a PhD is not an easy task. And I am not primarily talking about the academical challenges but dealing with the frustration of any failed submissions before finally publishing your precious work into which you put so much heart and effort. With this I want to thank my advisor Anja Feldmann for all the support you gave me. Working in your group was a privilege, as you give your students enough freedom to grow but always check that things do not go south. This means that one might get a, definitely well deserved, lecture of what one did wrong - in Germany we do call our advisor "Doktormutter" for a reason. Though, it also meant that you get an advisor who cares about your progression, even teaching you how to write better papers until deep in the night of the day of the deadline.

Thank you to Bala Chandrasekaran for teaching me that receiving even a bad paper review can be useful, for all the great discussions, and, most importantly, for showing me that the writing in research papers can be both functional and aesthetic. Thank you to Reese Enghardt for not only sharing an office with me in Berlin, a time I have fond memories of, but also for helping me get started as a PhD student at INET. Same goes to Philipp Tiesel, also for the joint work on our transfer simulator. Thank you to Thorben Krüger for sharing an office in Saarbrücken with me, for the delightful time I had doing pair programming with you and for helping me to grow as a person. Thank you to Malte Appel for the, likely, most productive time I had in my entire PhD, and for sharing the pain that taming the beast that the GQUIC code base was.

Thank you to my other collaborators: Kevin Spiteri and Ramesh K. Sitaraman - without collaborators like this, today's research would not be possible. In our research field the saying that something is more than the sum of its parts is very fitting because only by working together with great minds, we accomplish even greater things.

Thank you to all my colleagues at INET, many of which I call friends. Thank you to Florian Streibelt, Franziska Lichtblau, Thomas Krenc and Lars Prehn for the fun times hanging out with you. Thank you to Jawad Saidi for all the nice chats and never saying no when one needs help. A special thank you to all the admins back in Berlin and now here in Saarbrücken for always keeping the show running.

Thank you to my family who encouraged my passion for technology even from early age and for allowing me to not worry about anything but my passion. Thank you to all my friends from back in Berlin with whom I, as life unfortunately often goes, have less and less contact, but who are not forgotten.

Last, but most certainly not least, I would like to thank my girlfriend Susanne Dally for accompanying me, for always having my back and bearing with me, even when I was stressed out, yet again, because there was always another paper deadline.

Publications

This thesis was created with the help of contributions of our collaborations with several authors. Scientific work, in our discipline, is always a joint effort of combining expertise. The following list presents pre-published work, software and other collaborations of this author. A more detailed version of this author's main contributions are shown in Chapter 1.4.

Pre-published Papers

Parts of this thesis are based on the following peer-reviewed papers that have already been published.

Workshops

Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, Anja Feldmann
"The QUIC Fix for Optimal Video Streaming" In: Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC. *ACM EPIQ 2018*.
ACM, 2018, pp. 43-49. ISBN: `978-1-4503-6082-1`. DOI: `10.1145/3284850.3284857`.

International Conferences

Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, Ramesh K. Sitaraman
"VOXEL: cross-layer optimization for video streaming with imperfect transmission" In: Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies. *ACM CoNEXT 2021*.
ACM, 2021, pp. 359-374. ISBN: `978-1-4503-9098-9`. DOI: `10.1145/3485983.3494864`.

Internet-Drafts

Philipp S. Tiesel, Mirko Palmer, Balakrishnan Chandrasekaran, Anja Feldmann, Jörg Ott
"Considerations for Unreliable Streams in QUIC" Internet Draft draft-tiesel-quic-unreliable-streams-01, Oct. 2017.
https://tools.ietf.org/html/draft-tiesel-quic-unreliable-streams-01

Other collaborations

Philipp S. Tiesel, Theresa Enghardt, Mirko Palmer, Anja Feldmann
"Socket Intents: OS Support for Using Multiple Access Networks and its Benefits for Web Browsing"
Pre-print ArXiv, 2018, arXiv: `1804.08484`.

The Data Transfer Simulator from Chapter 3 was joint work with Philipp Tiesel.

Our frame importance ranking algorithm, explained in Chapter 5 was initially started as a collaboration with Malte Appel in the context of his Master Thesis, which I co-supervised. The continuation of the work on our frame ranking was published by the author of this thesis as part of *VOXEL* [1], and is presented in this thesis.

Software

The following software developed as part of this work has been made publicly available:

VOXEL.
https://github.com/derbroti/VOXEL

Data Transfer Simulator.
https://github.com/fg-inet/dtsimulator

Contents

Chapter 1

Introduction

Bill Gates' essay "Content is King" [2] proved to be well-founded. Innovation on the Internet was, according to Gates, and continues to be, shaped by answering the question of how to quickly, efficiently and reliably distribute content to billions [3] of people around the world. Content, in this case, is everything from small private homepages, posts on social media, to videos hosted on huge commercial streaming platforms. With video content as the largest[1] contributor to traffic on the Internet [5], the attention of many engineers and researchers is given towards video delivery. Many aspects of video delivery, and with that many aspects of the Internet itself, have improved significantly over time. These improvements range from the locality of video data with Content Delivery Networks (CDNs), the quality of video encodings with new codecs, to the quality of transferring video data with Adaptive Bitrate (ABR) Algorithms. For all the benefits these innovations brought to the Internet, it is impractical to repeat this process for any new, let alone all existing, types of content. We could, though, allow applications to influence the way content is delivered—an application knows exactly what type of content is to be transferred, but we are stuck with a many decades old layered internet architecture. Could we do better if we would allow a breaking of the layer separation? Answering the question if breaking the separation, or, in other words, if allowing an information sharing across layers is beneficial, is the focus of this thesis.

In order to optimize any content delivery by cross-layer information sharing, we first have to look at the kind of devices that are used to consume content. With an estimated $4.7B$ 4G subscriptions [6], mobile internet devices are more common than ever. Next, we have to ask how these mobile devices are connected to the internet? Mobile devices can, today, usually access the Internet over more than one access network, as they often have the choice between WiFi and a cellular network. Data in mobile networks is, though, still a sparse [7], imperfect [8] and expensive [9] resource. Having a system to utilize the available resource efficiently is, thus, very significant. A bulk download,

[1]Researchers forecast that video will account for 82% of all IP traffic by 2022 [4].

e.g., an automatic software update, that is not aware of a parallel video stream, can degrade an end-user's quality-of-experience. It is not enough, though, to simply coordinate between applications, the transport needs to be managed as well. Applications know their needs and intents, e.g., a background download neither needs low latency, nor a high throughput connection, an instant messenger needs low latency, a live video stream needs both. The transport layer knows the availability of links and their characteristics, e.g., which link does fit the needs of an application. Such a system could, for streaming video, with the help of systems like MPTCP, select a lower capacity, low latency connection for a fast video startup. The system could then then switch to a high throughout link or bond multiple links for the highest possible video quality, while throttling, currently unimportant, background downloads. Despite the possible benefits, the available diversity is usually not exploited. While the application knows its demands, selecting the path and destination within the application is impractical most of the time. This impracticability stems from the required selection information, which is often not available. The selection information encompasses detailed characteristics of the available paths or information about cross-traffic from other applications.

One approach for enabling applications to actually exploit the available paths is Socket Intents [10]. To first investigate feasibility, we evaluated a Socket Intents equipped system in the general area of web traffic. Unlike video streaming, web traffic has less strict real-time requirements and is, thus, easier to evaluate. It is noted, though, that Socket Intents can indeed be used for video streaming applications as shown by Enghardt et. al. [11].

With a system in place to communicate across layers, i.e., between application and transport layer, we can, next, take the cross-layer approach even further with video streaming. Compared to web traffic, the continuous playback of streamed video has a real-time requirement in the form of constantly needing fresh data from the video player. In other words, if a video frame arrives late, i.e., after it was supposed to be shown to the user, for example due to head-of-line blocking or retransmissions, it becomes useless and the playback stalls.

Late frames, resulting in playback stalls, are, for end users, the most detrimental aspect of streaming video [12–14]. Tackling the challenging task of streaming video in varying, and often less-than-ideal, network conditions while keeping end-users' Quality of Experiences (QoEs) unaffected is, hence, more dire than ever before [15–17]. To avoid stalls, or rebuffering, reducing the required amount of data is presumably one of the most viable solutions. Today, adaptive-bitrate algorithms, which in regular intervals switch between downloading different encoding versions of a video, adapt the required data to the available link capacity. This process suffers from a quantization problem. Without excessive storage requirements, there can only be a limited number of discrete encoding qualities. There encoding, or quality levels, can not perfectly match a continuous signal like throughput. We propose a system that can match the throughput signal. By analyzing a video, we can determine which frames are essential to not degrade an end-users' experience. Video data is highly compressed, how can anything not be essential? Video codecs are not perfect, a significant amount of still redundant video data can be dropped without users noticing it. If you allow minor visual glitches, this number is even higher. Though, this information is only available on the application layer, thus, we need a tight coupling between the application and the transport. A tight coupling would allow an application to not only influence the link selection but to determine data importance, i.e., if lost data is worth retransmitting or if the data is even required.

1.1 Problem Definition

Transferring content is a hard problem, especially if the available network connectivity is less-than-ideal. This problem is exacerbated when there is more than one network connection to chose from, i.e., the transport layer has information about link characteristics but can not communicate this information to the application.

While the application has knowledge about the type of content to transfer, it can not communicate this information back to the transport layer. We want to investigate the possibility to improve the performance of content delivery by breaking the traditional layered approach of networking. In other words, we ask if a cross-layer information sharing can improve today's content delivery systems.

The largest contributors to the overall traffic on the internet [18], are video streaming and web traffic. Web sites have many objects in various sizes [19, 20] and should, thus, be a prime candidate to investigate if an information sharing and a resulting network connection selection is indeed beneficial. Video streaming, the largest traffic contributor, has real-time requirements, i.e., video frames need to be delivered in time or are wasted bandwidth as they are unusable by the already moved on video player. These real-time requirements would need an even tighter coupling between the network layers to not only select a connection per object, or video, but per individual video frame.

1.2 Approach

We start with Socket Intents, a relatively coarse-grained approach towards cross-layer communication between the transport and the application layer. We take advantage of today's end-user devices' capability of connecting to multiple access networks by using Multi-Path TCP. We revisit the design space of video streaming systems with a novel view of imperfect transmissions by coupling the application and transport even tighter. To make informed decisions about how such an imperfect transmission would work, we get a better understanding of video streaming and specifically video codecs themselves. We make use of recent, in user-space implemented, transport protocols like Quick UDP Internet Connections (QUIC) that open up new possibilities to interact and alter the transport from an application. Finally, we design and evaluate a system, we call *VOXEL*, that allows cross-layer interactions and exploits video characteristics.

1.3 Contributions

This thesis presents work covering the following aspects:

Socket Intents. We evaluate our system based on the use case of Web browsing: Using a flow-based simulator and a full factorial experimental design, we study a broad range of access network combinations (based on typical Digital subscriber line (DSL) and Long-Term Evolution (LTE) scenarios) and real workloads (Alexa Top 100 and Top 1000 Web Sites). Our policies achieve performance benefits in more than 50% of the cases and speedups of more than factor two in 20% of the cases without adding overhead in the other cases.

VOXEL. We propose *VOXEL*, a cross-layer optimization system for video streaming. We use *VOXEL* to demonstrate how to combine application-provided "insights" with a partially reliable protocol for optimizing video streaming. To this end, we present a novel ABR algorithm that explicitly trades off losses for improving end-users' video-watching experiences.

VOXEL is fully compatible with DASH, and backward-compatible with *VOXEL*-unaware servers and clients. In our experiments emulating a wide range of network conditions, *VOXEL* outperforms the state-of-the-art: We stream videos in the 90th-percentile with up to 97% less rebuffering than the state-of-the-art without sacrificing visual fidelity. We also demonstrate the benefits of *VOXEL* for small-buffer regimes like the emerging use case of low-latency and live streaming. In a survey of 54 real users, 84% of the participants indicated that they prefer videos streamed using *VOXEL* compared to the state-of-the-art.

1.4 Pre-published Work and Collaborations

As early as 1963, Derek de Solla Price [21] showed a trend towards collaborative research work with an expected extinction of single-author publications by 1980. We want to emphasize that this trend prevails in our discipline's scientific work. This thesis is, thus, naturally based on collaborations and pre-published work with several authors.

In the following, we will outline the main contributions of this thesis' author to the work used in this thesis.

Chapter 3: This chapter lays the groundwork for the idea of the need for an information sharing across layers. The concept of Socket Intents, used here, was originally presented by Schmidt et. al. [10], and extended, in collaboration with this author, by Tiesel et. al. [22]. The main contributions of this author are (a) extending the existing prototype to work with MPTCP, (b) the joint implementation of the event simulator used for evaluation the concept, and (c) the *Earliest Arrival First* policy also used in the evaluation.

Chapter 4: The insights presented in this chapter originated from this thesis' author and are used in Palmer et. al. [1].

Chapter 5: This chapter extends on the frame importance ranking started in collaboration with Malte Appel in his master thesis.

Chapter 6: Here, we present *VOXEL* by Palmer et. al., published at CoNEXT 2021. As the main author of *VOXEL*, this author's contributions are (a) the overall concepts used, (b) a joint implementation of the system, which is published by the author as open-source, and (c) a joint evaluation of the system.

Chapter 7: Lastly, in this chapter, the author concludes this thesis in a summary and an outlook towards future work.

1.5 Structure of this Thesis

The remainder of the thesis is structured as follows.

Chapter 2 introduces background knowledge about sockets and socket intents, video coding and streaming, and approaches and metrics to determine the perceived quality of a video.

Chapter 3 investigates how to incorporate a multi-path transport into an existing multi-access prototype. We create an event simulator to analyze the performance of informed applications when delivering content, specifically web traffic.

Chapter 4 moves the focus towards another prevalent type of content, namely video. We study how video codecs operate and show that even though their compressed nature that there is still room for reducing file sizes, by dropping frames, which can alleviate the load on a network.

Chapter 5 takes the insights of dropping frames and we create "frame-orders" which sort video frames by importance to enable gracefully terminating video downloads prematurely while still guaranteeing a high perceived video quality.

Chapter 6 completes our frame-order approach into a full-fledged cross-layer video streaming system we call *VOXEL*. We explain in detail how *VOXEL* is created and performed an extensive performance analysis.

Chapter 7 finally concludes the thesis with a summary of our findings and provides an outlook towards future work.

Chapter 2

Background

In this section we will review the basic concepts involved in content consumption over the internet from the point of view of the transport layer and the application layer. We start with *Sockets*, the standard communication interface between applications and the operating system's network stack. Following, we explain how sockets are extended to allow an application to convey an intended use of a socket and discuss the challenges involved. Last in the transport, we show the basics of Multi-path TCP (MPTCP), a Linux kernel extension to TCP that allows to utilize multiple network interfaces simultaneously and how we combined it with socket intents.

Moving on to the application layer we explain how today's video streaming works on top of a transport. We start with video codecs themselves and how to make use of different encodings to generate video qualities that fit different network characteristics, i.e., generating a lower bitrate video requires less network throughput. Finally, we will go over how video quality can be measured and what care needs to be taken when comparing a video to a reference.

2.1 Sockets

In Unix-like OSes, BSD Sockets are the standard interface between applications and the network stack. Typically, applications that want to connect to a server first resolve the server's hostname using `getaddrinfo()`, then create a socket file descriptor using `socket()` and call `connect()` to establish the connection. To each of these calls information obtained from `getaddrinfo()` is passed.

With Vanilla BSD Sockets, taking advantage of multiple paths or choosing among several destinations is complicated. One reason is that the BSD Socket API designers considered multi-homed hosts a corner case. The `bind()` socket call allows applications to choose the source address of an outgoing communication[1]. If the system is configured with a routing policy to send traffic with a specific source address over an associated paths, application can set the source address to implicitly choose the outgoing interface and next-hop and, therefore, large portions of the path.

Besides this hack, Vanilla BSD Sockets do not offer support for multiple access networks: Applications that want to use multiple interfaces usually have to have their own heuristics for selecting paths. Choosing among paths is difficult as the necessary information is often difficult to gather and may require special privileges. Moreover, it differs greatly by Unix flavor.

2.2 Socket Intents Concept

To perform path and destination selection within the OS, the OS needs to know what to optimize for – the application demands. Therefore, we introduced the concept of *Socket Intents* [23]. Socket Intents allow applications to share their knowledge about their communication patterns and express performance preferences in a generic and portable way. Intents are hints for the OS, pieces of information, that allow an application programmer to express what they know about the application's needs or intentions for each communication unit. They indicate what the application wants to achieve, knows, or assumes. In contrast to transport features or QoS-style reservations, they are not requirements but only considered in a *best-effort* manner, e.g., as input to path and destination selection heuristics within the OS. Possible intents, as shown in Table 2.1, include *Traffic Category*, *Size to be Sent/Received*, *Timeliness*, *Duration*, or *Resilience of connectivity*. Applications have an incentive to specify their intents as accurately as possible to take advantage of the most suitable resources. We expect applications to selfishly specify their preferences. Since the OS knows about the available network resources and the intents of multiple applications, it can balance the different requirements and penalize misbehaving applications.

Socket Intents are independent of the actual Socket API and can be applied to message granularity communications, e.g., UDP messages or HTTP requests, as well as stream granularity communications, e.g., TCP connections. The information provided by the application is structured as key-value-pairs. The key is a simple string representing the type of a Socket Intent. Values can be represented as an *enum*, *int*, *float*, *string*, or a sequence of the aforementioned data types. Table 2.1 gives an overview of Socket Intent types as specified in our recent IETF draft [24]. Despite the variety of Intents we define in this section, the remainder of this paper focuses on how to realize Socket Intents as

[1]Otherwise, the OS uses the IP address of the interface via which it routes to the given destination.

an extension to the BSD Socket API and the benefits of using the *Size to be Received Intent*.

2.3 Challenges Imposed by BSD Sockets

With Vanilla BSD Sockets, taking advantage of multiple paths or choosing among several destinations is complicated. One reason is that the BSD Socket API designers considered multi-homed hosts a corner case. The `bind()` socket call allows applications to choose the source address of an outgoing communication[2]. If the system is configured with a routing policy to send traffic with a specific source address over an associated paths, application can set the source address to implicitly choose the outgoing interface and next-hop and, therefore, large portions of the path.

Besides this hack, Vanilla BSD Sockets do not offer support for multiple access networks: Applications that want to use multiple interfaces usually have to have their own heuristics for selecting paths. Choosing among paths is difficult as the necessary information is often difficult to gather and may require special privileges. Moreover, it differs greatly by Unix flavor.

Another complication occurs when selection a destination: When resolving the hostname to obtain a destination address, applications need to ensure not to mix results for the same hostname resolved via a different interface. For example, CDNs and major Web sites often rely on Domain Name System (DNS)-based server selection and load balancing. These mechanisms are most useful if the DNS query is sent via the same interface as the actual traffic. If the application sends the traffic over another interface, the chosen server may be suboptimal, which can lead to significant performance degradation. Yet, the resolver library of the vanilla BSD Socket API does not allow us to isolate results acquired via multiple paths. For a more detailed discussion, see [25].

Furthermore, the communication units used by vanilla BSD Socket API are implied by the transport protocol and must match the socket type passed to the `socket()` call. Thus, for stream-based communication protocols like TCP, the application can only choose a path and endpoint for the whole stream. But communication units of actual applications are often not aligned with the communication granularity of the transport protocol. For example, requests in HTTP —the dominant protocol on the Internet [26, 27]— correspond to a message based communication performed over a stream transport. An HTTP based application can choose for each request to either open a new TCP connection or reuse an existing one. The former allows choosing

[2]Otherwise, the OS uses the IP address of the interface via which it routes to the given destination.

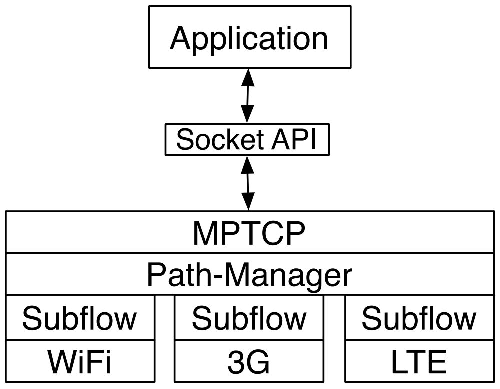


Figure 2.1: *Structured overview of an MPTCP connection on a device with three network interfaces, i.e., WiFi, 3G and LTE. The individual* sub-flows *are full-fledged TCP connections, isolated from the application, initiated by the MPTCP kernel code and controlled by the path-manager.*

among multiple interfaces using `bind()`. The latter saves 2 round-trip times (RTTs) for the TCP handshake, a few 100 KB for the Transport Layer Security (TLS) handshake (if applicable), and time spent in TCP slow-start. Therefore, the abstraction provided by the vanilla BSD Sockets does not assist the application in distributing traffic among multiple paths. Rather, it puts a huge burden on applications that want to do so.

In conclusion, these problem areas demonstrate that the vanilla BSD Socket API is not well suited to enable multiple access connectivity in an easy and portable way.

2.3.1 Multi-Path TCP

MPTCP is a set of kernel extensions to regular TCP[28]. It allows TCP connections to use multiple paths simultaneously. This means a single TCP connection is split up into several *sub-flows*, which represent full-fledged TCP connections by themselves which can be routed independently (see Figure 2.1). The possible gains are manifold. In a data center that is connected to multiple upstream providers, a server can increase its maximum throughput by simultaneously creating and maintaining multiple connections to a single endpoint. On the other end, mobile phones and an increasing number of notebooks have multiple network interfaces, which can be bundled to achieve better Internet connectivity without connection losses. This is due to MPTCP's ability of adding and removing sub-flows on demand, meaning that interfaces can be removed without affecting the connection as long as a single sub-flow is available. However, for this to work both ends of the communication need to be MPTCP-capable and no middle-box may alter the packets in a way that removes the MPTCP-added information.

In general MPTCP establishes multiple TCP connections (including the 3-way handshake) and schedules the transmission of packets through all of them. In the current implementation, any application that wants to utilize MPTCP does not have to be modified but in turn is not able to interface with or control MPTCP. This is possible since applications use the normal socket API, but a MPTCP enabled kernel adds the TCP option *MP_CAPABLE* into the SYN packet in order to negotiate multi-path capabilities [29]. Additionally a hash (token) is calculated from random keys that were exchanged during the connection (session) establishment to be able to identify connections. Now, if client and server are multi-path capable, sub-flows will be created by the path-manager, using the token to identify membership to a session, by establishing new TCP connections equipped with the *MP_JOIN* option.

MPTCP and the TCP stack are heavily interwoven. When a new session is established, the MPTCP kernel code copies all relevant information (e.g. *struct sock*) from the client socket to its internal data structure (e.g. *struct mptcp_cb*). This structure contains the identifying token, a list of sub-flows, the master socket, meaning the socket that the application sees, and all interfaces (local and remote) that can be used by MPTCP.

Path-managers are part of the MPTCP kernel modules that control how new sub-flows are created. There are three path-managers: *full-mesh*, *ndiffports* and *binder*. *ndiffports* is primarily used for testing purposes, as it does not utilize the ability of MPTCP to use multiple interfaces. It is nevertheless a convenient tool for specific scenarios, for example a link which limits the bandwidth of single TCP connections. It uses one interface and creates all sub-flows on it using different ports. *binder* is the newest path-manager that implements source routing for community networks [30], making it only useful for special purposes. The last path-manager, which is used in this thesis, is *full-mesh*. It is the most general purpose path-manager available. It creates a full mesh of sub-flows on all available local and remote interfaces and uses the structure *mptcp_cb* to determine which interfaces are available for establishing new connections. Each sub-flow is created in kernel space, meaning without the socket API, and is identifiable via the hash token.

2.3.2 Socket Intents with Multi-Path TCP

MPTCP is an orthogonal approach to Socket Intents to take advantage of multiple access networks. MPTCP splits a TCP stream across multiple network interfaces and can achieve almost the combined bandwidth for large transfers.

Since Socket Intents operate on request/response pairs and MPTCP offers a connection abstraction we can combine both approaches and (a) make MPTCP available via Socket Intents and (b) enable the Socket Intents Policy to control the usage of MPTCP.

Table 2.1: *Socket Intents Types*

Intent Type	**Data Type**	**Applicable Granularity Message**	**Stream**
Traffic Category	Enum		✓
Size to be Sent	Int (bytes)	✓	✓
Size to be Received	Int (bytes)	✓	✓
Duration	Int (msec)		✓
Bitrate Sent	Int (bytes/sec)		✓
Bitrate Received	Int (bytes/sec)		✓
Burstiness	Enum		✓
Timeliness	Enum	✓	✓
Disruption Resilience	Enum	✓	✓
Cost Preferences	Enum	✓	✓

As a result, Socket Intents promises good results for both small communication units—which the policy can distribute—as well as large ones—which MPTCP can handle.

To achieve (a) it is sufficient to use an MPTCP enabled kernel. To support (b) we modified the Linux MPTCP implementation to enable communication between the *Socket Intents Policy Manager* in user-space and the MPTCP path-manager in kernel-space via Netlink [31] sockets. If a policy decides to use MPTCP it selects an interface for the initial TCP connection of MPTCP. If MPTCP is available the path-manager notifies the policy, which can then add additional connections—the sub-flows. MPTCP can then distribute the TCP stream over all chosen interfaces. For more details on our implementation, we refer to [32]. Using MPTCP allows more fine-grained bandwidth sharing than on a per request/response basis while the policy can circumvent the head-of-line blocking problem of MPTCP for small objects.

2.4 Video Streaming

Streaming video over HTTP typically entails using either DASH [33] or HTTP live streaming (HLS) [34]. Although DASH and HLS have similar requirements regarding the video format, we restrict our attention to the codec-agnostic DASH. The video data itself is usually encoded as H.264, the most widely used video codec [35, 36], and encapsulated in an MP4 container. The video file is split into equal-duration, typically 2-10 s long, *segments*. A *manifest* file specifies the names and locations of the video segments stored on the server, the available quality levels or bitrates per segment, encoding details, and other relevant metadata. Streaming via DASH begins with the client requesting the manifest file from the server [33]. To handle varying network conditions, clients utilize an ABR algorithm to determine which of the available quality levels of a segment to download.

2.5 Codecs

The H.264 codec defines three types of *Frames*: *Intra-coded (I)*, *Predicted (P)*, and *Bi-directional predicted (B)*. Prediction refers to a frame using other frame(s) as reference to reduce the amount of data it contains; the referrer only stores the difference with respect to the reference(s). References between frames are at macroblock granularity. I-Frames do not have references to any other frames, and can, hence, be rendered instantaneously by the client. A P-Frame, in contrast, depends on one or more previous frames, of any type, and a B-Frame depends on both previous and following frames. The loss of a frame that is referenced by many other frames introduces errors in decoding the referring frames, which in turn results in visible impairments.

Technically, the H.264 codec defines *slices*, which refer to spatially distinct regions of a frame, and predictions happen at the slice level [37]. Since frames in our videos consist of only one slice, we use the generic term, frames.

2.6 Adaptive Bitrate Algorithms

Adaptive Bitrate (ABR) algorithms guide the client in selecting a video quality level that is appropriate for the current network conditions (e.g., throughput). Quality levels correspond to bitrates that are known *a priori*. Thus, the algorithms help a client to determine the highest segment quality that can be downloaded *on time* (i.e., before it must be rendered by the client) under the estimated conditions. There are 3 categories of ABR algorithms: *throughput-*, *time-*, and *buffer-based*. Throughput-based ABR algorithms, (e.g., PANDA [38]), base their decision, on estimated network throughput. Time-based ABR algorithms, (e.g., ABMA+ [39]), use segment download times, while buffer-based algorithms rely solely on playback buffer occupancy [40]. Recent hybrid ABR algorithms (e.g., Model predictive control (MPC) [41] and the Buffer Occupancy based Lyapunov Algorithm (BOLA) [42]) combine throughput and buffer-based approaches to improve performance.

2.7 QoE Metrics

QoE metric are divided into two main categories: subjective and objective measures [43]. The former is used traditionally, e.g., in surveys, and, given their nature, allow direct measurement of people's sentiment towards, in this case, the quality of experience when watching a streamed video. Subjective tests, involving human test subjects are, though, expensive and time consuming. To automate and streamline quality assessment, objective metrics can be used, for which a model is created to estimates the responses of human subjects. The structural similarity (SSIM) is a widely used objective *full-reference* QoE metric that captures the visual quality of a video stream [44]. To calculate *SSIM scores*, each frame is compared to its *pristine* version, the difference is scored and averaged over all compared frames (e.g., a segment). Such an average is widely used and is well-suited for our evaluations: 3 low-score (e.g., SSIM=0.6) frames in a 4 s segment suffice to lower the total segment score to drop below excellent (e.g., SSIM< 0.99).

As with any metric, SSIM has its shortcomings. In our case, it may mask the loss, if any, of temporal smoothness of the video. Alternatives to SSIM, including ITU p.1203 [45] and, the more recent, VMAF [46], exist. Both VMAF and p.1203, however, use trained models to assess the visual quality of videos delivered *without* any loss (i.e., via reliable transport). Unlike SSIM, utilizing these models on videos with corrupted frames will result in *undefined* model behavior and may yield invalid results [45, 46], which is also confirmed by the authors of p.1203 [47]. We focus, hence, on SSIM in our tests, but *also* use Video Multi-Method Assessment Fusion (VMAF) and Peak signal-to-noise ratio (PSNR), wherever feasible, to demonstrate that the *Video-streaming Optimization Across Enriched Layers (VOXEL) is QoE-metric agnostic.*

SSIM, as a full-reference metric, requires a pristine reference copy of the material that is to be examined. Ideally this reference copy is uncompressed source video material, to not introduce any artifacts. We do not use the original uncompressed source video as reference, as compression-artifacts are orthogonal to this study. We measure, instead, the difference between the highest quality a user could see and the quality that they actually see. An ideal video playback at the highest encoding would, thus, result in a perfect QoE score. Any deviation from the highest quality would result in a reduced QoE which can be captured in the QoE score accordingly. The quality score, hence, indicates how close the stream's quality is to the highest feasible quality.

Low-resolution video will be scaled up to the native resolution of end-users' devices, potentially degrading visual quality. To accurately capture the QoE of users with high-resolution devices, we chose 4K as the pristine reference in the SSIM, VMAF, and PSNR calculations. We chose 4K because more than half of the TV's shipped in 2018 are native

4K [48] and half of YouTube's recommended mobile devices have $\geq$1440 p screens [49]. YouTube also recommends uploading content in up to 4K [50].

Chapter 3

Socket Intents

The separation of network layers makes it undesirable for an application to have knowledge about the underlying transport it uses. We argue that this separation is detrimental, e.g., end-user devices are equipped with multiple network interfaces, with vastly different characteristics and protocols like MPTCP exist, yet the transport is unable to communicate their capabilities to an application. Vice versa, a multitude of different kinds of content with different communication needs exist, e.g., receiving the latest tweets quickly requires a low latency connection but needs hardly any throughout, yet a background download needs as much bandwidth as possible, but not on a volume limited connection. Having transports with different characteristics which can not coordinate with applications with vastly different needs make it obvious that there is a need for sharing information across layers to improve content delivery.

We start tackling this information sharing issue with *Socket Intents* [23], an easy but coarse-grained approach.

Socket Intents, as described in §2.2, allow applications to share their knowledge about their communication pattern and express performance preferences in a generic and portable way. With Socket Intents, an application developer can inform the OS about what the *intended* of the communication is and what they know about the communication itself:

- **Preferences** to whether to optimize for bandwidth, latency, or cost.
- **Characteristics** of expected packet rates, bitrates or the size of the transferred content.
- **Expectations** towards path availability or packet loss.
- **Resilience** of the application against certain error cases.

Those intents are soft requirements, e.g., transport protocol guarantees or QoS style reservation. They are, however, crucial for the OS to do path and destination selection on behalf of the application.

We demonstrate, In the remainder of this chapter, how MPTCP can be integrated into Socket Intents on top of vanilla BSD Sockets by designing a system that enables automated multi-path and destination selection within the OS and evaluate the *impact of Socket Intents on Web performance.*

The existing prototype implementation [10] extends the BSD Socket API and demonstrates the feasibility of automated path and destination selection within the OS.

Our evaluation demonstrates that using applications' knowledge for multi-path selection can largely improve applications' performance. We, therefore, use the *Size to be Received Intent* of individual Web objects as input to our path selection logic for the Earliest Arrival First (EAF) policy and for multi-path selection for the Earliest Arrival First with MPTCP (EAF-MPTCP) policy. *EAF* predicts the arrival times of the requested objects for all available paths and assigns each request to the path it is predicted to complete on first.

Our main contributions, described in the following chapters, are:

- We extend the existing Socket Intents prototype [22, 23] to support and control MPTCP.
- We introduce the *EAF* and *EAF-MPTCP* policies as informed multi-path selection strategy for Web objects. Based on the information provided by the *Size to be Received Intent*, the policies assign each request to the set of paths it is predicted to complete on first.
- We study the benefits of using Socket Intents with the *EAF* policy in an extensive evaluation using a custom flow-based simulator. Our simulation utilizes a full factorial experimental design and covers the Alexa Top 100 and Top 1000 Web sites over a wide range of network characteristics resembling typical residential broadband and cellular network characteristics.

3.1 Combining Multi-Path TCP with Socket Intents

Modern end-user devices are always equipped with at least two network interfaces on different mediums, i.e., WiFi and cellular. The typical approach is to simple disable any cellular data if the device is connected to a WiFi network.

We extend the existing Socket Intents prototype to make use of MPTCP to split a TCP stream across multiple paths. This split allows bandwidth aggregation for large transfers and complements a per-request scheduling. As a result, Socket Intents can choose appropriate interfaces for both small communication units—which a policy can distribute appropriately—as well as large ones— which MPTCP can handle. This informed distribution of communication units via the Socket Intents Policy avoids opening, for example, MPTCP sub-flows on already crowded or high latency interfaces, and, thus, avoids head-of-line blocking for small objects.

On a high level, we added an additional path-manager to the Linux MPTCP implementation, to enable the Socket Intents Policy to control the usage of MPTCP. Our user-space Multi Access Manager uses Netlink [31] sockets to communicate with the kernel-space MPTCP path-manager. If a policy decides to use MPTCP it selects an interface for opening the TCP connection, which acts as the the initial MPTCP sub-flow. After the connection is fully established and the connection is MPTCP-capable, the path-manager notifies the Multi Access Manager, which starts the policy which chooses to add sub-flows on specific interfaces. MPTCP then distributes the TCP stream over all chosen interfaces.

In more detail, implementing the combining of the MPTCP framework and the Multi-Access Management (MAM) framework required a detailed understanding of how a MPTCP path-manager works. The existing path-managers, at the time, utilized all available network interfaces without any means to alter this behavior. In order to select a subset of all available interfaces, it was necessary to add means to control the path-manager to create sub-flows on demand. A controllable path-manager requires an interface to communicate with MAM and its decision making point, the Multi Access Manager to pass the selected interface subset to MPTCPs kernel-module.

As stated, netlink [51], a Linux standard inter-process communication (IPC) interface, was used as the communication channel between MAM in the user-space and MPTCP in the kernel-space.

MAM was originally designed to be stateless, though, in order to accommodate a delayed, or on-demand, creation of connections the information necessary to establish a connection, or specifically a sub-flow, needs to be cached. As MPTCP manages all sub-flows as one logical connection, connection handover is achieved by simply closing existing sub-flows and establishing new sub-flows on different interfaces. Although the initial sub-flow does not need to be handled differently, i.e. closing it, while another sub-flow is connected will keep the logical TCP connection established, it is not created in the MPTCP path-manager. The initial sub-flow is, instead, created via the main socket of the client application and binding it to the desired interface has to be handled by the MAM itself.

3.2 Policy Design

Integrating MPTCP into the MAM prototype requires MPTCP-aware policies. To create MPTCP-aware policies, we first have to discuss the policy design in general. Socket Intents Policies are entities that decide which access network to use for a given communication unit. They range from simple static configurations to complex dynamic algorithms that aim to take full advantage of all available information.

To make informed decisions, these Policies require knowledge about the intents of an application as well as interface parameters and statistics, including byte counters and transport protocol state. Within its decision logic, the policy needs to respect the optimization of external communication partners, i.e., to only rely on DNS replies from the same interface (see §2.3).

For the sake of simplicity, we chose not to support per-application policies, but rely on the information provided by Socket Intents. This allows us to treat communication units of a single application with different communication needs appropriately.

As a first application-aware policy, we introduce the *EAF* policy: This policy is based on the idea that downloading objects of different sizes can benefit from different path characteristics, as download time largely depends on the object's file size as well as the latency and throughput of the path. We use the *Size to be Received Intent*, which allows an application to provide hints for the expected size of a communication unit, e.g., an HTTP client may hint about the size of an object to be transferred. Assuming the availability of at least two access networks which vary in delay and bandwidth, our intuition is that if the communication unit is small, the policy should choose the interface with the lower latency. If the communication unit is large, the interface with the larger available throughput should be preferred. Each unit is, thus, scheduled on the interface with the earlier arrival time, which, in turn, contributes to a shorter overall completion time.

3.2.1 Implementation Considerations

A policy implements logic for interface selection, i.e., selecting an appropriate source and destination address pair. The actual Socket Intents Policy is implemented as modules of the Multi Access Manager. It is shared by all applications of a host that use the prototype's socket interface, as their individual needs are communicated using Socket Intents and are not realized via individual policies. A policy may pick a suitable interface for each individual communication unit. Given a set of open sockets and a chosen interface, for efficiency, a policy tries to select and, thus, reuse that socket that uses the chosen interface. If no set is given or if no suitable socket is found, the policy advises the application to open a new connection and suggests an IP address of the chosen interface as source. When a policy has decided on a source and destination address pair, it instructs the Multi Access Manager to send this information back to the socket library. With MPTCP, the policy may have to keep track of the requests to aid, e.g., the setup of MPTCP sub-flows. The Single Interface Policy, for example, always chooses a particular, statically configured interface. The Round Robin Policy uses multiple interfaces, one after the other, in a round robin fashion.

The more complex *EAF* policy uses the *Size to be Received Intent* to predict the completion time of a communication units on each available interface and chooses based on the arrival time. For *EAF* the Socket Intents Prototype uses estimates of the minimum smoothed round-trip times (SRTT) per prefix and the available bandwidth on the interface. *EAF* estimates the available throughput by dividing the maximum observed interface bandwidth by the number of already scheduled objects on that interface.

We divide the file size by the estimated available throughput to approximate the download duration. We add one RTT if a connection can be reused and two RTTs if a new connection has to be established[1]. Finally, the interface with the shortest predicted arrival time is chosen.

3.2.2 MPTCP-aware Policies

Using this knowledge of designing and implementing policies, we now describe policies that utilize MPTCP. The first MPTCP-aware MAM policy is the *default interface* policy. The default interface policy allows to select a single interface and establishes an MPTCP connection on it. The selected interface can then be changed at any point in the lifetime of the connection, e.g., when the network conditions on the interfaces changes. Consider a user starting an application on their phone at home which is connected to a WiFi and an LTE network. In case the mobile network connection is volume-limited, it is desired to exclusively use the WiFi connection. In case the user leaves their home, though, a seamless handover to the LTE network needs to happen to not interrupt the applications network connections. This handover can be realized with the default interface policy.

[1]We do not consider TLS handshakes.

The second implemented MPTCP-aware MAM policy is the *file-size* policy. The file-size policy is used to select sets of interfaces depending on the size, hence the name, of the data, e.g., HTTP traffic, to be transferred. Modern websites consist of many assets of vastly different sizes, all transferred individually. Depending on the size of the asset and the latency of the interfaces it can be beneficial to either select only one low-latency interface or combine the throughput of multiple interfaces.

3.3 Data Transfer Simulator

To evaluate the benefits of seamlessly using multiple interfaces and scheduling requests according to our policies at scale and across a wider range of network properties and Web pages, we build an event-based data transfer simulator. As evaluation metric we use page load time[2], which has a high influence on the end-user QoE [52]. Additional metrics, e.g., from Kelton et. al. [53], can easily be implemented in the simulator.

3.3.1 Design

The basic operation principle of our simulator is to take a Web page, a Socket Intents Policy and network interfaces as input, and to calculate a page load time. The Web page includes all Web objects and their dependencies (represented as a HAR files — see Section 3.4.3), the list of network interfaces include their path characteristics. The simulator takes the input and replays the Web page download by transferring all Web page objects while respecting their inter-dependencies. It uses the given policy to distribute the object transfers across the given interfaces and calculates the total page load time.

Since the simulator has global knowledge, it knows all object inter-dependencies a priori. It can, thus, decide when a transfer can be scheduled, i.e, whether all objects that it depends upon have already been loaded. To schedule a transfer, we assign it to a "connection". The scheduling is done by the policy module, which returns either an existing TCP or MPTCP connection, an interface, or a list of interfaces to use for opening a new connection, or postpones the transfer if the limit of parallel connections has been reached. A connection is reused if the host name matches and it is either idle or it is expected to become idle before a new connection can be established.

[2]Here we focus on network time, i.e., the total time to download the objects of the Web page. The complete time to display a Web page also includes times for DNS resolution, page rendering and possibly client-side JavaScript computation.

The simulator then determines the next event for this connection, such as the completion of a transfer or a TCP event. TCP events are triggered by connection handling, TLS handshake, changed available bandwidth share, and once per RTT during slow-start. When a transfer completes, the simulator records the time, marks all transfers that depend on it as enabled, and schedules them. After the last transfer finishes, the total page load time is reported.

The simulator supports persistent connections with and without pipelining for TCP as well as MPTCP connections across multiple interfaces. It uses a default connection time-out of 30 seconds and limits[3] the number of parallel connections per server to 6 and the overall number of connections to 17. We simulate TCP slow-start using a configurable initial congestion window size with a default value of 10 segments [54]. Our motivation for simulating slow-start is to generate more realistic load times especially for small objects, which are common on Web pages, when they are downloaded on high latency links, which are common in access networks. To simulate slow-start and fair bandwidth sharing, we keep track of the current throughput for each connection. The throughput is updated according to the TCP slow-start specification and is capped by the congestion window or the available bandwidth share of that interface to assure TCP fairness[4]. Our underlying assumption is that TCP tries to fairly share the available bandwidth between all parallel connections [55]. Rather than fully simulating the congestion avoidance of TCP we assume instantaneous convergence to the appropriate bandwidth share. The available bandwidth share of each interface is adjusted by each connection event for that interface if needed. The time of the next event is then adjusted accordingly. For MPTCP, our simulator aggregates the bandwidth of the sub-flows by simulating them as separate TCP flows. We do not implement a coupled congestion control, as it is not required, as the network scenario we use in our evaluation, see §3.4.1, does not contain a shared bottleneck.

3.3.2 Implementation

We implemented our data transfer simulator as a heap-based discrete event simulator. It consists of approximately 3000 lines of Python code and is available under a relaxed CRAPL license[5]. It models the process of loading a Web page by keeping track of the status of the transfers, connections and interfaces.

[3]These values correspond to the defaults of the browser we use to retrieve our workload.

[4]In our simulator, a connection leaves slow-start once it reaches the available bandwidth share and never returns to slow-start.

[5]https://github.com/fg-inet/dtsimulator

Each *transfer* corresponds to a Web object. A *Web object* has a size, a relationship to other transfers, the hostname of server where the object is located and an indicator whether the object was transferred via HTTPS.

A *connection* contains transfers and estimates and updating their completion times. Connections also simulate (MP)TCP. In case of MPTCP, we maintain a master connection and per-interface sub-flows.

The *interfaces* bundle connections and calculate the available bandwidth shares.

The *transfer-manager* keeps track of all transfers and informs a policy if a transfer can be scheduled.

The *policy* is the main decision-making entity of the simulation. A policy selects the most appropriate set of interfaces to use or which connection to re-use for each transfer. Once a connection or interface set is chosen, the policy notifies the transfer-manager to schedule the transfer.

3.3.3 Web Object Dependencies

Web objects can not be schedules arbitrary, due to their interdependence. We, thus, derive the order of the objects from the interdependence gathered from HAR files, see Section 3.4.3. While identifying all objects of a Web page from the HAR files is trivial this does not apply to all object dependencies. It is straightforward to extract some object dependencies from the base page, the HTML document, and the client-side Document Object Model (DOM). JavaScript or other Web objects, however, can modify the DOM, by adding or removing objects, at any point during the page load. For example, when using JavaScript to dynamically load images, the simulator may not start downloading the images before the JavaScript object itself has been retrieved and, more importantly, been executed.

We decided against using sophisticated systems to derive object interdependence, e.g., [56], since their focus is on finding the true dependency tree to speed up future downloads. Using these dependencies, thus, often leads to much more optimistic results compared to the capabilities of current browsers. Ensuring compatibility, we, instead, use a more conservative heuristic. We identify the dependencies from the download times contained in the HAR files. This method is feasible, since we use a non-bandwidth limited client to gather the HAR files. Given that a web browser cannot start a new transfer before finishing the download of an object that the new transfer depends upon, we assume that Web objects that are downloaded in parallel do not depend on each other. The same holds true for an object download that has started before a previous object download has finished. In other words, an object can not depend on a partial, or not yet fully downloaded, object.

Since the simulator tries to provide an upper bound of the benefits it relies on its global knowledge about all currently active transfers. The latency and maximum interface bandwidth, as well as the size of the objects for the *Size to be Received Intent*, are known a priori. Within the simulation, we add one RTT if a connection can be reused, two RTTs if a new connection has to be established, and two additional RTTs for each TLS handshake.

Socket Intents Policies can use transfer predictions and policies can, thus, reuse the simulator logic to obtain an estimate of the completion time given the current state and an interface/connection option. This reuse is realized by partially cloning the simulator's state, including all currently active transfers, and simulating the completion time for that transfer.

3.3.4 Policy Realization

In the following we give more detail of how our policies are actually implemented in the simulator.

Single Interface. The policy always chooses a particular, statically configured interface.

Round Robin. The policy uses multiple interfaces in a round robin fashion.

Earliest Arrival First (EAF). The policy (see § 3.2) uses the *Size to be Received Intent* to predict the completion time for each available interface. The prediction is based on estimating the available bandwidth as the difference between the current and the maximum observed throughput and the interface latency. If additional downloads have been scheduled on an interface, since the most recent measurement, we reduce the available bandwidth in our calculation accordingly. We divide the file size by the estimated available bandwidth to approximate the download duration. In other words, the policy predicts the arrival time of each transfer by "scheduling" it on each candidate interface and computing the arrival time. Finally, the interface with the shortest predicted arrival time is chosen.

MPTCP. The policy uses MPTCP with the vanilla full-mesh path manager across all interfaces. It presumes that MPTCP sub-flows can be opened on all local and remote interfaces. With two network interfaces at the client and one interface at the server, the policy establishes two sub-flows. The interface for the initial sub-flow is configurable.

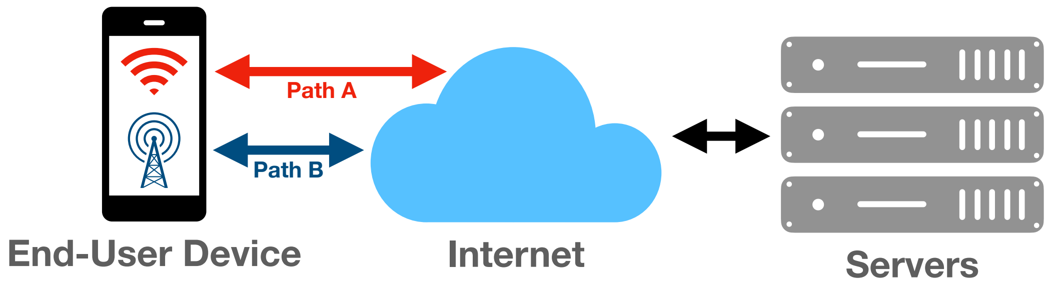


Figure 3.1: *Simplified network scenario with an end-user device equipped with two network interfaces, one resembling WiFi and one a cellular connection. Both interfaces are connected to the internet. The depicted servers resemble content server connected to the internet with high bandwidth, low latency links.*

We considered two variants: starting the initial MPTCP sub-flow on the same statically chosen interface (*MPTCP if1*) or always on a different, randomly chosen interface (*MPTCP rnd*). This policy considers neither the Socket Intents nor the current network performance.

Earliest Arrival First with MPTCP (EAF-MPTCP). The policy combines the *EAF* Policy with MPTCP. In addition to predicting the arrival time for each interface, it also considers MPTCP for all possible interface combinations by establishing sub-flows accordingly. The intuition is that MPTCP is beneficial for some cases but not all cases. This policy can, for example, avoid scheduling small communication units on a high latency interface. We expect that this policy to give the best results.

We test the basic functionality of the various simulator policies using various traffic patterns that can take advantage of *MPTCP*, *EAF*, and *EAF-MPTCP*, with and without connection reuse. For these cases we manually calculate the expected page load times and check the simulator results against them. In addition, we use extensive assertions and cross checks within the simulator to ensure the consistency of the results.

3.4 Evaluation Scenario

To evaluate Socket Intents we assume the following network scenario, which serves as the basis for our simulator.

3.4.1 Network Scenario

Our motivation for the network scenario is that, for mobile devices, access networks almost always are the performance limiting bandwidth bottleneck introducing major delays. Internet backbone delays are in the order of a few milliseconds, while access delays are typically significantly larger. With increasing access network capacities, the bottleneck might be in the core in some cases. Capacities are, however, not increasing in all regions of the world. Our network scenario, see Fig. 3.1, thus, consists of a client, Web servers, and the paths between them. We presume that all Web servers are reachable via both network paths. Moreover, we choose to neglect the RTT variability introduced by the Internet, since queuing delays on Internet core links ($\geq$10 Gbit/s bandwidth) are negligible [57]. Therefore, we capture the path characteristics as "interface" RTT and bandwidth. To model connection reuse, we assume a separate server per hostname.

3.4.2 Experimental Design for Simulator Evaluation

To evaluate the potential benefits of using Socket Intents across a wide range of parameters, i.e., with different policies under different network scenarios and for different Web pages, we use a full factorial experimental design. Each factor can, in principle, influence the page load time. For each factor, we consider multiple values that cover the possible value ranges. By simulating all combinations, see Table 3.1, we run 9M simulations.

In our experimental design, the primary factor is the **Policy** used with all of our Socket Intents Policies, see Section 3.3.2, as levels. The **Web pages** of our workloads, see Section 3.4.3, are the second factor: Here, the levels are the different Web pages (with their 26 repeated crawls for Alexa Top 100 and one crawl for the Alexa Top 1000).

The remaining four factors describe the scenario: Since our simplified network scenario as illustrated in Fig. 3.1 consists of one client using two access networks and various Web servers which are reachable via both interfaces, these factors are: **Interface 1**

Table 3.1: *Levels of the Factorial Experimental Design.*

Factor	**Levels**
Policy:	*Interface 1*, *Interface 2* *Round Robin* (starting on if 1), MPTCP starting on Interface 1 (*MPTCP if1*) or on a random interface (*MPTCP rnd*), Earliest Arrival First (*EAF*), or EAF with MPTCP (*EAF-MPTCP*).
Web page:	Alexa Top 100 and Top 1000.
Interface 1 RTT:	10, 20, 30, or 50 ms.
Interface 1 Bandwidth:	0.5, 2, 6, 12, 20, 50 Mbit/s.
Interface 2 RTT:	20, 50, 100, or 200 ms.
Interface 2 Bandwidth:	0.5, 5, 20, or 50 Mbit/s.

RTT and **Bandwidth** as well as **Interface 2 RTT** and **Bandwidth**. The levels for these were chosen to reflect typical interface characteristics: We consider mobile devices that have WiFi as well as cellular connectivity. Interface 1 should resemble the possible characteristics of fixed broadband connectivity (e.g., DSL or cable) and Interface 2 should resemble the range of possible 3G/LTE coverage[6]. This results in the levels shown in Table 3.1.

3.4.3 Web Workload

To include a wide range of Web pages, we crawl the landing pages of the Alexa Top 100 Web sites on 26 consecutive days starting on December 07 2015 and the Alexa Top 1000 Web sites on October 10 2016[7].

As browser we use Firefox version 38.4.0 automated with Selenium and the Firebug 2.0.13 and NetExport 0.9b7 plugins to retrieve the objects and to record the crawled Web pages in the HTTP Archive (HAR) format. Each HAR file contains a summary of all objects of the page as well as their sizes, types, origins (remote sites), and timings. We focus on the mobile version of the pages by overriding the user-agent of our Firefox browser to impersonate a generic Android mobile device.

We use a single vantage point with a high available network bandwidth, i.e., a virtual machine within a university network.

While most of the pages comprise between 1 and 50 objects there are some with more than 100 objects or even up to 260 objects. Many Web pages also have a low median object size. Moreover, the number of hosts that have to be contacted ranges from a single one to more than 20 with a median of 7. The total size of the Web pages is between 23.1 KB (5th quantile) and 1.8MB (95th quantile) with a large fraction of pages below 300 KB. These results are in line with previous work [19, 20].

For comparing with a less complex Web workload, we add handcrafted Web pages to our workload. These pages consist of a different number of objects (ranging from 2 to 64) of various sizes (1 KB to 1 MB), and a mix of these objects.

[6]Costs or restrictions of the data plan are beyond the scope of this paper, but could easily be taken into account by an elaborate policy.

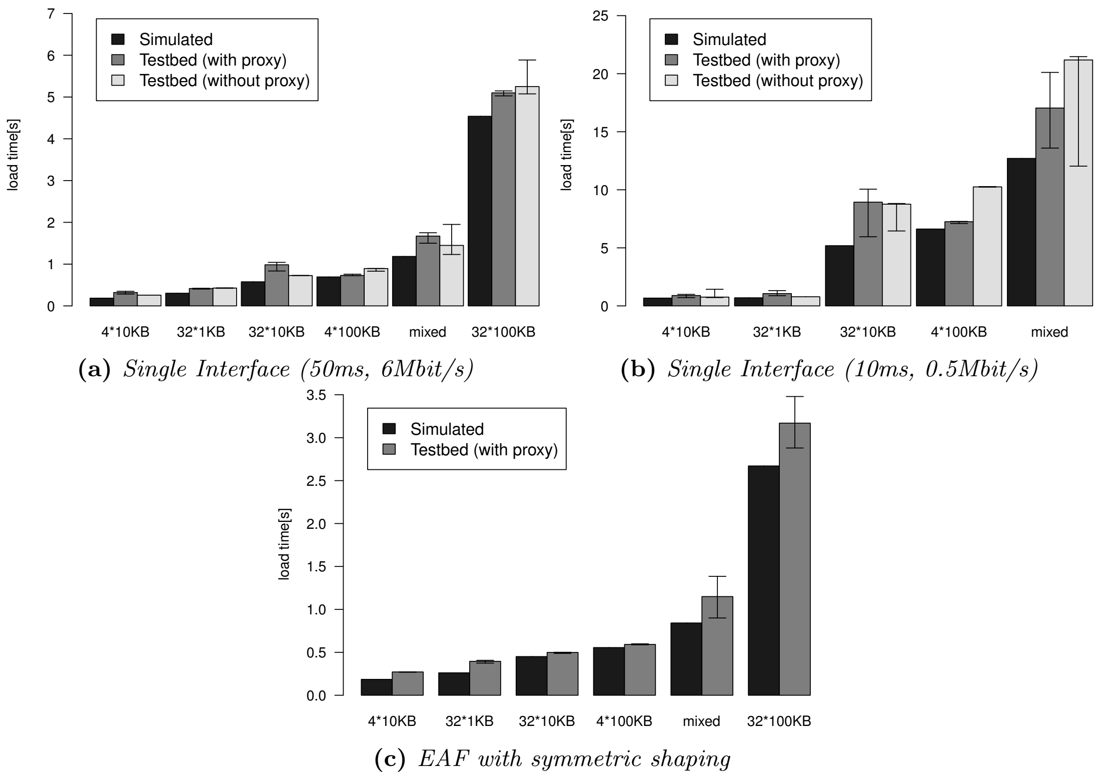
[7]http://www.alexa.com/topsites

3.5 Evaluation

To explore the benefits of seamlessly combining multiple access networks for speeding up Web page load time, we evaluate Socket Intents.

We use our simulator to explore speedups across a wide range of web pages and network characteristics. Moreover, we validate our simulator against the existing Socket Intents proxy in the existing testbed to show that the speedups are comparable. We show the possible performance benefits of Socket Intents and highlight in which cases they are most prominent.

3.5.1 Validation of our Simulator with the Proxy



(a) *Single Interface (50ms, 6Mbit/s)* **(b)** *Single Interface (10ms, 0.5Mbit/s)*

(c) *EAF with symmetric shaping*

Figure 3.2: *Comparison of simulated load time and actual load time in the testbed with different synthetic workloads. There are differences in the y axes.*

We validate our simulation results against the existing Socket Intents proxy by measuring the Web page load time of our workload in the existing testbed with similar traffic shaper settings as the interface parameters we use in the simulator. In Figure 3.2 we compare the simulated and the actual load times for the handcrafted workloads of different sizes, showing the median load time and the 95% confidence intervals. The mixed workload consists of 32 objects of 1KB, 16 objects of 10 KB, 2 objects of 100 KB and 2 objects of 200 KB. Using a single interface with an RTT of 50 ms and a bandwidth of 6 Mbit/s,

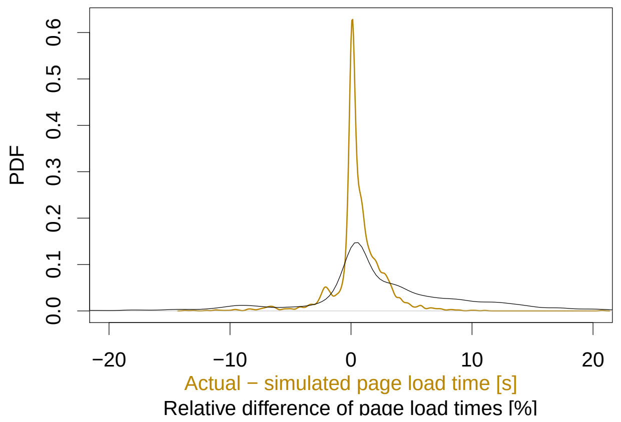


Figure 3.3: *Simulator validation: Relative and absolute difference of simulated vs. actual page load time.*

see Figure 3.2a, we see slightly higher load times on the testbed both with and without the proxy, especially for large workloads. Using a single interface with only 0.5 Mbit/s, see Figure 3.2b, we do not get a page load time for the workload with 32 objects of 100 KB because the browser times out after 10-20 seconds, so we do not show it in this plot. Using our *EAF* policy with symmetric shaping (50 ms and 6 Mbit/s on one interface, 50 ms and 5 Mbit/s on the other), we cannot test the case without proxy, as we cannot use *EAF* without the proxy. Both our simulator and the proxy in the testbed show speedups, see Figure 3.2c. We get similar results for RTTs up to 200 ms and bandwidths up to 50 Mbit/s and similar load times with and without the proxy.

The speedups show that the two step download in the proxy does not have a major influence on the load time. The simulator is, overall, more optimistic than the testbed, however, the differences are quite small. These small differences can be explained by the following observations: First, the gzip transfer encoding conflicts with range requests: Sometimes the server sends the whole object even though only the initial part is requested. First, disabling compression for the initial request is not feasible as it eliminates compression also for the second request since the content-range refers to the range after compression. Second, the simulator presumes that all independent transfers start immediately, which is not always the case in practice. This can skew timings, in particular for small workloads. These effects are independent from the use of the Socket Intents Prototype. Accordingly, we can use the simulator to conduct a realistic comparison between scenarios with and without Socket Intents.

3.5.2 Simulator vs. Actual Page Load Time

We compare the actual page load time to the simulated one for all Web pages of our workload. Given that our crawl uses a machine with a single interface we use the single interface with our policy "Single Interface" accordingly. To determine the interface parameters, we estimate the available bandwidth as well as the RTT to the servers of the actual download. To estimate the available bandwidth, we use all objects larger than a minimum size of 50 KB. We take into account that several of these transfers can occur in parallel. Using the median of the estimated bandwidth results in a typically used bandwidth of 67.13 Mbit/s – this suggests that none of the transfers were actually bandwidth bound. To estimate the RTT, the simulator issues a series of pings for each Web page. The median RTT of all servers of that Web page is then used as an estimator for the interface for the validation run for that Web page.

The simulator, as well as the validation, uses several simplifications. First, the simulator assumes that all Web objects share a single network bottleneck and that the RTT is the same for all servers. In reality, some embedded objects of Web pages are fetched from hosts with different network bottlenecks and RTTs. We use ICMP ping rather than TCP ping and the pings are not executed while the HAR files are gathered.

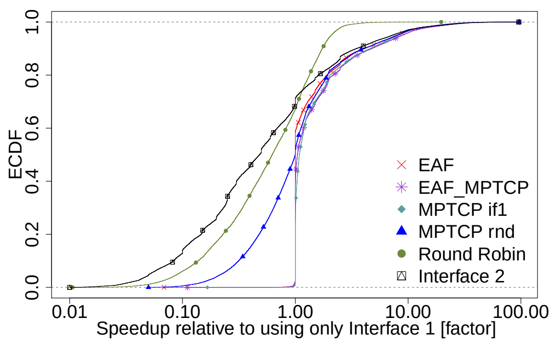
Figure 3.3 shows the absolute as well as the relative differences of the simulated vs. the actual page load times for all Alexa Top 100 Web pages from Section 3.4.3. The main mass of both distributions is around zero, indicating that the simulated page load times are very close to the actual load times. This closeness is confirmed by the median value which is 0.3548/1.5% for the absolute/relative differences. The small difference highlights that the simplifying assumptions of the simulator still enable us to approximate the actual page load times and that we capture most of the intra Web page dependencies.

There are some differences for some Web pages. We manually checked them and found a majority is caused by differences in the estimated bandwidth, server delays, and name resolution overhead. These differences are, e.g., related to Web back-office interactions [58]. Overall, the results are rather close and show that our simulations result in reasonable approximations of the actual Web page load time.

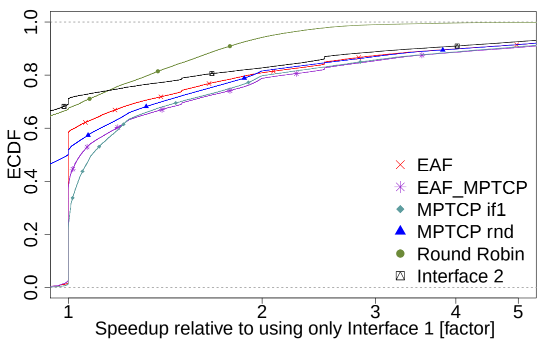
3.5.3 Benefits of Combining Multiple Access Networks

To explore the benefits of combining multiple access networks by using Socket Intents, we compare the speedups of the page load times against the baseline policy *Interface 1*. The baseline policy *Interface 1* resembles what most current mobile OSes do: If WiFi is available, use it, and therefore, the fixed broadband exclusively.

Fig. 3.4a shows the empirical cumulative distribution functions (ECDF) of the speedups achieved using a simulated Socket Intents Policy relative to only using *Interface 1*, all



(a) *Full Data Range*



(b) *Speedups between 1 and 5.*

Figure 3.4: *CDF of Speedups vs. Interface 1 for the Alexa Top 100 workload.*

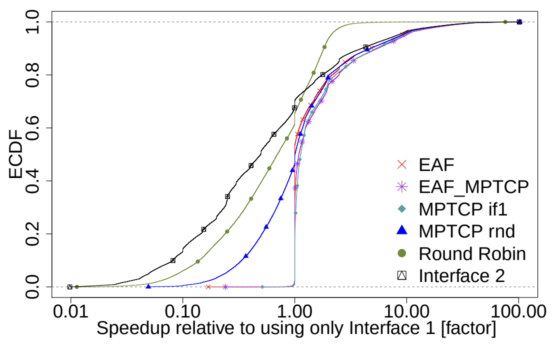


Figure 3.5: *CDF of Speedups vs. Interface 1 for the Alexa Top 1000 workload.*

other parameters being equal. The ECDF shows speedups across all network scenarios outlined in Table 3.1 based on the Alexa Top 100 Web pages and categorized by the Socket Intents Policy used. We see that in more than 42% of the cases for *EAF* and 63% of the cases for *EAF-MPTCP* these policies provide a speedup of more than 1, which means that loading a Web page using these policies is faster than using *Interface 1* in the same scenario. In the remaining cases, the policies almost always provide a speedup of 1, which means that they neither gain nor lose from using multiple interfaces. In the speedup of 1 cases, the page load was not bandwidth limited and simply loading the page over Interface 1 was the fastest option. Using the other interface in addition, thus, did not provide any speedup as the *EAF* and *EAF-MPTCP* simply choose to use Interface 1. We also see that in about 1.5% of cases *EAF* and *EAF-MPTCP* is slower than *Interface 1* which turned out to be a limitation of the simulator. In these slower cases, the simulator fetches a single huge object via the less suitable interface while the connection limit prevents starting a new connection on a more suitable one. Overall, these results show that using *EAF* and *EAF-MPTCP* is a good choice in any case.

The speedups of both MPTCP policies are very dissimilar: When establishing the first sub-flow over Interface 1 (*MPTCP if1*), it shows a speedup greater than 1 in 78% of the cases and neither improvement nor penalty in the other cases. In contrast, if starting the first sub-flow for MPTCP over a randomly chosen interface (*MPTCP rnd*), MPTCP

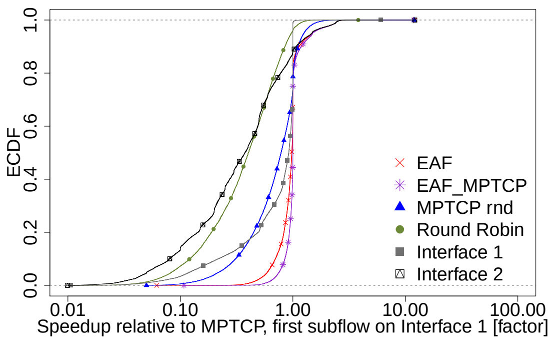


Figure 3.6: *CDF of Speedups vs. MPTCP if1 for the Alexa Top 100 workload.*

performs worse than *Interface 1* in 48% of the cases and can be up to 10x slower. We take a closer look at these effects in Section 3.5.4.

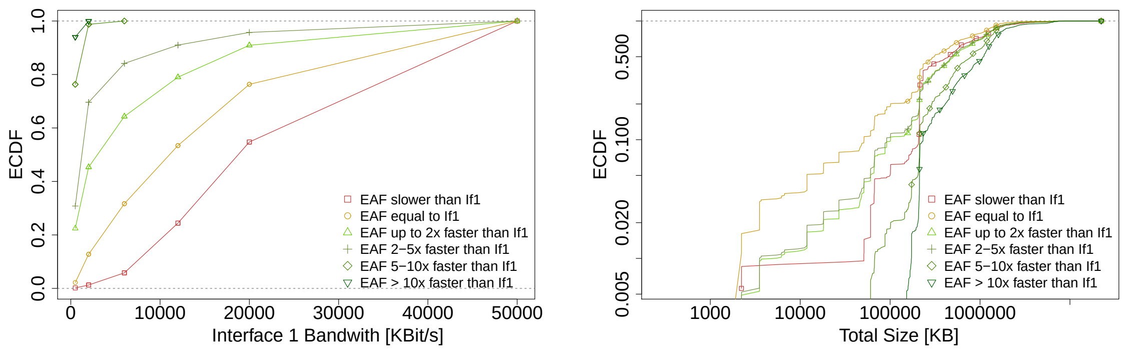
The other baseline policies, *Interface 2* and *Round Robin*, show a penalty in about 70% of cases as in most network scenarios Interface 2 has a much higher RTT than Interface 1.

Figure 3.4b shows the speedups between 1 and 5 from Figure 3.4a in more detail. From our data, we find that *EAF* was up to 2x faster than *Interface 1* in about 23% of the cases and from 2 to 5x faster in about 11% of the cases. We even see speedups of more than 5x in 8.5% of the cases. *EAF-MPTCP* and *MPTCP if1* shows negligibly higher speedups than *EAF*. All three policies, overall, perform similarly and can have significant advantage over not combining multiple access networks. Finally, Fig. 3.5 shows the ECDF of the speedups against *Interface 1* for the Alexa 1000. These look similar to the ones for Alexa 100 in 3.4a. This gives us confidence that our benefits are stable for a wide variety of different Web pages.

3.5.4 Benefits of Using MPTCP

As described in Section 3.5.3, for our dataset *MPTCP if1* and *MPTCP rnd* behave very differently. While both show speedups in almost all cases, *MPTCP rnd* is at a disadvantage in 48% of the cases while *MPTCP if1* almost never encounters a penalty.

In Fig. 3.6, we compare speedups of our policies for all scenarios and Web pages against *MPTCP if1*. The curves for *EAF* and *EAF-MPTCP* are close to 1, which means that the page load times are similar to MPTCP in most cases and never considerably worse. In contrast, if establishing the first sub-flow for MPTCP over a randomly chosen interface (*MPTCP rnd*), MPTCP performs worse and can be up to 10x slower than using *Interface 1* and about 30x slower than *MPTCP if1*. This worse performances stems from the fact that Interface 1 has a shorter RTT in most network scenarios. As many Web page downloads, in our workloads, were short and not bandwidth bound,



(a) *Interface 1 bandwidth* **(b)** *Sizes of Web pages*

Figure 3.7: *Levels of factors for which we see a certain level of speedup for Alexa Top 100.*

MPTCP will often perform most of the download over the initial sub-flow. Thus, not picking the most suitable interface in 50% of the cases bears a considerable performance penalty. *EAF-MPTCP* can always choose the most suitable interface for the first sub-flow and, therefore, can improve over *MPTCP if1* in cases where Interface 1 is not the most suitable first-sub-flow interface.

The *EAF* shows a similar performance as *MPTCP if1*. The cases where *EAF* and *EAF-MPTCP* perform slightly worse than *MPTCP if1* seem negligible to us given the benefits. These cases occur because *EAF* and *EAF-MPTCP* do not take future transfers into account. They cannot change their decision whether to use MPTCP, while always using MPTCP allows a later re-balance of traffic between sub-flows.

3.5.5 Explaining Page Load Time Speedups

To understand how the factors of the scenario and Web page affect the speedups of our policies, we take a closer look at the cases when *EAF* is slower, similar to, or faster than *Interface 1*.

In Fig. 3.7, we bin the simulation results of *EAF* into six categories of benefits and show how these distribute among the total Web page sizes and Interface 1 bandwidths. These categories contain different numbers of observations, i.e., *EAF is slower* for just 1.5% of all cases while *EAF is equal to Interface 1* for 56.6% of all cases.

The CDF, in Figure 3.7a, shows the frequency of the speedup categories over the different levels of Interface 1 bandwidths from Table 3.1. In cases were *EAF* was slower or equal to *Interface 1*, higher values for the Interface 1 bandwidth are more prevalent, while high speedups mostly occur when the Interface 1 bandwidth is low. Similarly, we tend to see high speedups for higher levels of Interface 2 bandwidth and for lower levels of Interface 2 RTT (plots omitted).

To explore what kind of Web pages can benefit from our *EAF* policy, we plot the CDF of the speedup categories over the total Web page size in Fig. 3.7b. As high speedups occur much more frequently for large Web pages, we conclude that these take most advantage of using multiple access networks. For the median object size and the number of objects in a Web page we see similar results, with high speedups occurring more frequently in cases with high median object sizes.

Both analyses show that our multi-access policies are most useful when Web page download is bandwidth limited.

3.6 Related Work

We, first, review related work regarding multipath support in general and on the end-host's acOS in particular, and then focus on how application needs are taken into account. Finally, we discuss the benefits of using multiple access networks in the context of WiFi offloading and MPTCP.

Multipath

For a comprehensive survey of network layer multipath solutions see Qadir et al. [59]. They present a detailed analysis of the design choices of how to compute and select routes as well as how to split the flow across the chosen paths.

A survey of multipath approaches in some current OSes [60] points out several problems that we also discuss in Section 2.3. Many OSes support mechanisms for source and destination address selection for IPv6 multihoming [61, 62] and there are proposed Socket Application Programming Interface (API) extensions that enable applications to set preferences [61]. These address selection algorithms, however, focus on reachability, while we consider bandwidth aggregation and performance improvement. Some OSes implement a central connection manager to choose the appropriate access network, as is also proposed in current research such as by Kiefer et al. [63]. The latter uses policies controlled by the application and the user and relies on observations of the current network performance, which is similar to our Multi Access Manager. However, it only works on a per-flow basis and not per communication unit. Also, their application policies specify flow prioritization and constraints, but not different characteristics of the traffic.

Application Needs. Previous work where an application can specify its requirements and needs often focuses on QoS, e.g., QSockets [64]. We use the best-effort approach of Socket Intents. The term *Intents* has its origin in Intentional Networking [65], an attempt to explore mobile network diversity by letting applications specify traffic characteristics via an extended Socket API. However, they use a per-packet approach, which introduces complications and overhead, while we use a per-socket/per-flow or per-request approach. Moreover, they imply guarantees while we suggest best-effort. Other approaches include ideas from machine learning to guide application choices, e.g., Deng et al [66].

Socket APIs

Alternative socket APIs move parts of the application logic to the socket API, e.g., by requesting a service rather than a protocol, port, and address in the protocol-independent transport API [67] or by exposing all protocols and auxiliary information of the application in a tree-like structure [68]. In contrast, the Socket Intents Prototype takes transport protocols as given. Thus, the idea of Socket Intents is complementary to the before-mentioned work.

Offloading. Multi-access connectivity enables one to balance traffic, e.g., from the mobile network to the WiFi—offloading—or from WiFi to the mobile network—onloading. Recently, both variants have gotten a lot of attention in the research community, e.g., [69–72], as well as in industry, e.g., [73, 74]. For a survey on offloading, we refer to, e.g., Aijaz et al. [69]. For a summary of multi-access connectivity, we refer to, e.g., Schmidt et al. [75]. Examples of recent work on offloading include the work by Lee et al. [70], who demonstrate via a quantitative study the performance benefit of offloading 3G mobile data to WiFi networks, and Balasubramanian et al. [71], who propose Wiffler to augment mobile 3G capacity with WiFi. For onloading, we, e.g., point to Vallina et al. [72]. For an analysis of the economics of offloading see Lee et al. [73]. Offloading typically implements support for using multiple access networks within the network or on the application layer, while we provide support for it within the end-host OS.

MPTCP

There have been many studies exploring how an end host can benefit from multiple paths using MPTCP. Chen et al. [76] evaluate MPTCP performance in the wild by comparing its use over a home WiFi network and several different cellular providers to the use of a single path. They find that for small files, using a single path over WiFi is best, while larger files benefit from MPTCP's aggregated bandwidth. This observation is shared by Raiciu et al. [77] and Deng et al. [78], who emphasize that the choice of the interface to establish the first sub-flow is important, which is in line with our observations.

Han et al. [79] evaluate page load times of HTTP and SPDY over WiFi and LTE using a proxy-based setup. They find that SPDY over MPTCP is always beneficial. This is in contrast to HTTP over MPTCP which in some cases performs even worse than plain TCP. Similarly, Nikravesh et al. [80] observe a performance penalty and energy consumption overhead for apps with small flows in the wild. As a solution, they propose a proxy with persistent connections over multiple paths.

A similar approach for controlling MPTCP (see Section 3.1) has been proposed by Hartung and Milind [81] to use MPTCP for LTE bandwidth management.

Stream Control Transmission Protocol (SCTP)

Dreibholz et al. propose an advanced stream scheduling policy for SCTP[82] and achieve performance benefits in asymmetric path scenarios using a simulation. It would be possible to integrate such an advanced scheduler into our policies in future work.

3.7 Summary

We created an event-based transfer simulator to explore speedups across a wide range of Web pages and network characteristics. Our simulations demonstrate that Socket Intents with the *EAF* policy can improve Web page load time in about 50% of the cases without incurring penalties. Therefore, without kernel modification, we can achieve about the same speedups possible by using MPTCP. In cases where the access networks characteristics are very different, our *EAF-MPTCP* helps MPTCP to pick the right interface for the initial sub-flow and prevents performance penalties that can occur when using the "‘wrong”’ interface.

Socket Intents are a first step for sharing information between applications, the OS, and the network. This idea is not limited to the use case of multiple access networks, but can also be beneficial to automatically choose among transport protocols and can give valuable input to traffic management systems for datacenter networks.

It was also shown that Socket Intents can be beneficial for video streaming traffic [11]. We will, though, move away from Socket Intents, as we gained insights, described in the next chapter, that require a more fine-grained control over the information sharing between the application and the transport.

Chapter 4

Video Transfer Design

Following the cross-layer approach that Socket Intents brought us for web traffic, we will now look at an even bigger player in today's content landscape: video. Although it was shown by Enghardt *et al.* [11] that an informed network selection does improve video streaming performance, we want to take this approach even further and couple the transport and application even tighter.

The focus of our efforts lie specifically on a key determinant of end-user QoEs: uninterrupted playback. Supporting uninterrupted playback will likely have a far reaching impact, especially with live streaming and broadcasts over the Internet becoming commonplace [83, 84]. Streaming video over the Internet without interrupting playback at the client (i.e., video player) is, though, a notoriously hard problem. Such playback interruptions have a discernible impact on end users and degrade their video-watching experiences, as measured via QoE metrics [85]. It is exacerbated by the fact that each video frame has an implicit *deadline* within which it must be delivered to the client; the client will, otherwise, *stall* playback to wait for the frame to arrive.

The frames are, in today's HTTP streams, grouped into *segments*—sequences of bytes spanning a predetermined (typically, 2-10 s) time duration [86]. There is a rich literature on video streaming aiming to reduce playback interruptions, but virtually all of them focus on either "fine-tuning" TCP or making clients "smarter" by using ABR algorithms.

Prior efforts on fine-tuning TCP for video streaming include side-stepping packet losses [87], supporting real-time delivery [87–90], adding deadline-awareness [91, 92], using retransmissions for sending new data [93], and using coding techniques for loss recovery [94, 95]. They all use TCP, a reliable transport, although video streaming can tolerate *some* packet loss [96]. Upgrading or improving TCP is also hard. While the QUIC protocol makes it easy to extend the transport [97], it offers only reliable streams for communication,[1] and, thus, inherits most of TCP's problems. Using a reliable transport for video streaming has remained, thus far, the status quo, and we question this choice.

To make clients smarter, prior work also proposed numerous ways of improving ABR algorithms [41, 42, 101–104], which are client-side mechanisms that dynamically choose a bitrate for each video segment based on several factors, e.g., available bandwidth estimate and playback buffer occupancy. They directly or indirectly optimize one or more objective QoE metrics, e.g., rebuffering (or stall) duration, average bitrate, and bitrate switches. Notwithstanding the ABR-algorithm improvements, they simply assume that video segments must be downloaded in their entirety—thus, inherit the problems of reliable delivery for video streaming.

Consequently, we present *VOXEL*, our two-pronged approach that combines a partially reliable transport with application-provided "insights" for optimizing video streaming. More specifically, *VOXEL* combines our partially reliable implementation of Google QUIC (henceforth, referred to as QUIC*) with the insight that not all frames of a video require reliable delivery—frame-drops do not necessarily compromise end-user QoE [96, 105].

⋆ *First, in Chapter 5, we present a new server-side algorithm that performs a one-time analysis of the impact of varying amounts of frame-drops on the end-user QoE for a given video.*

One recent work, the bandwidth-efficient temporal adaptation (BETA) [105], pursues a similar approach, though we show that BETA (a) implements only a subset of features of *VOXEL*, (b) is not as deployment friendly as *VOXEL*, and (c) performs poorer than *VOXEL* in most of our tests.

⋆ *Second, in Chapter 6, we present a novel ABR algorithm that* explicitly *trades off frame losses for optimizing end-user QoE.*

While we leverage some ideas from prior work, *VOXEL* is the first end-to-end system that introduces a QoE-metric-based frame-importance measure and ranking—a new capability not found in any prior work. This capability enables *VOXEL* to control video quality in a fine-grained manner by dropping different types of frames (including referenced frames) to combat challenging network conditions, with minimal QoE impairment. Thus, *VOXEL* outperforms state-of-art solutions (e.g., MPC [41] and BOLA [42]) as well as recent work, e.g., BETA (see §6.4).

[1]There have been a few proposals to support unreliable delivery in QUIC (e.g., [98–100], but none have been *standardized* or implemented in Google QUIC, as of this writing.

⋆ *We want to emphasize that VOXEL is more than a simple research prototype.*

VOXEL works seamlessly with Dynamic Adaptive Streaming over HTTP (DASH) and is backward-compatible with existing *VOXEL*-unaware clients. *VOXEL* is also easily deployable given QUIC's widespread adoption both on the server side (e.g., in CDNs [106, 107]) and client side (e.g., in major web browsers [108, 109]). In our evaluations, *VOXEL* streams videos in the 90^{th}-percentile with up to 97% less rebuffering than the state-of-the-art without sacrificing end-users' QoEs.

⋆ *Lastly, VOXEL is QoE-metric-agnostic.*

We use the all-component SSIM [44] of `FFmpeg`'s `ssim` filter for estimating QoE in these evaluations, but also show that *VOXEL is QoE-metric-agnostic*: *VOXEL* outperforms the state-of-the-art even with respect to other widely used metrics, e.g., VMAF [46] and PSNR.

Following our insights, we will, in the next section, discuss work related to our video transfer design.

4.1 Related Work

A rich body of work about improving video streaming exists, though, virtually all prior work follow a piecemeal approach—either "tweaking" the fully reliable transport layer or making the client "smarter." Confining the parameter space even further, besides early academic work on video streaming that used UDP, practically all recent work and all HTTP streaming use TCP as the reliable transport. We are, in the following, highlighting important related work, and compare similar, yet limited approaches from recent work in Tab. 4.1.

Feamster et al. [96] and, more recently, we [110] demonstrated that video streaming will benefit from a partially reliable transport. The idea that these prior work build on—some frame losses do not substantially impair the visual quality of the video stream—was also recently explored in BETA [105]. Unlike *VOXEL*, BETA considers loss of only unreferenced B-Frames, uses TCP, and implements only a subset of features of *VOXEL*.

Almost all prior work on ABR algorithms (e.g., [38, 40, 41, 102–104]) leave the underlying transport—typically, TCP—intact. Bhat et al., show that porting such ABR algorithms to QUIC does not suffice [111].

Prior work also looked at "tweaking" or "tuning" TCP for video streaming. TCP variants such as TCP-RTM [87] and TL-TCP [91] either ignore retransmissions or avoid

Feature / **System**	**VOXEL**	**BETA**	**PR-SCTP**	**BOLA**
QoE metric based optimization	X			(3)
Frame importance ranked by VQ	X			
Allows tail drop of less important frames	X	X		
Frame-level quality selection	X			
dynamic quality switching [2]	X	(1)		
Selective retransmissions	X		(4)	
Partial segment request	X	(2)		
Any frame type can be dropped	X		X	
Allows minor visual glitches	X		X	
No manifest alteration needed			(non applicable)	X
No video alteration needed	X		X	X
Evaluated with real network traces	X		(non applicable)	X
Open source	X		X	X
Evaluated in user survey	X			

Table 4.1: *List of features of different protocols and systems. Note: (1) BETA only specifies a single visual quality (VQ) threshold per segment, per encoding, stays on that quality level and falls back to the lowest quality if the required frames are not downloaded in time. (2) BETA can stop segment downloads, resulting in a random partial segment. VOXEL, instead, requests a specified partial segment. (3) While not implemented, BOLA lends itself to utilizing a QoE metric as its optimization function - VOXEL makes use of that. (4) PR-SCTP has a timed-reliability, though, requires an application to determine it.*

retransmitting data that have already missed the deadline. Both complicate application design, making deployment impractical, if not impossible. Brosh et al. [89] suggest optimizations to make TCP more friendly for delivering real-time media. In a similar vein, Goel et al. [88] tune TCP's send buffer for mitigating delays. These optimizations will be even more beneficial when applied selectively—to only the portion of data that requires reliability. McQuistin et al. [93] propose a TCP variant that uses retransmissions to deliver new data. This idea alleviates some but not all of the overhead.

Research work on components for dropping frames exists, but seldomly as a full streaming system. Yahia et al. [112] propose video delivery schemes that exploit HTTP/2 mechanisms to drop frames during low-latency live streaming. They, however, use only single-bitrate videos without any ABR algorithm. Stewart et al. [113] present the Partial Reliability Stream Control Transmission Protocol (PR-SCTP), a partial reliability extension to SCTP. The protocol allows disabling retransmissions on a per-message basis. As a transport protocol, it does not have any notion of video frames and, thus, requires an application on top to determine how to drop frames, and when to enable retransmissions.

Feamster et al. [96] explore the effect of selective reliability for streaming video via RTP, necessitating substantial changes to the network stack. *VOXEL*, in contrast, requires

[2]Allows quality switching mid-segment-download based on VQ adaptation function.

minimal changes and can be deployed incrementally. Fouladi et al. [114] designed *Salsify*, a system for video conferencing, that adjusts the encoding quality per video frame. This approach is less suitable for Video on Demand (VoD) as it would require re-encoding the video for each and every view. *VOXEL* introduces a one-time computational overhead, a priori when enriching the manifest, regardless of how many times the video is streamed.

BETA from Cyriac et al. [105] is most relevant to this work. BETA, however, uses TCP and, thus, does not allow for imperfect transmissions. Their optimization, instead, stems primarily from dropping unreferenced B-Frames; the videos in Tab. 6.1 in §6.4.2, for instance, contain more than 30% P-Frames, which constitute at least 56% of video data—much more than B-Frames. In challenging network conditions (e.g., cellular networks), the inability to drop referenced frames (even if they do not introduce perceivable issues) hurts their performance. On average, *VOXEL* dropped frames in 9% of segments. In 85% of those cases it was not sufficient to only drop b-Frames, but 46% of all referenced frames had to be dropped as well. Unlike *VOXEL*, BETA does not analyze the QoE implications of losses of different type and number of frames. Their single virtual quality level does not allow them to adjust the segment quality at any instant of time, when conditions deteriorate. If the throughput does not suffice, they either accept a segment unaware of its visual quality, or, in the worst case, simply discard the data and fetch the same segment at the lowest quality.

4.2 Insights

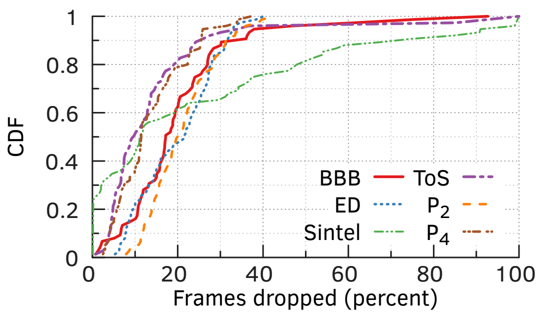
Our goal is to systematically identify which frames can be dropped *a priori*, i.e., in an offline process, while still delivering a high QoE to end users.

Although such a system would benefit from socket intents, their interface selection is too coarse-grained to satisfy the specific needs of video streaming.

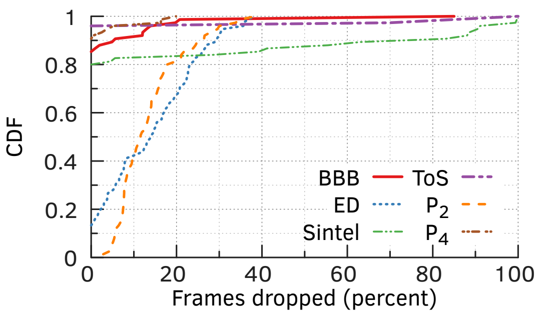
In the following sections we describe the key insights (highlighting similarities and differences with relevant prior work) that underlie our fine grained cross-layer approach for realizing this goal.

We selected 4 widely used videos from prior work, namely *Big Buck Bunny (BBB), Elephants Dream (ED), Sintel, and Tears of Steel (ToS)*, for demonstrating the insights; Tab. 6.1 in §6.4.2 provides a brief characterization of the videos. To test whether they generalize to other videos, we also used 10 public YouTube videos (P_1 - P_{10} in Tab. 6.3 in §6.4.3) that were retrieved following an approach similar to that of Yeo et al. [115].

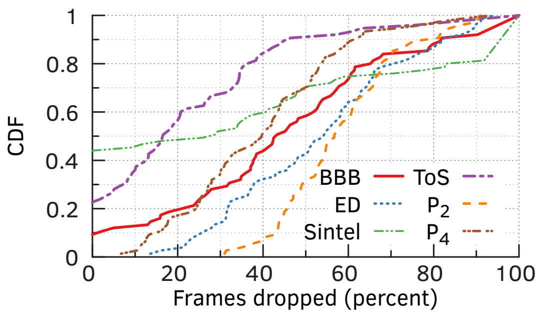
We restrict our focus to only the 4 videos in Tab. 6.1 (in §6.4.2) and 2 randomly selected videos from Tab. 6.3 (in §6.4.3) to simplify the plots and explanations. We include a detailed discussion of the analyses of all the YouTube videos in §6.4.3. For each video, we selected a 5-minute section comprising 75, 4 s long segments. Each video is transcoded at 13 different bitrates (Tab. 6.2 in §6.4.2) ranging from the lowest quality (Q_0) at 0.16 Mbps to the highest quality (Q_{12}) at 10 Mbps. The bitrates in the plots denote the bandwidths required to stream the concerned segments, since we utilized the segment sizes and not the video-wide average bitrate typically used in ABR implementations.



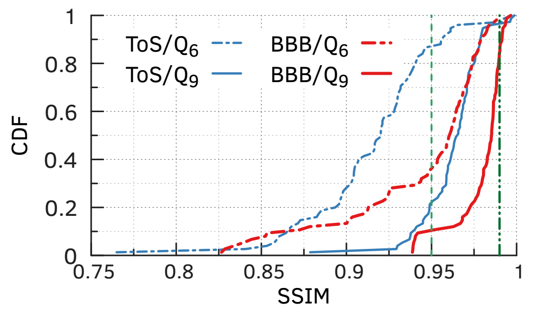
(a) *Q_{12}, SSIM 0.99*



(b) *Q_9, SSIM 0.99*



(c) *Q_9, SSIM 0.95*



(d) *Low quality levels and SSIM*

Figure 4.1: *(a) A significant number of frame-drops (excluding the I-Frame) can be tolerated while still guaranteeing an SSIM score of 0.99; The frame-drop tolerance (b) diminishes when switching down from Q_{12} to Q_9, but (c) improves if we also lower the target SSIM score from 0.99 to 0.95; (d) Low quality (e.g., Q_9 & Q_6) segments typically have SSIM scores less than 0.99.*

4.2.1 Drop Frames while still Delivering a high QoE

Typical video content can tolerate some dropped frames without a significant impact on QoE [96, 105, 112]. Fig. 4.1a shows the percentage of frame-drops that we can tolerate across different segments of video at quality Q_{12}, while still maintaining a *segment average SSIM score* (or *SSIM score*, in short) of 0.99, i.e., excellent quality with imperceptible impairment. In calculating the impact of frame-drops, we assume that the I-Frame of each segment was delivered reliably, i.e., without loss.

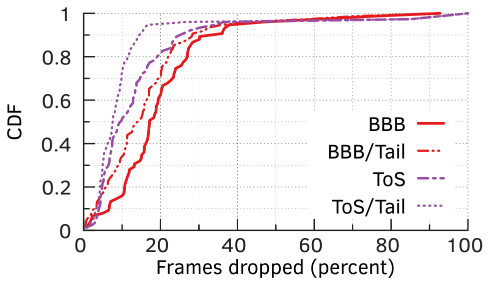
The number of frames that can be dropped depends to a large extent on the content of the video. In a scene with almost no action or a title scene, for instance, it might suffice to drop all but the I-Frame, and the video player could repeatedly show this frame for the entire segment. Brooks et al. [116] substantiate that a lossy image with higher encoding rate has a higher visual quality (here, SSIM) than a lossless image at lower encoding. For each of the six videos, at least half the segments can sustain a 10% to 20% loss in frames while still delivering an SSIM of 0.99. Of these dropped frames an average of 12.6% for ToS, 22.8% for BBB, 27% for ED, and 30% for Sintel are referenced frames. We, hence, drop 6%, 15%, 9.5%, and 24.2% of all referenced frames in ToS, BBB, ED, and Sintel, respectively. This ability to drop referenced frames while maintaining an SSIM of 0.99, thus, allows *VOXEL* to efficiently adapt to challenging network conditions, better than all prior work.

Fig. 4.1b repeats the experiment at the lower quality Q_9 at 4.3 Mbps (refer Tab. 6.2), demonstrating the interaction between encoded bitrate and frame-drop rate. The low bitrate of Q_9 lowers the SSIM even in the absence of any loss: 85% of the BBB segments and 96% of the ToS segments at Q_9 have, per Fig. 4.1d, an SSIM score less than 0.99. Fig. 4.1c shows, however, that to improve the frame-drop tolerance at Q_9, we can target an SSIM score of 0.95, which still offers good quality [117] with perceptible but not detrimental impairment.

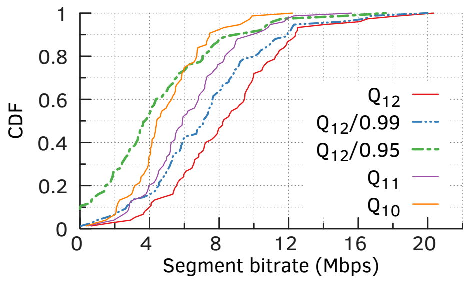
This idea of tolerating frame losses was explored in [96] and more recently in BETA. Our approach of identifying "importance" of frames analyzes the dependencies between frames is more involved (as we later show in §6.1) than relying on the relative positions of frames (as in [96]) or simply tagging only unreferenced B-Frames as "unimportant" [105]. Unlike *VOXEL*, prior work either use a complex solution involving RTP/RTSP [96] or rely on TCP [105].

4.2.2 Reorder "unimportant" Frames to Segment's Tail

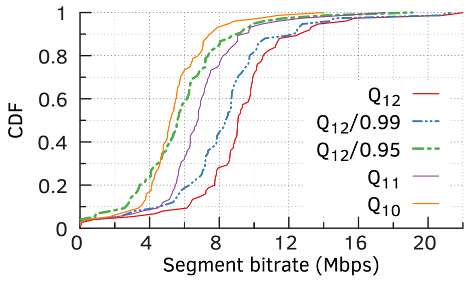
Although we can drop a significant fraction of segments while still guaranteeing a high-quality video stream (Fig. 4.1a), the frames that may be dropped are typically distributed throughout the segment. While restricting consecutive frame-drops to the segment tail increases the number of dropped reference frames to 51.75% in BBB and 46% in ToS ("*/Tail" lines in Fig. 4.2a), the total number of frames that can be dropped is much smaller than that with the newly proposed frame-ranking system. This ability to identify more frames that can be dropped with in minimal visual quality impairment is crucial for tackling challenging network conditions.



(a) *Q_{12}, SSIM 0.99*



(b) *BBB*



(c) *ToS*

Figure 4.2: *(a) It is inefficient to only drop frames from the tail part of segments; and ABR algorithms can adjust video bitrates by also lowering SSIM scores instead of only switching quality levels, as shown for two videos—(b) BBB, and (c) ToS.*

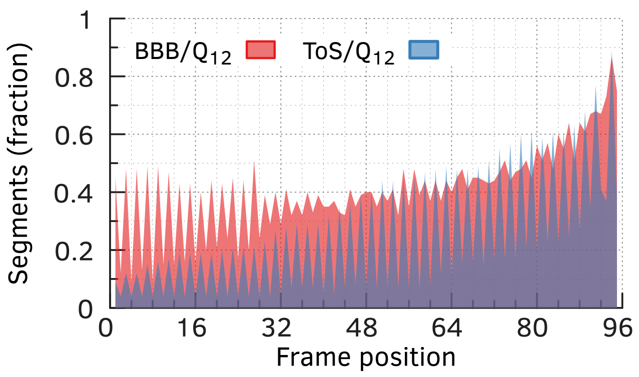


Figure 4.3: *Distribution of frames that can be dropped while guaranteeing a specific SSIM of 0.99 are distributed throughout the segments.*

Fig. 4.3 shows whether a frame at a given position across all 75 segments at Q_{12} may be dropped while still delivering an SSIM score of 0.99. A Y-axis value of 0.5 for some position implies that a frame at that position can be dropped from half the video segments, without reducing the SSIM by more than 0.01. The first frame in the segment is generally the I-Frame and cannot be dropped, and a 4 s segment at 24 fps has 96 frames. We only show these distributions exemplary in two the videos BBB and ToS but we observed unimportant frames to be distributed throughout the segment in all videos.

The distribution of unimportant frames throughout the segment presents the non-trivial challenge of designing a frame-drop-tolerant ABR algorithm. The challenge stems from conveying the "importance" of different frames to the ABR algorithm. Some frames are more important than others, and ABR algorithms should focus on downloading the important frames first. Suppose an ABR algorithm terminates the download of a segment after a particular time, say a deadline determined by when the segment needs to be rendered on the screen. The sequential download of frames within that segment, using a reliable transport, then implicitly assigns each frame a priority based on the order in which they are decoded for the segment. This default order must be altered and passed to the client in order to prioritize important frames.

BETA uses a similar frame-ordering approach, albeit they significantly differ in two respects. BETA considers only unreferenced B-Frames as "unimportant", which offer little flexibility for adapting the video quality in challenging network conditions (as we show later in §6.6). To effect the reordering BETA modifies the video files, whereas *VOXEL* only changes the manifest (refer §6.1). In a typical scenario where the videos are streamed via a CDN, small manifest updates are easier to synchronize than the comparatively large video-file changes between servers (see also §6.8).

4.2.3 Fine-grained Quality Switching via Drame-drops

Optimizing for SSIMs does not alleviate the quantization problem: We have only a discrete, finite set of quality levels to cope with the continuous variations in network throughput. Redesigning ABR algorithms to optimize for QoE and exploit frame-drop tolerance allows us to mitigate the quantization problem. Allowing an ABR algorithm to specify the number of ordered frames that can be dropped creates fine-grained *virtual* quality levels, which may align closely with and quickly to the throughput variations.

Adding more (real) quality levels instead of virtual ones does not address the quantization problem in the same fashion. The key difference with virtual quality levels is the ability to move the decision boundary from segments to frame granularity. Traditional

ABR algorithms need to deliver a complete segment. If the network conditions do not permit timely download of a segment, they can only resort to re-downloading that entire segment at a different quality level. The time spent on the already downloaded partial segment is wasted, whereas with virtual quality levels, incomplete segments are acceptable, thereby obviating segment retransmissions. Without the virtual quality levels, we observe in our experiments, for instance, that the state-of-the-art ABR algorithm BOLA retransmits near entire segment data for more than 25% of the segments, particularly in small-playback-buffer, low-latency scenarios.

Romaniak et al. [117] show that perceptual-quality-impairing artifacts caused by packet loss, or frame-drops, induce structural changes that can be captured by SSIMs. Figs. 4.2b and 4.2c show such a virtual quality in SSIM $Q_{12/0.99}$ (i.e., quality level Q_{12}, but with an SSIM score of 0.99) between qualities Q_{12} and Q_{11} for the BBB and ToS videos respectively. $Q_{12/0.99}$ maintains an excellent SSIM score of 0.99 while reducing the bitrate of all segments halfway from Q_{12} towards Q_{11}. Redesigning ABR algorithms for optimizing for QoE and accounting for frame-drop tolerance fundamentally alters the landscape of ABR algorithm design and offers several advantages over virtually all current ABR algorithms.

BETA also pursues the idea of virtual quality levels, but only drops unreferenced *B*-Frames. Besides, BETA only determines one virtual quality threshold per quality level, whereas we (as we discuss later in §6.1) analyze the losses of all combinations of *P*- and *B*-Frames. We can, therefore, determine the expected quality of a segment at any point during the download and make fine-level mid-segment quality adjustments. This greater flexibility allows us to outperform all other streaming solutions (as we show in §6.6).

4.3 Summary of our Contributions

Since video is today's king of content, we will present our cross-layer approach called *VOXEL* with the following video streaming contributions.

⋆ First, in Chapter 5, we describe an offline, server-side algorithm that rank orders frames, within each segment, based on the impact of their loss on QoE. Our algorithm reveals that at the highest quality level at least half the segments (of all videos in our tests) can sustain a 10% to 20% loss, and still offer an excellent video with imperceptible impairment (i.e., SSIM of 0.99).

⋆ Next, in §6.3, we present a novel ABR algorithm, ABR*, that combines the frame-drop-tolerance insights and the partially reliable transport to optimize directly the end-user QoE.

⋆ These ABR* in combination with QUIC* and our frame ranking are the key components of *VOXEL*, which we evaluate in §6.4. *VOXEL* outperforms state-of-the-art (BOLA with QUIC) as well as BETA, suffering little or no rebuffering, and offers excellent QoE across a wide range of conditions. Even in live-streaming-like settings over challenging cellular-network conditions, *VOXEL* suffers at least 25% and at most 97% *less* rebuffering, in the 90$^{\text{th}}$-percentile across all video segments, than the state-of-the-art.

⋆ Lastly, in §6.7, we conducted a survey to confirm the superiority of our, with objective metrics, evaluated system and gathered the opinions of actual users. We conducted the real user survey with 54 participants, 84% of whom indicated that they prefer videos streamed using *VOXEL* compared to the state-of-the-art.

In the following, in Chapter 5, we will start by explaining our frame importance ranking, followed by a system description of *VOXEL* in Chapter 6.

Chapter 5

On Frame Importance

Using our insights about video transfer, we, in this chapter focus on how to rank frames by order of importance. With this ranking, we can reorder frames in such a way that we can drop the tail end of a segment while avoiding QoE degradation. This process is nontrivial as video codecs are designed to reduce the amount of video data by referencing already encoded data, instead of retaining redundant copies of the same information, as shown in Fig. 5.1. If we allow an imperfect transmission, there is, though, still room for reducing the amount of data by dropping frames. The process of selecting which frames can be dropped, without impacting the end-users QoE will be discussed in the following. Fig. 5.2 shows two examples of frame drops with vastly different impacts on the resulting QoE.

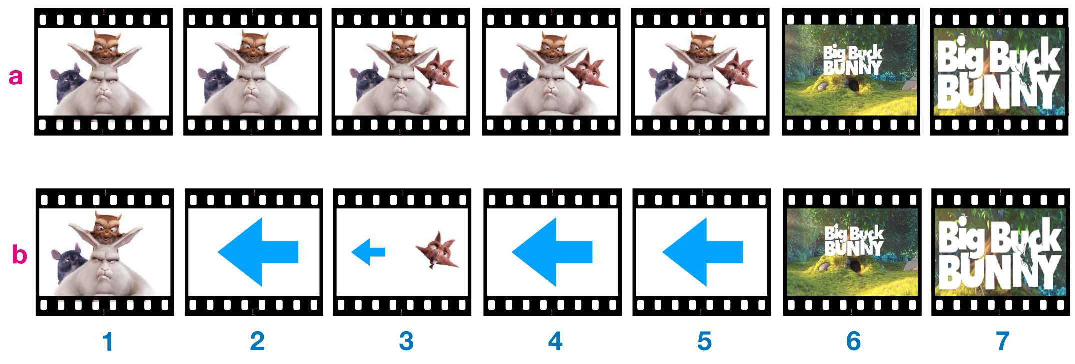


Figure 5.1: *Illustration of frame references. (a) Shows a sequence of frames taken from the Big Buck Bunny video, as full frames, without any data redundancy removed. (b) shows the same sequence of frames, with redundancy removed, e.g., frame 2 is identical to frame 1 and, thus, only a reference to frame 1 is saved. In frame 3 we see a difference resulting in a partial reference and some new content in the form of the newly appearing character. Lastly, frame 7 in almost entirely different to frame 6 and practically no reference are possible.*

As explained in §4.2, it is preferable to drop frames from the tail-end of a segment. Thus, we will download frames in order of importance to be able to drop the tail-end which then contains the least important frames. The importance of a frame depends on the

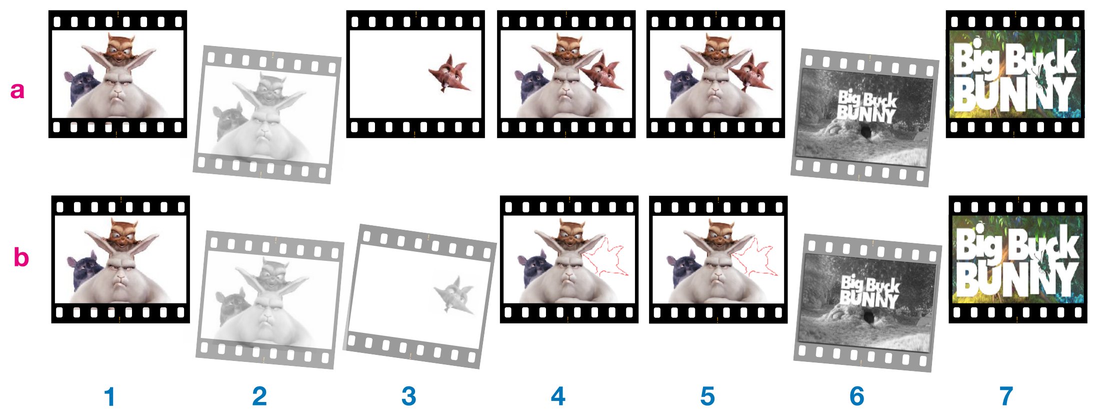


Figure 5.2: *Illustration of frame importance and its impact on the perceived quality of a video segment. (a) Dropping frames 2 and 6 have negligible impact on the QoE as frames 1 and 2 are identical and a simple continuation of displaying frame 1 conceals the loss; frame 7 is vastly different to frame 6, making the loss of frame 6 barely noticeable, i.e., there is next to no error-propagation. (b) In addition to the previous frame-drops, now frame 3 is dropped as well. Dropping frame 3 has significant implications on the following frames as it contained pixel data that is referenced by frame 4 and 5, both of which would experience a large error as there is no data to conceal the loss, showing the importance of determining which frame can be dropped.*

actual content of a video segment, meaning there is no one frame-order that yields the best QoE results. We narrowed the orders down to a set of four different prioritization orders, which we will investigate as follows.

① ***Original order***, in which we retain the frames in the same order as generated by the Moving Picture Experts Group (MPEG) encoder.[1]

② ***Order by grouping unreferenced frames***, where we move unreferenced frames, i.e., frames with no inbound references, to the end of the segment. If a segment download terminates prematurely, errors or losses in these tail-end frames will not affect other frames; the visual quality of the segment experiences, hence, no direct degradation except for a reduced playback fluidity. This order closely resembles BETA's approach.

③ ***Order by summed up inbound references***, where we sum up all inbound references towards a frame. This first approach ranks frames with more references higher. Intuitively, the more a frame is references by others, the more frames would be impacted if said frame is lost.

④ ***Order by chained inbound references***, in which we rank order frames no longer only based on direct inbound references but also by transitive references. "Unimportant" frames in the tail end of the segment will be the ones with fewer inbound references than

[1]MPEG encoders perform some limited frame re-ordering to ease decoding: They neither analyze the transitive dependencies between frames nor the number of macroblocks referenced by a frame.

those in the head end. When frame-drops occur at the tail, only a minimal number of other frames will likely be affected than when frames are retained in the other two orders.

We note that the prioritization effort does not affect I-Frames as they have the highest importance and are always downloaded first.

Finding the best among the four orderings. For each order, we estimate the implications of partial segments for QoE as follows. We iterate over the "unimportant" (tail-end) frames in each segment and calculate the QoEs (e.g., SSIMs) as a function of number of dropped frames. The process results in a mapping from the number of bytes downloaded, i.e., calculated from the frame sizes, under each ordering to QoE scores. For each quality level Q_n we use the QoE score of the pristine version of a segment at level Q_{n-1} as a lower bound. If frame-drops lower the score below this bound, we simply fetch the segment at quality Q_{n-1}. We find, therefore, the smallest number of bytes required at Q_n to achieve a QoE score higher than the bound at Q_{n-1}. The number of bytes required to satisfy a given QoE threshold varies based on the ordering, as for a given percentage of frame-drops, in the tail-end, the number of affected frames varies across the four orderings. We pick the ordering with the minimal number of bytes[2] that achieves the required QoE score. The following section individually discusses the implications the relevant orderings have on the QoE in detail.

5.1 Frame Orders and their Implications

The order in which the frames are dropped has an impact on the QoE score. As mentioned, if only consecutive frames are dropped at the tail, the decoder will simply repeat the last frame it was able to decode. The same behavior can be observed if frames are dropped at random points in the segment. If the dropped frames are, however, used as a reference by frames that occur later in the display order, the error propagates to these later frames since the decoder can not resolve the reference. This case illustrates the need to carefully evaluate different orders and their impact on the visual quality.

As a first interpretation of SSIM scores, we follow the mapping of Zinner *et al.* [118]. This mapping divides the SSIM score, ranging from 0 to 1, into intervals, and then assigns a mean opinion score (MOS) to each interval. We use this mapping as a guideline because the SSIM score range is not linear: the lower *half* of the range is considered "bad," and the top three scores are all located in the range from 0.88 to 1. The following plots contain markers (highlighted in bold font in table Table 5.1) that indicate the boundaries of the intervals. We omit the second order, "order by grouping unreferenced frames" in

[2]A client may fetch bytes beyond this threshold, if conditions permit.

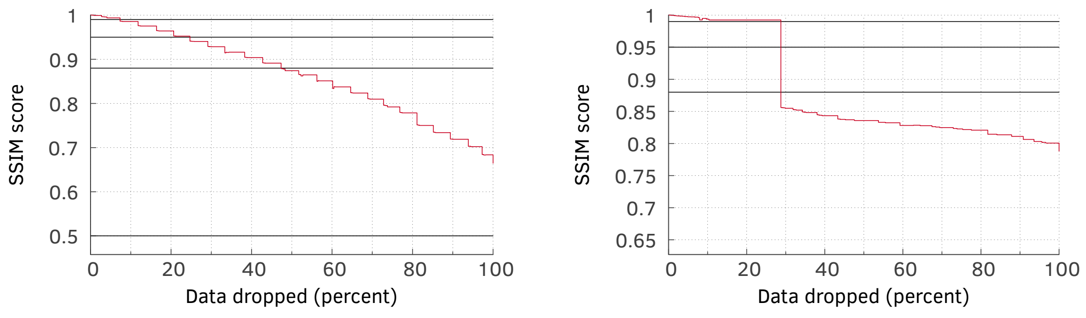
MOS	*SSIM*
Excellent	$x \geq \mathbf{0.99}$
Good	$0.99 > x \geq \mathbf{0.95}$
Fair	$0.95 > x \geq \mathbf{0.88}$
Poor	$0.88 > x \geq \mathbf{0.5}$
Bad	$0.5 > x$

Table 5.1: *Mapping of MOS to SSIM.*

this discussion as it is trivial and has no degradation implications in regards to other frames.

5.1.1 Original Order - Dropping the Tail without Reordering

We first ran computations for the simple scenario described above: Dropping consecutive frames in reverse playback order, i.e., starting from the end of the segment. Fig. 5.3 shows how the SSIM score degrades as more and more frames are dropped. Note that the x-axis is in percent of *data* dropped instead of *frames* dropped. In order to make a fair comparison between different frame-dropping orders, we need to compare the amount of data that is dropped instead of the number of frames that are dropped. Frame sizes can vary because of the variable bitrate (VBR)-characteristics of the video. For example, order 1 might achieve an SSIM score of 0.99 by dropping 10 frames with a total size of 1 MB and order 2 achieves the same score by dropping 1 frame with 2 MB. The goal of our approach is to download less data, so we would prefer order 2.



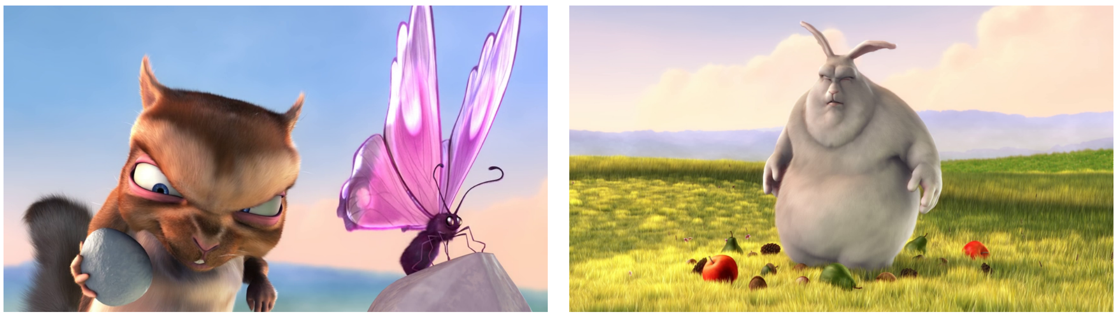
(a) *Segment 11* **(b)** *Segment 49*

Figure 5.3: *SSIM scores for segments 11 and 49 of BBB without reordering. Image quality for segments without scene changes decreases gradually (a), but drops significantly if a scene change is present (b).*

Looking at Fig. 5.3a, the segment without scene change, we observe that the score decreases gradually. The gradual quality decrease is expected because although the time a still image is displayed increases, its contents are still similar to the reference, which leads to a comparatively low degradation. The segment in Fig. 5.3b contains a scene change, which is reflected in the SSIM graph. This scene change occurs at frame 44 (Fig. 5.4), which corresponds to 29% of data dropped (i.e., frames 44 to 96 dropped). As expected, the image quality drastically decreases as soon as the still image is displayed before the scene change occurs (i.e., frame 43 or earlier; at least 29% data dropped), but stays almost perfect with less than 29% data dropped.

Another observation is that even if 100% of the data is dropped (i.e., the I-frame is repeated 95 times), the SSIM score only decreases to a value of 0.66 for segment 11 and 0.79 for segment 49. This is because the SSIM score is not linear and therefore the "poor" quality category already starts at scores less than 0.88.

In the following we analyze if a dropping frames in a different order enables us to maintain a higher SSIM score for a longer time. Recall, unimportant frames should be dropped first so that important frames can be kept. To specify a new order means to assign a notion of importance (weight) to each frame.



(a) *Frame 43* **(b)** *Frame 44*

Figure 5.4: *Scene change in segment 49 of BBB.*

5.1.2 The Summed Reference Order - An Intermediate Step

A first, simple, approach is to use the sum of references pointing to a frame as the weight. Naturally, if a frame that is referenced by many other frames gets dropped, it should have a greater impact on video quality than dropping a frame with no references. The computation results of this new order are shown in Fig. 5.5.

We can see an improvement for segment 49 (Fig. 5.5b), i.e., we can drop more data before encountering the quality decrease caused by the scene change. However, there is a problem with segment 11 (Fig. 5.5a): The new order introduces a "cliff" where there should be none. A closer look at the frame-level scores of the data points before and after the cliff gives insights into what happened.

Recall that each step in Fig. 5.5a represents an SSIM score for segment 11 with a number of frames dropped. For example, looking at the "reorder" line, the step starting at 29%

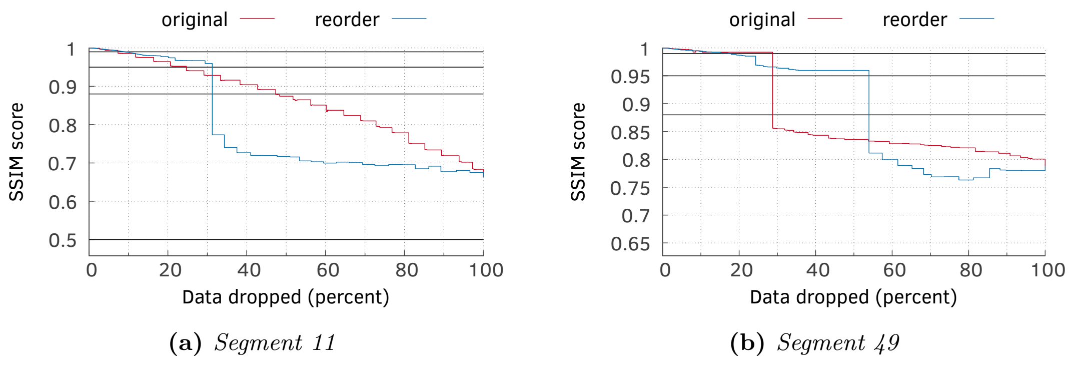


Figure 5.5: *Comparison of SSIM scores for summed reference order with no reordering for segments 11 and 49 of BBB. The summed reference order delays quality degradation for segments including a scene change (b), but introduces a significant quality decrease for segments without scene changes (a).*

of dropped data has an SSIM score of 0.96. However, this final score of 0.96 is the result of averaging the 96 individual frame scores of the segment. Fig. 5.6 expands three steps from Fig. 5.5a by visualizing these individual frame SSIM scores.

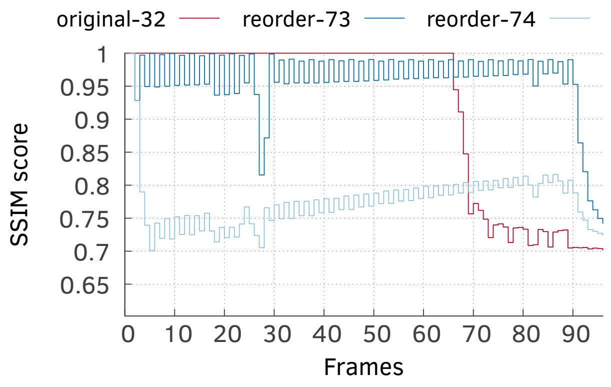


Figure 5.6: *Comparison of frame-level SSIM scores for summed reference reordering and no reordering for segment 11 of BBB. A difference of a single dropped frame can have a great impact on visual quality.*

Although there is only a single frame difference between the two reorder cases, it has a huge impact on the SSIM score. “reorder-73” (“reorder-74”) expands the step directly before (after) the cliff in Fig. 5.5a. It corresponds to 73 (74) dropped frames, which amounts to 29.44% (31.25%) of data dropped. The corresponding segment SSIM score is 0.96 (0.77). The closest step in terms of data size for the “original” line in Fig. 5.5a is expanded as “original-32” in Fig. 5.5a and corresponds to 32 dropped frames (33.35% data dropped) resulting in a segment SSIM score of 0.92. Because the step size granularity of our calculations is limited to entire frames dropped, we can not make a comparison of the two orders with exactly equal amounts of data being dropped.

From Fig. 5.6 we can observe that the reason for the difference in SSIM scores is that the frame in question is dropped early in the segment, so the errors introduced by its absence propagate through the segment. This motivates a need to refine the order in which the frames are dropped.

5.1.3 The Chained Reference Order

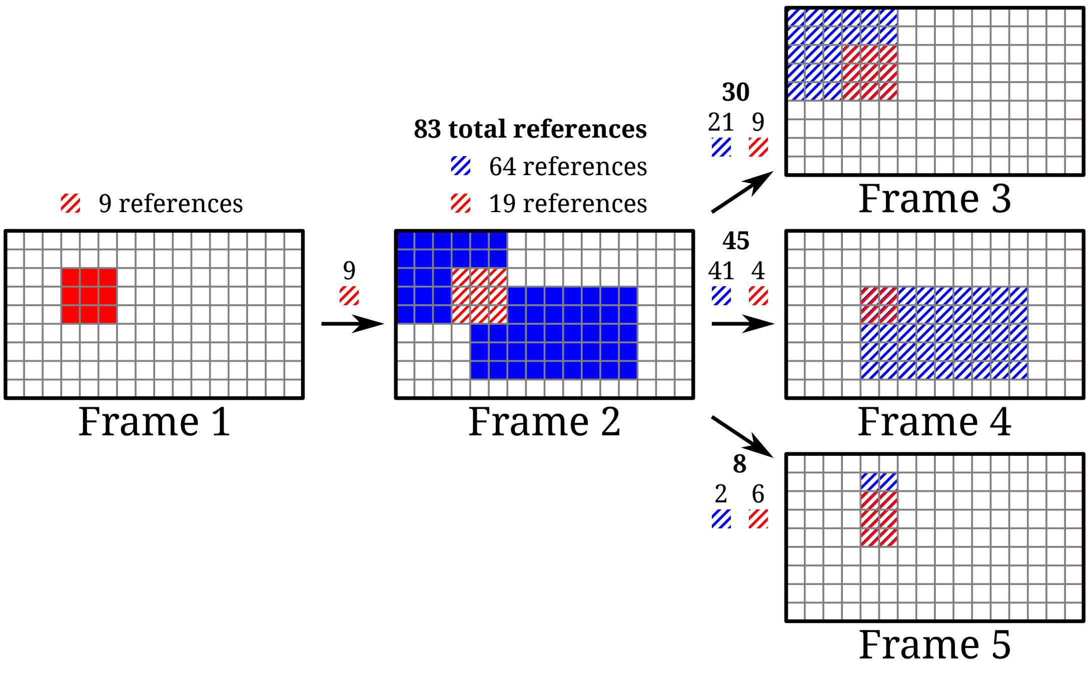


Figure 5.7: *Example for a reference chain with five frames. Original macroblocks are visualized by a solid color, whereas macroblocks that are created by reference are striped.*

The results of §5.1.2 indicate that there are frames which are not heavily referenced but still important because they are heavily used indirectly. To capture this, we transitively increase the importance (weight) of a frame that is at the beginning of a "chain" of references.

Fig. 5.7 visualizes a small example with five frames. Frame 2 references nine macroblocks from frame 1 (striped red blocks). Frame 2, in turn, is referenced by three other frames (3 to 5) with a total of 83 references (striped blue and red blocks). These references, however, can be divided into two groups. References to macroblocks that occurred in the reference frame for the first time, like the solid red blocks in frame 1 or the solid blue blocks in frame 2, are referred to as *direct* references. A referenced macroblock that was created by another reference, i.e., to a third frame, is called a *transitive* reference. An example for this are the striped red blocks in frame 2, which are referenced by frames 3 to 5.

If frame 1 was dropped in this example, then the quality of frame 2 would decrease because of the missing direct references. Furthermore, frames 3 to 5 would also be

Algorithm 1 Graph traversal algorithm used to calculate the chained-reference weight of a node.

```
To calculate the weight of a node $x$, WALK($x, \infty$) is called.
function WALK(node, max_weight)
    weight = 0
    for suc in node.successors do
        if EDGEWEIGHT(node, suc) < max_weight then
            max_weight = EDGEWEIGHT(node, suc)
        end if
        weight = weight + max_weight
            ▷ threshold refers to the minimum weight required to continue traversal
        if max_weight < threshold then
            continue
        end if
        weight = weight + WALK(suc, max_weight)          ▷ Depth-first traversal
    end for
    return weight
end function
```

impacted due to the error propagation caused by the transitive references. For this reason, the weight of frame 1 should not only be measured by the direct references of frame 2, but also by the transitive references of frames 3 to 5.

The algorithm that is used to calculate this weight is described in Algorithm 1. There are two changes to the example from above. The first change is that in order to simplify the algorithm we do not utilize the exact positions of the referenced macroblocks, but only their amount. The algorithm does, therefore, not determine the precise amount of transitive references and will use the total number of references as an upper bound (lines 5 to 7). The references are modeled as a **DAG!** (**DAG!**).

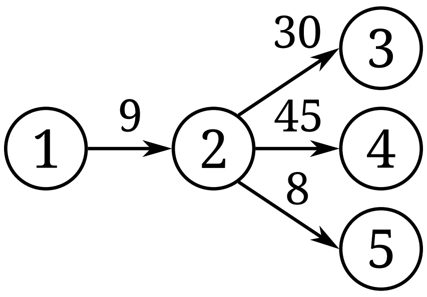
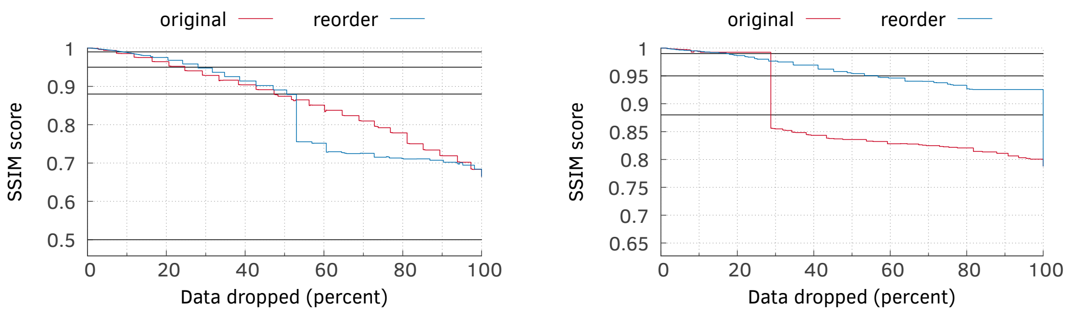


Figure 5.8: *Graph view of a reference chain.*

Fig. 5.8 shows the corresponding **DAG!** to the example from Fig. 5.7. Frame 1 has an initial weight (and upper bound) of 9 for the direct references from frame 2. For the transitive references from frames 3 to 4 an additional weight of 18 (9 per frame) is added. Note that the real amount of transitive references from frame 4 is only four macroblocks. But as mentioned above the algorithm does not calculate this exact information. Instead, the algorithm calculates that 45 macroblocks of frame 2 are referenced by frame 4 and thus the upper bound is applied. In contrast, the additional weight gained from frame 5 is only 8 since the weight of the edge from frame 2 to frame 5 is below the upper bound. Here line 6 is applied and a new upper bound is set, which is valid for the remaining traversal of this branch. The resulting total weight for frame 1 is 35.

To limit the length of reference chains, we introduce a minimal weight threshold (line 10) that is required for an edge to be traversed. In the current implementation this threshold is set to 1% of the image size, which amounts to 324 macroblocks for a frame size of 4K (3840x2160px / 16x16px = 32400). This threshold is only used as a decision to continue traversal. We still account for small weights in line 8, but do not traverse deeper into the graph. This is important in order to not lose the distinction between frames with absolutely no references and frames that have at least some references.

We re-ran the computations with this new order and we take a look at segments 11 and 49 of the Big Buck Bunny video in Fig. 5.9.



(a) *Segment 11* **(b)** *Segment 49*

Figure 5.9: *Comparison of SSIM scores for chained reference order with no reordering for segments 11 and 49 of BBB. The chained reference order manages to prioritize the scene change correctly (b), but still introduces a significant quality decrease in segments without scene changes (a).*

We can see an improvement for both cases, but we still introduce a cliff in segment 11 (Fig. 5.9a). The steps up to the cliff are better for the "reorder" line, but worse afterwards. For segment 49 (Fig. 5.9b) the reordering worked perfectly, pushing the cliff all the way to the right of the graph.

5.2 Summary

There is no one-fits-all approach to prioritizing frames. While the chained reference order works well for segments containing a scene change, it is not so clear for other segments. In order to find the most suitable solution, we need to evaluate, for each segment individually, which order works best. However, asking which order is "the best" first requires to answer another question: What SSIM score do we want to achieve? Intuitively speaking, as the lines of the different orders in the previous plots intersect, the "best" order changes depending on the target SSIM score. Optimizing for a "good" SSIM ($\geq$ 0.95) in Fig. 5.5a would result in the summed reference order being better (29% of data dropped compared to 22%), but optimizing for "fair" ($\geq$ 0.88) results in the opposite (still 29% compared to 44%).

Next, we will look *VOXEL*, a system we designed to show how to take advantage of our ranking orders.

Chapter 6

VOXEL

To take advantage of our insights from §4.2 and our frame importance ranking from §5, we created *VOXEL*, a complete and easily deployable streaming system. Streaming video using *VOXEL* is similar to traditional systems, but with several important differences highlighted as follows.

⋆ We extend the manifest with an order of frames to prioritize "important" frames (i.e., to download them before the "unimportant" ones) and mark their influence, or lack thereof, towards the QoE. The frame order is prepared offline by identifying the "important" frames in each video segment, and saved on disk, i.e., all subsequent streams of this video reuses the generated order.

⋆ At the transport layer, we build on the QUIC extensions from [110] and offer a partially reliable transport for exploiting the insight that not all frames require reliable delivery.

⋆ At the application layer, we present a new class of ABR* algorithms that use QoE metrics as the optimization utility and allow the delivery of partial segments. ABR* exploits virtual quality levels, obtained by dropping "unimportant" frames, to quickly adapt to varying network conditions. We show *VOXEL*'s backwards compatibility with existing clients and means for incremental deployment. The following sections will describe the highlighted difference to traditional streaming systems and show their advantages.

6.1 Extending the Manifest

Before extending the manifest, we are, similar to most streaming solutions, preparing video content for streaming with *VOXEL* starts with a transcoding phase. We transcode the content into a number of different bitrates and slice them into segments, which can be presented via the DASH manifest [119]. Referring to Section 5, we add an extra step or phase to *prioritize* frames within each segment. The prioritization helps a client to download the frames of a segment in an order that, even if the download is prematurely terminated, likely improves the QoE of the partially downloaded segment. This reordering does ***not*** involve any change to the video files, but only enriches the manifest by specifying the order in which different byte-ranges of the video-segments are downloaded.

We, now, update the manifest with frame-level details (see Listing 6.1) based on the chosen ordering. We clearly demarcate the subset of data that requires reliable delivery ('reliable' and 'unreliable' byte-range attributes in Listing 6.1), and a client can fetch the byte ranges via HTTP `range requests`. This type of request renders it unnecessary to alter the video files. Lastly, we add the mapping from bytes-fetched-at-a-quality-level to the QoE scores, to assist an ABR algorithm in making better decisions. We show, for instance, in the 'ssims' attribute in Listing 6.1 comma-separated tuples, with each tuple containing a colon-delimited triplet: (a) A QoE score, e.g., SSIM, and the number of (b) frames and (c) bytes of the given segment that must be downloaded to achieve that QoE score. For the videos in Tab. 6.1 in §6.4.2, the updates increase the manifest size to approximately 16% of the size of an average Q_{12} segment. This size overhead can, however, be mitigated by using a better encoding scheme for the metadata than the naïve, unoptimized version we used in our proof-of-concept implementation. We can also reduce any startup delays introduced by a large manifest simply by incrementally downloading the manifest using the *MPD update* feature of DASH [120].

Listing 6.1: *A frame-level entry from VOXEL's manifest.*

```
<SegmentURL mediaRange="367500239-374182132"
  ssims="0.988:49:4303546,...,0.999:93:5222995,1.0:95:5310048"
  reliable="367500239-367501146,...,374125556-374125570"
  unreliable="370076394-370171472,...,369318627-369389193"
  reliableSize="1371846"/>
```

A size vs. compatibility tradeoff. A key benefit of the extended manifest, despite its size, is that, unlike other prior work (e.g., BETA) we do not require any modification to the video files on the server. *VOXEL*-aware clients can exploit the additional metadata (frame-level ordering) and download the frames in the best order. *VOXEL*-unaware clients, in contrast, ignore the frame-level metadata and simply download segments in the original or decoding order.

6.2 QUIC*: Enriching the Transport Layer

We designed a modified version of QUIC, referred to as QUIC*, that supports not only ("vanilla") QUIC's reliable streams but also unreliable streams with *optional* retransmissions. The unreliable streams of QUIC*, unlike UDP, are subject to the congestion (CUBIC) and flow-control mechanisms of the QUIC connection. We borrow some design principles from the QUIC extensions of Palmer et al. [110], albeit our design significantly departs from theirs. While we also introduce a new unreliable stream, for instance, we avoid a separate control stream as in [110].

Interfacing transport and application layers. We use HTTP headers to connect the transport and application layers. A client that wants to open an unreliable stream, for instance, sends an HTTP `GET` request and includes the custom `x-voxel-unreliable` header. A *VOXEL*-aware server would open an unreliable stream to the client and deliver the response over that stream. A *VOXEL*-unaware client, for instance, will simply not use the custom header, and it will receive the response via a reliable stream. A *VOXEL*-unaware server ignores the header and opens reliable streams only.

Life cycle of a video session. A client begins the session by downloading the manifest, either in its entirety or incrementally over time. With the details for fetching the next segment available, a *VOXEL* client uses two sequential HTTP requests as follows. The first request fetches the I-Frame and headers of all frames (i.e., 'reliable' attribute in Listing 6.1) over a reliable stream. This second request downloads the video data for a subset of the remaining frames (i.e., 'unreliable' attribute in Listing 6.1) over an unreliable stream. This subset is determined by the QoE score and the number of frames required to achieve that score based on the 'ssims' attribute. As a result, depending on network conditions, some packets, i.e., some parts of frames, may be lost in transit. How to cope with the losses that may introduce some QoE deterioration will be discussed next.

Enabling selective retransmissions. We follow a two-pronged strategy for retransmissions: Given a target QoE score (e.g., SSIMs), we can determine whether the loss will lower the QoE score below the target value; we can, hence, avoid needless retransmissions. We also exploit typical video-client behavior to opportunistically retransmit lost data: Video clients typically do not download *new* segments when the playback buffer is full. *VOXEL* use any such periods to re-request lost data on the unreliable stream via HTTP `range requests`. We gather the loss information in the (QUIC) transport layer and pass it up to the application layer that then generates the corresponding HTTP request ranges. We stop any selective retransmissions, immediately, if conditions become unfavorable (e.g., buffer occupancy drops), to avoid introduce rebuffering issues. Yet, we recover all losses in small buffer scenario and have a remaining loss of only 0.9%, 1.5%, 1.8% for 2-, 3- and 7-segment long buffers.

Handling partially downloaded segments. Since we always fetch the *I*-Frames and all frame headers reliably, when ending up with partially downloaded segments, we have precise knowledge about the losses, i.e., which frames were affected, and to what extent. We use this information to *mask* the “holes” in the frames by simply zero-padding the losses. The QoE score, e.g. SSIM, calculations are performed on the decoded padded frames. With frame headers kept intact and at the expected position in the file, decoding was no issue. The decoder utilizes error-concealment techniques for zero padded frames and only replaces a frame with the previous one if said frame is missing entirely.

A quality vs. rebuffering tradeoff. Skipped frames indeed have implications for end-user QoE. We systematically tradeoff the losses, however, to avoid rebuffering, since the latter significantly degrades user experience. Our experiments in §6.4 show that *VOXEL* significantly reduces rebuffering, particularly in scenarios with smaller buffer sizes. Confirming, as of 2020, rebuffering to be the most frustrating [12, 121], our user survey shows that an overwhelming majority of participants prefer trading off buffering for quality.

6.3 ABR*: Enhancing the ABR Algorithm

VOXEL provides a framework for a new class of ABR algorithms with the following key features.

6.3.1 Optimize for QoE

ABR algorithms such as MPC [41] and BOLA [42] optimize a utility function often based on bitrate. Both algorithms allow, however, different utility functions. *VOXEL* uses a QoE-metric-based utility. While we use SSIM as the QoE metric for most of our evaluations, *VOXEL* is metric-agnostic, and we show that our results generalize to other metrics such as VMAF and PSNR.

6.3.2 Support Partial-segment Downloads

Traditional ABR algorithms choose from a limited set of bitrates. *VOXEL* allows partial segment downloads while ensuring frame-header integrity (§6.2) and presents the respective QoE metrics for different download subsets (Listing 6.1), thereby significantly increasing the available decision space. This frame-header integrity enables decoding of segments with missing referenced-frame data. *VOXEL* has, thereby, significantly more freedom in dropping frames than prior work that only allows unreferenced B-Frame drops [105].

6.3.3 Segment Abandonment Options

State-of-the-art ABR algorithms (e.g., BOLA) support segment abandonment, albeit in a narrow scope: They simply *discard* a high-bitrate segment download and restart a low-bitrate download if a high risk of rebuffering is detected. *VOXEL* extends this idea in a key way: We retain the partial segment and move on to the next, *even if*, compared to recent work, referenced frames are missing. This extension allows us to download fewer bytes than other ABR algorithms (such as BOLA and BETA) during periods with less-than-ideal network conditions. Also, a partial high-bitrate segment might give better QoE than a complete low-bitrate segment (refer §4).

To complete our *VOXEL* implementation, we designed ABR*, a novel ABR algorithm based on BOLA. The decision to bootstrap ABR* using BOLA was manifold. BOLA already supports low-buffer scenarios [122]. BOLA allows for a custom utility function by design, and has only two tuning parameters γ and V. Before streaming, *VOXEL* automatically tunes γ and V for the video's bitrate ladder characteristics following a calculation described in [42]. Further, the complexity of choosing a segment's quality is linear in the number of qualities available, which is particularly useful since the partial download option can significantly increase the number of qualities. Finally, BOLA sees industry adoption and is already integrated in the *dash.js* reference player [123].

We developed ABR* by extending BOLA-E, a variant of BOLA, described in [122], with two updates. First, we changed the utility function to use SSIMs and added the capability to select partial-segment downloads. We refer to this intermediate step as BOLA-SSIM. We then extended BOLA's segment abandonment option to keep a partial segment and move on to the next download. We refer to this final step as ABR*.

While we based ABR* on BOLA, *VOXEL* will work with other ABR algorithms, either novel or updated established algorithms. A few design aspects warrant, however, further attention. For example, it is relatively simple to update MPC to use a QoE metric as the utility function. MPC, however, searches the entire decision space within a window, typically around five segments into the future. Thus, the large decision space provided by *VOXEL* would require further modifications to MPC to curb the search space.

Table 6.1: *Overview of evaluation videos from prior work.*

Video	*Genre*	*Std. dev. (Mbps)*	*Range (Segments)*
Big Buck Bunny (BBB)	Comedy	3.77	1–75
Elephants Dream (ED)	Sci-Fi	5.6	39–113
Sintel	Fantasy	7.5	148–222
Tears of Steel (ToS)	Sci-Fi	3.52	1–75

6.4 Evaluation Scenario

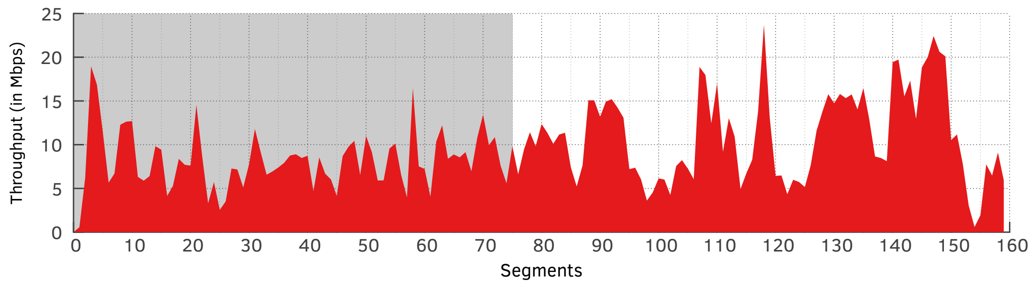
We now evaluate the efficacy of *VOXEL* by incrementally deploying the different components and measuring the implications of such a rollout for video-streaming performance.

6.4.1 Video Selection

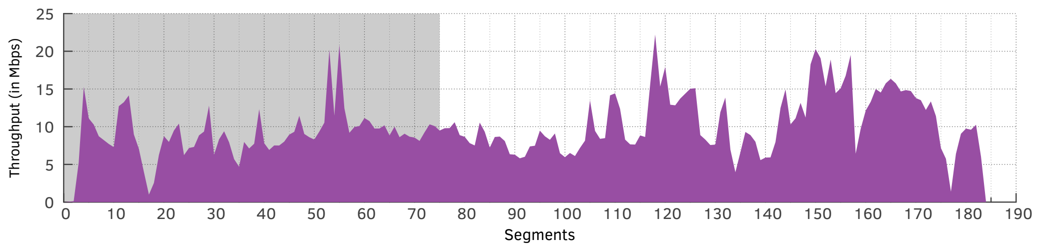
For both demonstrating the key insights and evaluating the design of *VOXEL*, we chose 14 videos. For the majority of our evaluations, we restrict our attention to the 4 videos we used earlier in §4. The 4 chosen videos are, in video-streaming research, widely used (Tab. 6.1) e.g., [124–126]. While *VOXEL*'s performance varies across different videos, its relative performance holds across all the videos. From each video, we chose five-minute long subsections (75 segments) to get different and challenging bitrate variations (see Fig.6.2). Following the encoding procedure outlined in [127], we produced "2x capped" VBR videos. We used `FFmpeg` version 4.1.3, and transcoded 13 quality levels (Q_0 with 0.16 Mbps through Q_{12} with 10 Mbps), as per Tab. 6.2. Transcoding was done using 2-pass encoding, preset *slow* and no custom encoding settings. Unlike [127], we used 4 s segments, which are a good balance between encoding quality and fast quality switching [86]. The resulting videos are, in percent of bytes, comprised of ≈15 % *I*-Frames, ≈65 % *P*- and ≈20 % *B*-Frames. We describe the video files and transcoding process in detail in §6.4.2.

6.4.2 Widely used Videos from Prior Work

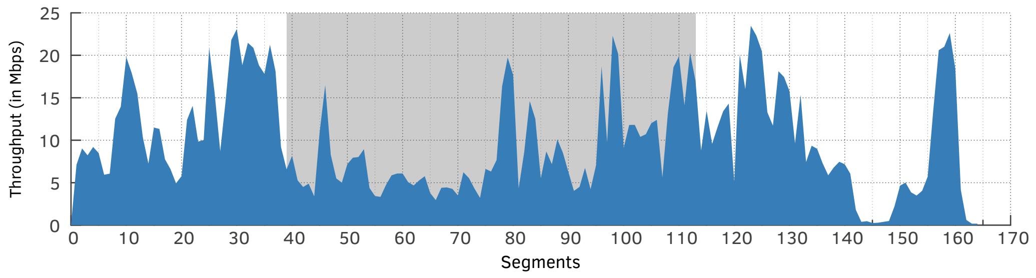
Our 4 widely used videos from Tab. 6.1 are: *BBB, ED, Sintel, and ToS.* Since the actual videos are quite long in duration—ranging from ≈10.5 min to ≈15.75 min—we chose five minutes (75 segments) from each video, to obtain video clips with different, challenging bitrate variations (shown in Tab. 6.1 and illustrated in Fig. 6.1). Starting with BBB in Fig. 6.1a, where the selected segment region has a steady standard deviation throughout. We chose this region to potentially induce osculations in the quality level selection of ABR algorithms. ToS, in Fig. 6.1b has a large drop in bitrate, starting



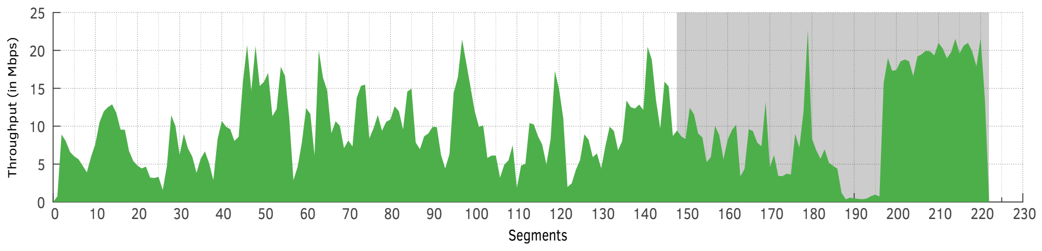
(a) *Big Buck Bunny (BBB)*



(b) *Tears of Steel (ToS)*



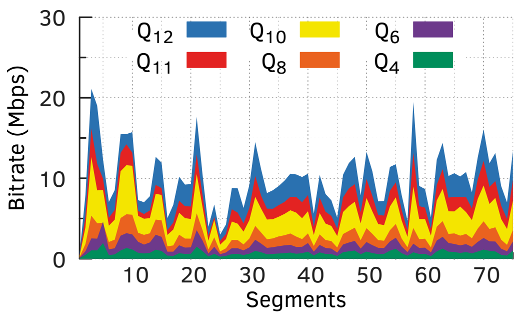
(c) *Elephants Dream (ED)*



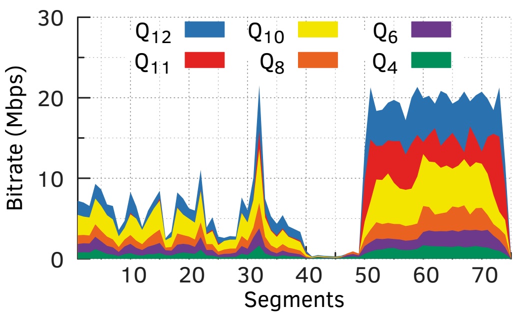
(d) *Sintel*

Figure 6.1: *The dark shaded background area depicts the segments selected from the four widely used videos. The selection was made to cover different and challenging bitrate variations.*

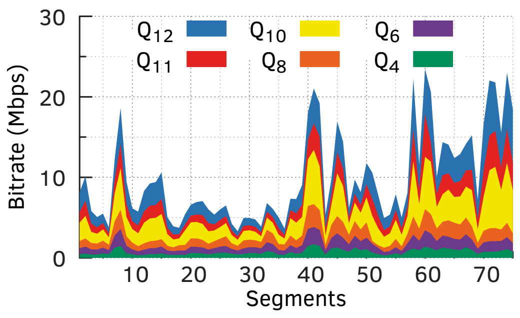
from segment 12 and two large spikes at segment 55. They challenge ABR algorithm's reactivity to sudden changes in bitrate. ED, in Fig. 6.1c is similar but contains more and larger spikes, challenging the algorithms even more. Lastly, Sintel's selected segments, in Fig. 6.1d, show a continuous drop in bitrate, until the very low bitrate area around segment 190, with a huge and sudden increase of bitrate that continues until the end. We transcoded the videos using `FFmpeg` version 4.1.3, at 13 quality levels (Q_0 with 0.16 Mbps through Q_{12} with 10 Mbps), as per Tab. 6.2. As ED was not available in 4K,



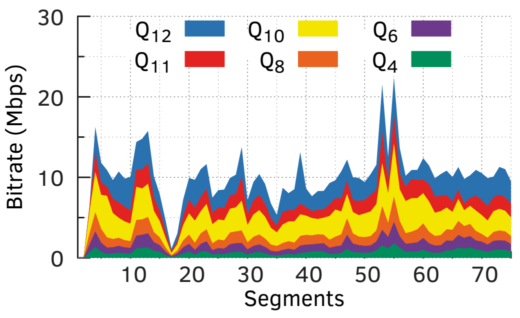
(a) *Big Buck Bunny (BBB)*



(b) *Sintel*



(c) *Elephants Dream (ED)*



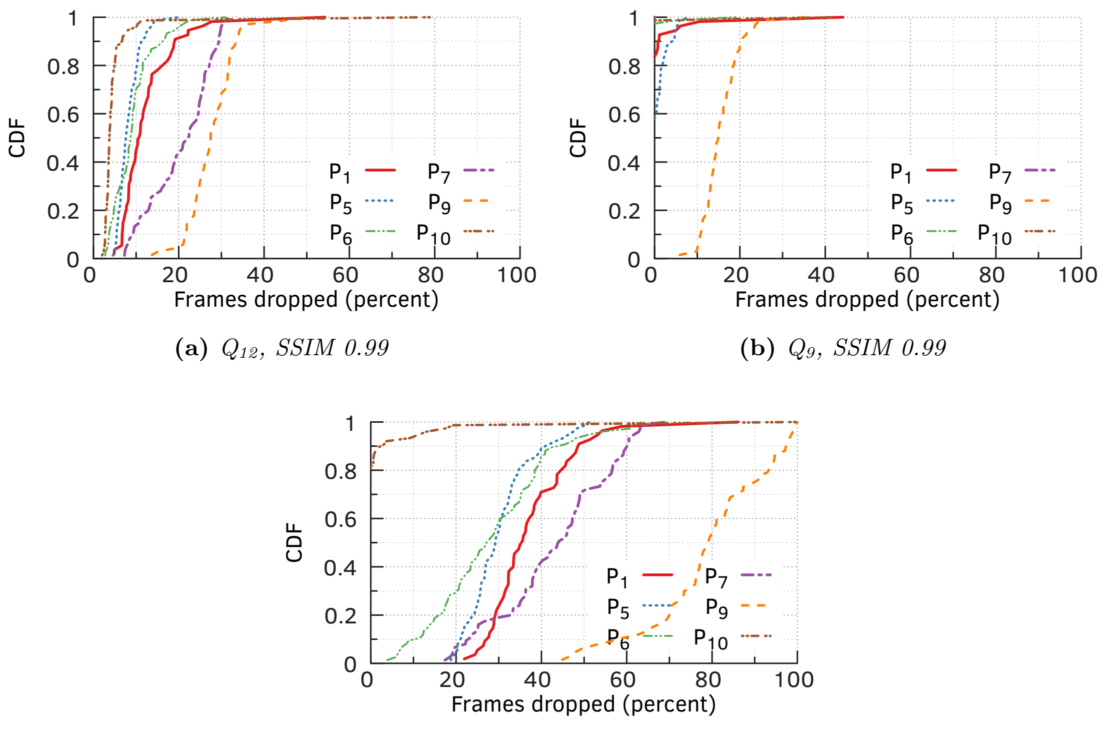
(d) *Tears of Steel (ToS)*

Figure 6.2: *Variations in segment sizes across a subset of 6 quality levels of the 13 available quality levels for the four widely used videos.*

Q_{11} and Q_{12} were, thus, encoded as 1080 p. The levels are based on common 16x9 aspect ratio resolutions, and bitrates from a combination of the "bitrate ladders" of YouTube obtained via *youtube-dl* [128], and Netflix [129]. These capped VBR videos have a peak bitrate of at most 200% the average bitrate (see Fig. 6.2) and 24 fps. The clips exhibit (in Fig. 6.2), depending on the content, vastly different bitrate variations across the video segments. This encoding increases the visual quality of the videos, since it uses more bits (i.e., incurs a high bitrate) in segments where they are most needed.

Table 6.2: *Quality levels of encoded videos.*

Resolution	*Quality Level*	*Avg. Bitrate (Mbps)*	*Total Size (MB)*
144 p	Q_0	0.16	5.8
240 p	$\{Q_1, Q_2\}$	$\{0.23, 0.37\}$	$\{8.5, 14\}$
360 p	$\{Q_3, Q_4\}$	$\{0.56, 0.75\}$	$\{21, 27\}$
480 p	$\{Q_5, Q_6\}$	$\{1.05, 1.75\}$	$\{38, 63\}$
720 p	$\{Q_7, Q_8\}$	$\{2.35, 3\}$	$\{84, 108\}$
1080 p	$\{Q_9, Q_{10}\}$	$\{4.3, 5.8\}$	$\{154, 207\}$
1440 p	Q_{11}	7.4	264
2160 p	Q_{12}	10	357



(a) *Q_{12}, SSIM 0.99* **(b)** *Q_9, SSIM 0.99*

(c) *Q_9, SSIM 0.95*

Figure 6.3: *The insights from §4 hold true for a diverse video set. (a) A significant number of frame-drops can be tolerated while guaranteeing an SSIM of 0.99; The frame-drop tolerance (b) diminishes when reducing the quality, but (c) improves when lowering the target SSIM from 0.99 to 0.95.*

6.4.3 Public YouTube Videos

To confirm whether the insights in §4 hold for a broader spectrum of videos, and not only for the, although, widely used videos in Tab. 6.1, videos, we analyzed the remaining set of our video collection.

The remaining set consists of 10, see Tab. 6.3, publicly available videos from Youtube. In Fig. 6.3, we present the analyses of some of the videos that were not already presented in §4. For readability, we omit P_3 and P_8, as the results from these do not offer any additional insights.

Our observations concerning frame-drop tolerance for most of these videos, were similar to those drawn from the four videos from prior work (in Tab. 6.1). There were, however, two exceptions: P_9 and P_{10}. For the rest, at quality level Q_{12} half of the segments can tolerate at least 10% frame-drops while maintaining an SSIM score of 0.99 or higher. In case of P_{10}, only 4% segments can tolerate 10% frame-drops or more, while for P_9, we can drop 14% of frames from *all* segments. At a quality level lower than Q_{12}, it is nearly impossible to tolerate frame-drops when streaming P_{10}, even when targeting an SSIM score <0.99 (Fig. 6.3c). At Q_9 with a target SSIM score of 0.95, for instance, only 18% of the segments of P_{10} can tolerate a frame-drop. P_9, on the other hand, can tolerate 80% frame-drops for at least half of the segments.

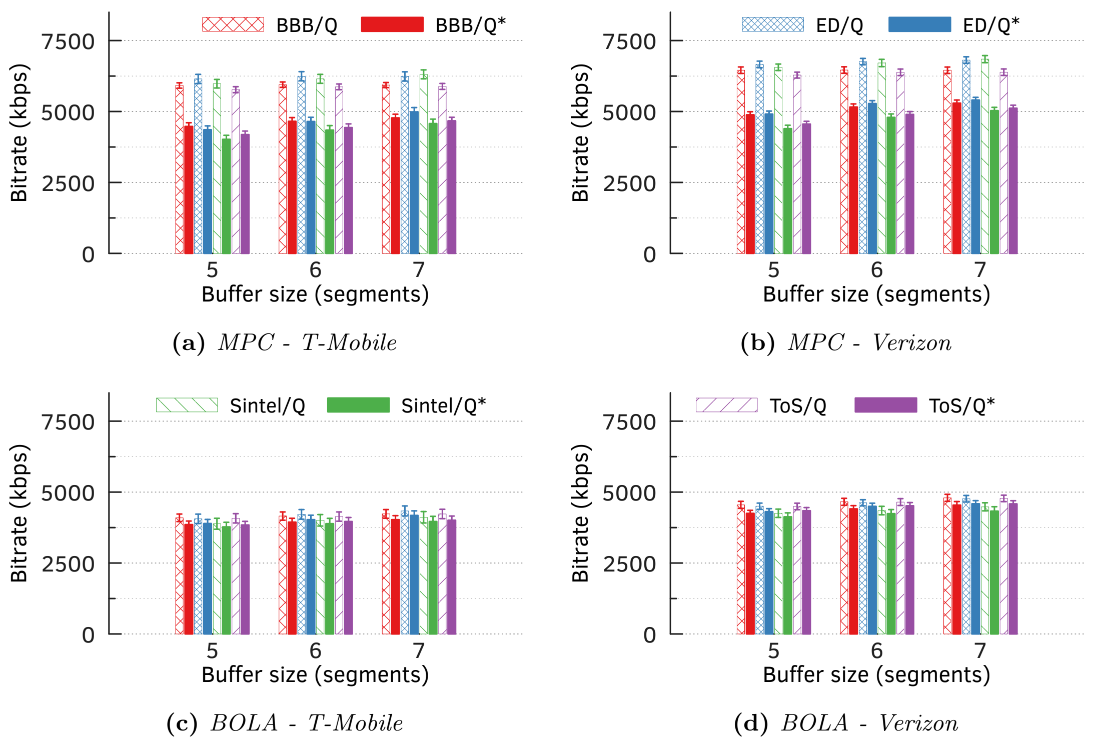
The striking difference in frame-drop tolerances between P_9 and P_{10}, and also between these two videos and the rest can partly be explained by looking at the content of these two video clips. P_{10} is a Japanese street-dance performance with roughly 50 performers; the video involves many subjects and a lot of "changes" (or dance movements) captured across the different frames. In addition, the video clip has no "cuts" or scene changes. The combination of all these factors implies that regardless of where frame-drops occurs in a segment, the resulting (decoding) errors propagate to the end of the segment. P_9, in contrast, is an "unboxing" video, which mostly features a presenter standing against a gray background, or involves a top-down view of his hands against a white table (or background). From one frame to another, the video shows little movement, i.e., changes, if any, are constrained to a relatively few macroblocks. As a result, the video can tolerate a substantially large number of frame-drops without significantly degrading the SSIM score.

6.4.4 Network Testbed

We ran all in-lab experiments on a testbed consisting of multiple sets of three bare-metal Linux machines running Linux (Debian 9) with the *4.19* kernel. Each triplet emulates a one-hop network—a server and client connected via an intermediate host (or router). We *shape* the traffic flowing through the router using the `tc` utility in Linux. Depending on the experiment, we either fix the available bandwidth to accommodate our video stream and a certain amount of competing traffic, or change the available bandwidth on a per-second basis to mimic prerecorded network traces. As the router constitutes

Table 6.3: *Overview of public YouTube videos used to generalize our insights to a diverse set ot Internet videos.*

Channel	*Category*	*Std. dev. (Mbps)*	*Range (Segments)*
Brooklyn and Bailey (P_1)	Beauty	2.2	1–55
CollegeHumor (P_2)	Comedy	1.88	56–131
Dude Perfect (P_3)	Sports	2.52	5–80
FaZe Adapt (P_4)	Gaming	2.05	2–77
Gordon Ramsay (P_5)	Cooking	1.76	1–74
Katy Perry (P_6)	Music	4.35	23–98
Tana Mongeau (P_7)	Entertainment	2.03	33–108
The Young Turks (P_8)	Politics	1.6	4–79
Unbox Therapy (P_9)	Tech	1.7	1–67
CHARI Yosakoi ch (P_{10})	Entertainment	1.94	3–78



(a) *MPC - T-Mobile*

(b) *MPC - Verizon*

(c) *BOLA - T-Mobile*

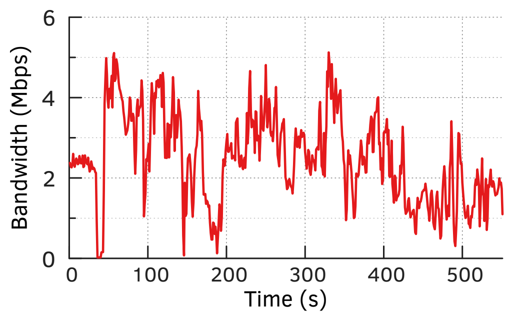
(d) *BOLA - Verizon*

Figure 6.4: *When testing with unreliable streams "vanilla" MPC trades off average bitrates for lowering rebuffering ratios in all settings. BOLA, however, is unable to balance such a tradeoff in all settings.*

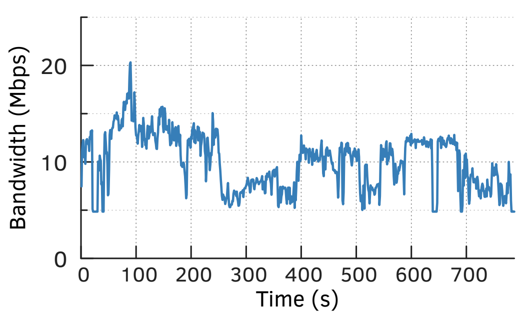
a bottleneck in the internet, assuming that not every video is cached on-site, we fixed the network queue size to 1.25 × the bandwidth-delay product. To cover a cached-video scenario, we ran the same experiments with a large, 750 packets long, network queue in §6.6.2. We configured a 30 ms delay on the router-to-client link, to emulate a typical "last mile" latency.

6.4.5 Network Traces

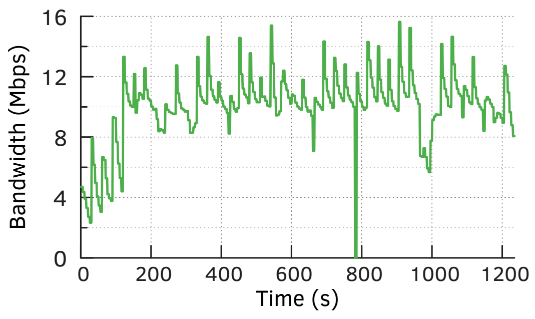
Our in-lab experiments utilized 5 different network traces: 3 LTE (4G) traces from [130], a 3G trace collected in Norway from [131], and a fixed-line broadband network trace from the Federal Communications Commission (FCC) dataset [132]. We primarily focus on cellular networks for two reasons: (a) Video traffic constitutes a substantial portion of IP traffic on mobile networks [133], and, most important, (b) they present the most challenging conditions for video streaming. We linearly offset the throughput of the traces to ensure that the average rate matches the 10 Mbps bitrate of the highest video bitrate (i.e., Q_{12}). We set the network queue to 32 packets accordingly. The adjustments leave the network throughput variations intact, while ensuring that the ABR algorithm has, on average, adequate bandwidth to stream at the highest quality. The T-Mobile and Verizon LTE traces have high throughput variations (with standard



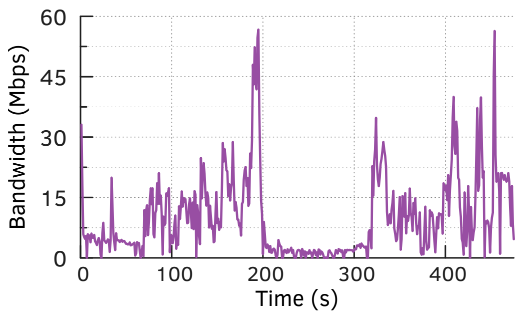
(a) *3G*



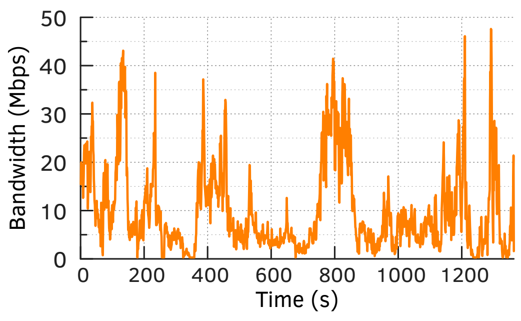
(b) *AT&T*



(c) *FCC*



(d) *T-Mobile*



(e) *Verizon*

Figure 6.5: *Network throughput variations.*

deviations between ≈9 Mbps and ≈10 Mbps), representing the less-than-ideal network conditions under which we intend to evaluate *VOXEL*. The 3G, FCC and AT&T traces have less variations, with standard deviations of 1.1 Mbps, 2.35 Mbps and 2.88 Mbps, respectively.

6.4.6 ABR Algorithms

In our evaluations we compare four ABR algorithms against ABR*. They include BOLA [42] and MPC [41], two state-of-the-art ABR algorithms, and, the more recent, BETA [105] from the literature. We did not modify the ABR algorithms, with the exception of providing BOLA and MPC with the exact segment sizes, instead of average bitrates. [1] The fourth is a naïve throughput-based ABR algorithm (abbreviated as "Tput") to identify what—the transport or the ABR algorithm, or both—contributes the most, in the various experiments, towards improving the streaming performance. We also varied the playback buffer size across a wide range of values from 4 s through 28 s. A new segment download can start only if the buffer is not full. Most academic ABR algorithms use buffers larger than 24 s (e.g., [42, 102, 104]), but small buffers are crucial for supporting low-latency or live-streaming-like applications.

6.4.7 Experiment Considerations

An *experiment*, in our evaluation, involves streaming a video from a server to a client via the router, under a fixed *configuration*. A configuration specifies the ABR algorithm, buffer size, video, and network trace. Unless otherwise stated, we repeat each experiment 30 times and report the aggregate statistics of the metrics gathered. For each repetition we linearly shift the network trace by $d/30$ s, where d is the trace duration in seconds, to investigate the interactions between throughput variations and variations in segment sizes of our VBR-encoded videos (see Fig.6.2). To evaluate the performance of each trial, we instrumented our video streaming software to obtain segment-level timing information with packet-level precision.

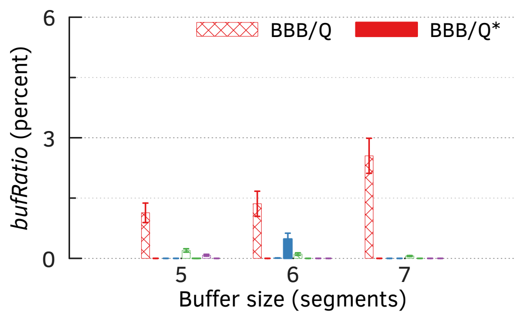
6.5 ABR Algorithms with QUIC*

To evaluate the implications of an incremental deployment and, most importantly, ascertain the need for a cross-layer, coordinated approach for optimizing video streaming, we first check the performance of an unmodified ABR algorithm with QUIC*. An unmodified ABR algorithm will use QUIC* identically to QUIC—using only reliable streams for transport. Hence, we added minimal support for exploiting QUIC* by requesting the I-Frames over reliable streams and all other frames over unreliable streams.

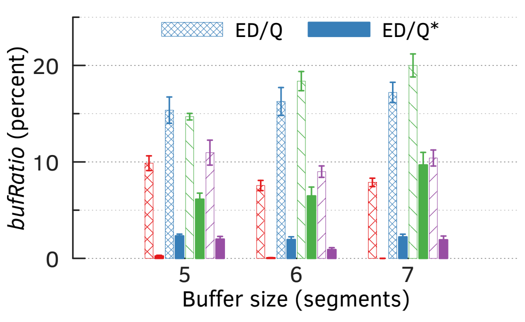
6.5.1 In-lab Trials with Network Traces

In our evaluations with mobile and fixed-line network traces, ABR algorithms encounter less rebuffering when running atop QUIC* (in plots labeled "Q*") than QUIC (labeled "Q"). We define *bufRatio* as the total stall time divided by the video duration during one video playback. The playback buffer sizes used in this evaluation—from 24 s through 32 s,

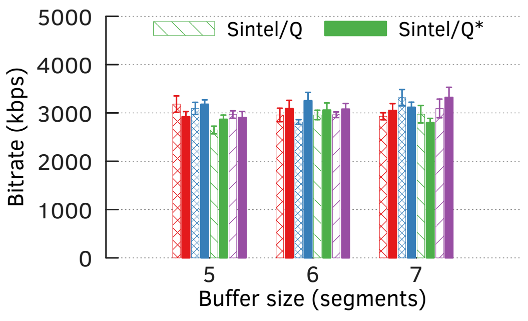
[1] We implemented BETA from scratch, to the best of our ability, based on the details in their paper [105], since it is not publicly available.



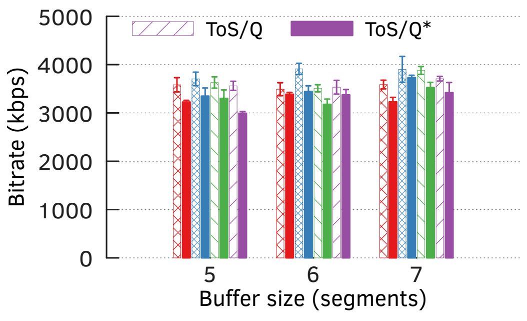
(a) *BOLA; 20 Mbps*



(b) *MPC; 20 Mbps*



(c) *BOLA; 20 Mbps*



(d) *MPC; 20 Mbps*

Figure 6.6: *ABR algorithms with QUIC* in realistic cross-traffic conditions, with a 20 Mbps cross-traffic, offer substantially lower bufRatios (a) and (b) by trading off bitrates (c) and (d).*

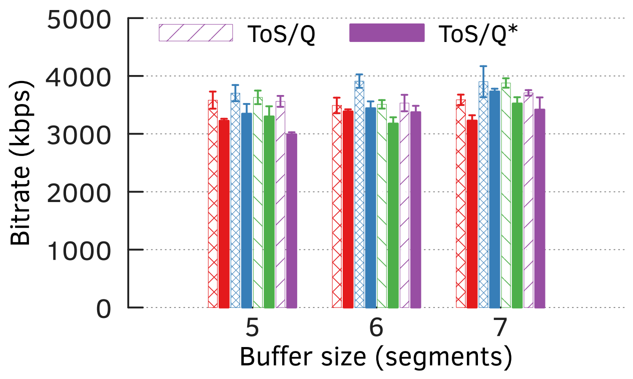
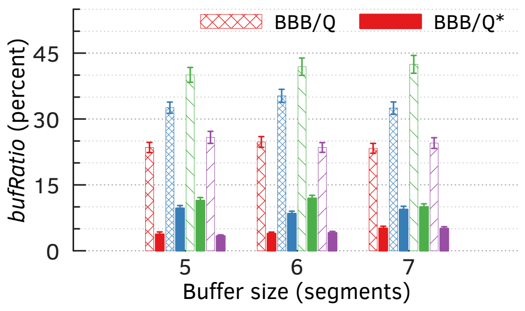
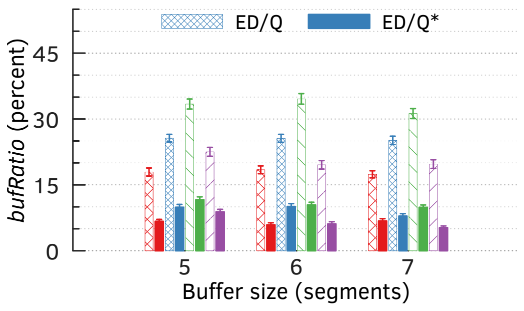


Figure 6.7: *MPC with QUIC* against 20 Mbps cross-traffic experiences slightly lower bitrates.*

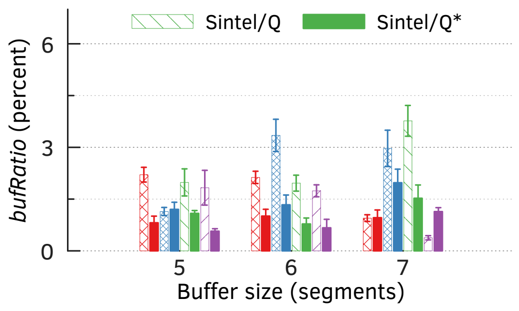
or 5 through 7 segments—are a representative lower bound of prior work [38, 42, 104]. Fig. 6.8 shows the 90th-percentile and standard error of *bufRatio*s for 30 trials of each ABR algorithm over both QUIC and QUIC*, under different network conditions. QUIC* delivers, per this figure, lower *bufRatio* than QUIC for all ABR algorithms; we omit the plots for "Tput" in the interest of space. The results are quite telling for the highly varying T-Mobile (Fig. 6.8a) and Verizon (Fig. 6.8b) traces. The inferences hold for the AT&T, 3G, and FCC traces (omitted to conserve space).



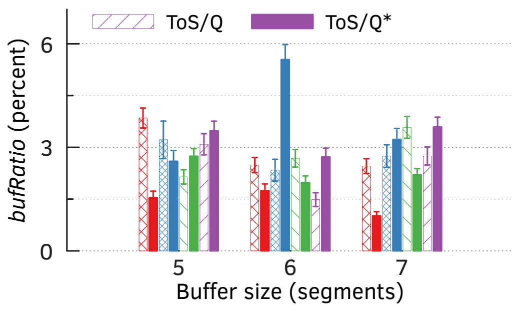
(a) *MPC - T-Mobile*



(b) *MPC - Verizon*



(c) *BOLA - T-Mobile*

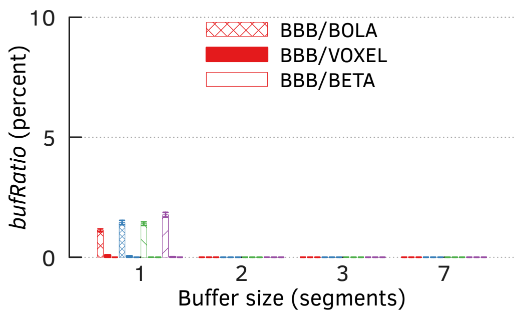


(d) *BOLA - Verizon*

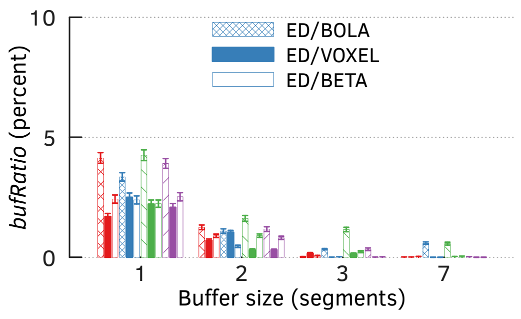
Figure 6.8: *When testing unreliable streams with "vanilla" (i.e., unmodified) ABR algorithms, MPC offers substantial improvements in rebuffering ratios across all videos and buffer sizes. BOLA, in contrast, improves rebuffering ratios in some settings and degrades in some others.*

The 90^{th}-percentile improvements in *bufRatio*s for MPC is on average 71.7% larger than that for BOLA (9.2%), mainly because MPC's network-throughput prediction performs poorly for our traces. To shed light on how QUIC* lowers the *bufRatio*s we estimate the average bitrates across all trials for each video under different configurations. Unmodified ABR algorithms seem to trade off bitrates (Fig. 6.4) for *bufRatio*s; the tradeoffs are particularly conspicuous in case of MPC with −24.7% but also across all configurations. Although unmodified BOLA also experiences lower average bitrates when running over QUIC*, the differences are much smaller (−4.1%) than for MPC. Even in scenarios where BOLA is worse than QUIC* (in Figs. 6.8c & 6.8d), we do not significantly degrade the bitrate.

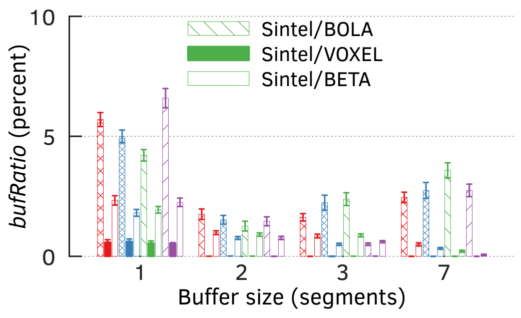
The rare occasions where QUIC*'s performance is not on par with QUIC emphasize the need for a cross-layer optimization, to enable ABR algorithms to fully utilize the underlying partially reliable transport, instead of opaquely sending frames unreliably based on type.



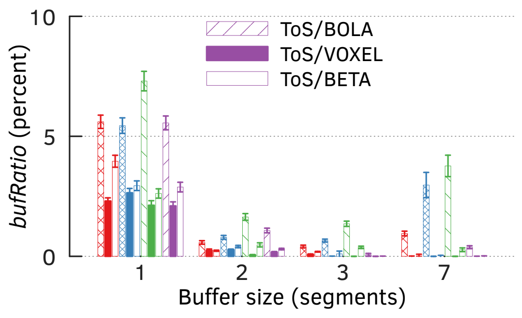
(a) *AT&T*



(b) *3G*



(c) *Verizon*



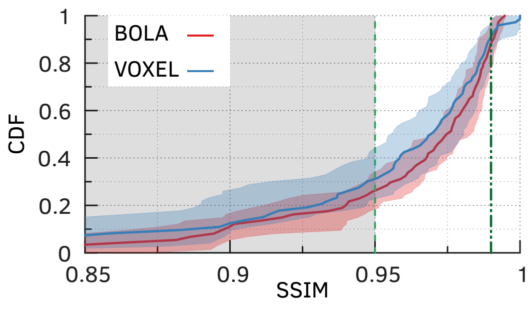
(d) *T-Mobile; VOXEL less aggressive*

Figure 6.9: *bufRatio while streaming with BOLA, BETA, and VOXEL over different networks. VOXEL outperformed BOLA and BETA in practically all scenarios. In T-Mobile (d), VOXEL was too aggressive overall, and we corrected this behavior by tuning a single bandwidth-safety parameter to underestimate slightly the estimated throughput. We show the untuned VOXEL in Fig. 6.19c.*

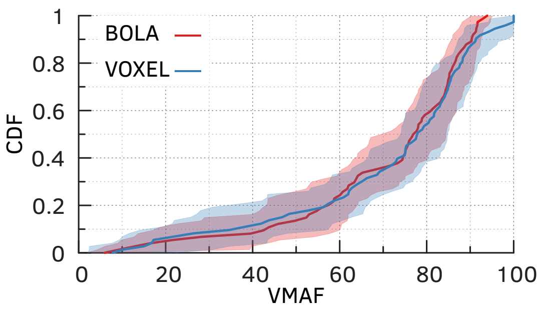
6.5.2 In-lab Trials with Cross Traffic

The trials with network traces cannot capture the dynamic behavior of competing flows in a real network. To test an ABR's performance in the presence of reactive flows, we generate cross-traffic using Harpoon [134] while streaming video. Harpoon is a flow-level traffic generator that generates traffic based on web workloads. It takes a number of clients, C, and servers, S, as input and generates traffic by making the clients fetch files of varying sizes at varying times from the servers. We vary C to generate varying amounts of cross traffic, averaging to $\mathcal{T}$: 10 Mbps, 15 Mbps, and 20 Mbps. The self-similar nature of the cross-traffic does not represent a constant load: Rather, it has many high and low bandwidth regions.

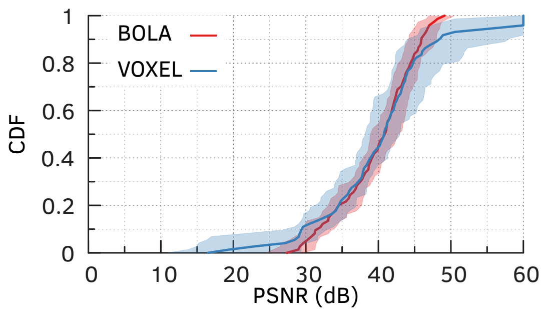
The link capacity in all scenarios was 20 Mbps. We measure, for each value of $\mathcal{T}$, the 90$^{\text{th}}$-percentile of *bufRatio*s and average bitrates across five trials for the three ABRs as before. ABR algorithms using QUIC* experience much less rebuffering than when using QUIC (Fig. 6.6). Even though ABR algorithms experience a slight reduction in average bitrates, the improvements in *bufRatio*s are substantial. MPC, again, shows with 82%



(a) *BBB - Verizon; SSIM - distribution*



(b) *BBB - Verizon; VMAF - distribution*



(c) *BBB - Verizon; PSNR - distribution*

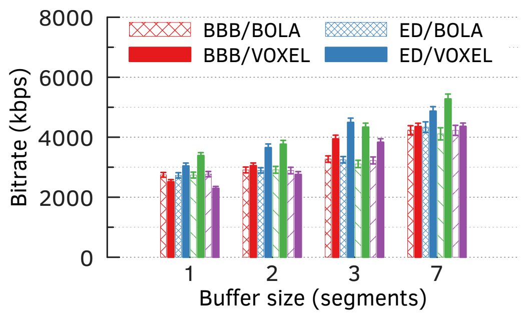
Figure 6.10: *(a) SSIM, (b) VMAF, and (c) PSNR measurements while streaming BBB over Verizon show that VOXEL outperforming BOLA independent of the quality metric is not achieved by trading a lower visual quality. In both metrics VOXEL even achieves better scores in the upper range of each scale.*

more improvements than BOLA (63.6%), but also experiences degradation in average bitrates when using QUIC*.

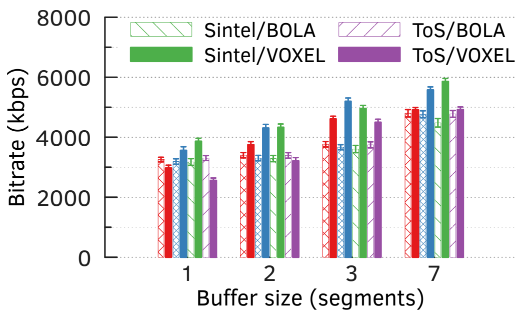
The in-lab trials with network traces and against competing flows show that even with utilizing unreliable streams, a superficial ABR modification, we significantly lower re-buffering under varying network conditions. Without a meticulous redesign, unsurprisingly, ABR performance might suffer under some network conditions.

6.6 ABR* with QUIC* (or *VOXEL*)

To fully exploit QUIC* and optimize video streaming without sacrificing end-users' QoEs, we upgraded BOLA to optimize for visual quality and to exploit frame-drop tolerance (§4). The redesigned ABR algorithm, ABR*, retains it's primary goal of avoiding rebuffering. We evaluate *VOXEL* (ABR* with QUIC*) against a wide variety of network conditions and playback buffer configurations. We also included BETA into our trials, to demonstrate that *VOXEL* outperforms BETA's similar, but limited, feature-set. We do not show MPC in these comparative evaluations due to its poor performance



(a) *T-Mobile*



(b) *Verizon*

Figure 6.11: *VOXEL outperforms BOLA/QUIC even in bitrates when streaming various videos over the (a) T-Mobile and (b) Verizon network.*

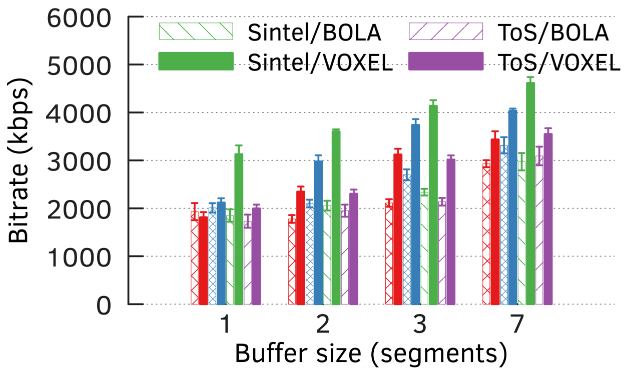


Figure 6.12: *VOXEL outperforms BOLA in average bitrates in the presence of cross-traffic with an average throughput of 20 Mbps.*

against *VOXEL* in our highly varying network traces as well as challenging cross-traffic scenarios.

6.6.1 In-lab Trials with Network Traces

In our evaluations, under all network conditions emulated using the different trace files, *VOXEL* experienced either substantially low or virtually zero rebuffering. We mainly focus, due to space constraints, on the plots for AT&T (Fig. 6.9a), 3G (Fig. 6.9b), Verizon (Fig. 6.9c) and T-Mobile (Fig. 6.9d). We observe similar results for FCC (Fig. 6.20a). *VOXEL* practically eliminates rebuffering against the state-of-the-art, BOLA, and BETA under all buffer sizes ranging from 5 through 7 segments; we only plot the largest buffer size in this range, where the 90th improvement is 100%. If we compare the 7-segment buffers with the smaller sizes, we observe an increase in *bufRatio*. Comparing the *bufRatio* with the average bitrates in Fig. 6.11 reveals the reason: With a large buffer, BOLA aggressively requests higher quality segments but fails to deliver them in

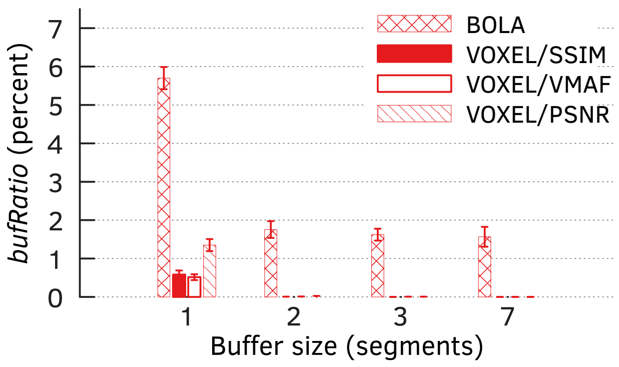


Figure 6.13: *The bufRatios while streaming BBB over Verizon show that VOXEL outperforms BOLA independent of the quality metric, emphasizing its agnostics.*

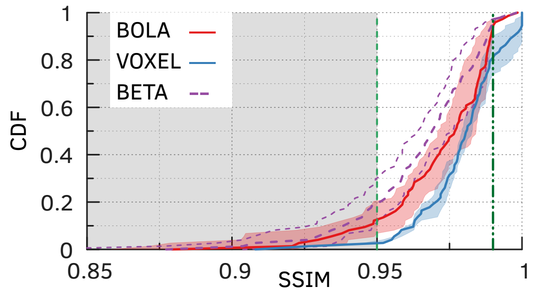
time, resulting in an overall increase in *bufRatio*. These observations show that choosing the right quality level is hard and large buffers cannot always prevent rebuffering.

To demonstrate that *VOXEL* can even perform well in low-latency or live-streaming-like scenarios, we experimented with very small playback buffers. Per Fig. 6.9, even when the playback buffer is as small as 1 segment (along with one "in-flight" segment), *VOXEL* vastly outperforms today's state-of-the-art streaming implementations in *bufRatio* by 74%. The rebuffering experienced when streaming BBB over Verizon, in Fig. 6.13, shows that *VOXEL* outperforms BOLA/QUIC regardless of the choice of the QoE metric, demonstrating that *VOXEL* is QoE-metric-agnostic.

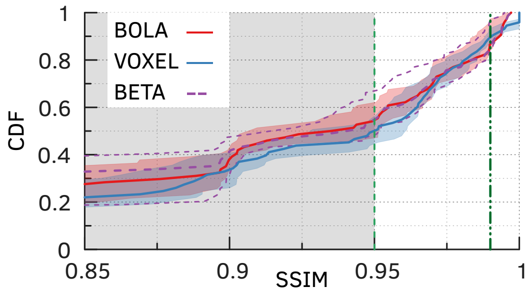
We show the average bitrates (i.e., mean of average bitrates of 30 trials) of the video streams under different network conditions in Fig. 6.11. In addition to streaming the videos with virtually low or no rebuffering *VOXEL* sustains average bitrates that are at least on par and in most cases significantly higher than that of the state-of-the-art. Next, we focus on *VOXEL*'s performance with respect to the SSIM metric.

First, we refer to the Verizon trace experiment, where *VOXEL* reduces rebuffering significantly by 96.3% (Fig. 6.9c). When comparing the CDFs of all streamed segments' SSIMs for both BOLA/QUIC and *VOXEL* (see Fig. 6.10a), we observe that there is little difference in the median SSIMs (solid lines). The rebuffering reduction was, thus, no trade-off for a lower SSIM score. The shaded area in the plot indicates the range between the best and worst SSIMs recorded across all the 30 runs.

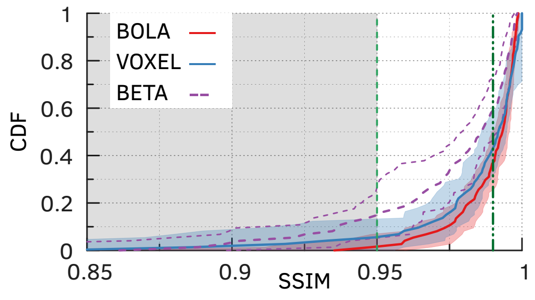
To illustrate *VOXEL*'s QoE-metric agnosticism, we repeated the Verizon experiment, replacing SSIM with VMAF and PSNR. *VOXEL* performs on par with BOLA in the lower VMAF score region, but outperforms BOLA in the upper score region (Fig. 6.10b); *VOXEL* even achieves perfect scores for several segments. The same holds for *bufRatio*, in Fig. 6.13: We show that *VOXEL* almost eliminates rebuffering with either *SSIM*,



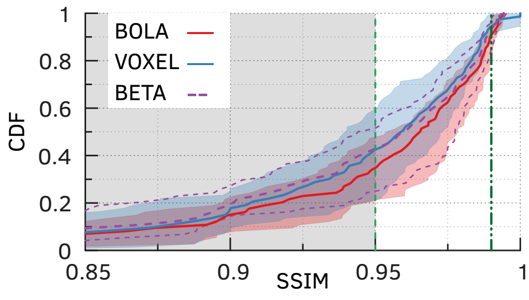
(a) *ToS - AT&T; 2-segment buffer*



(b) *Sintel - 3G*



(c) *ED - Verizon*



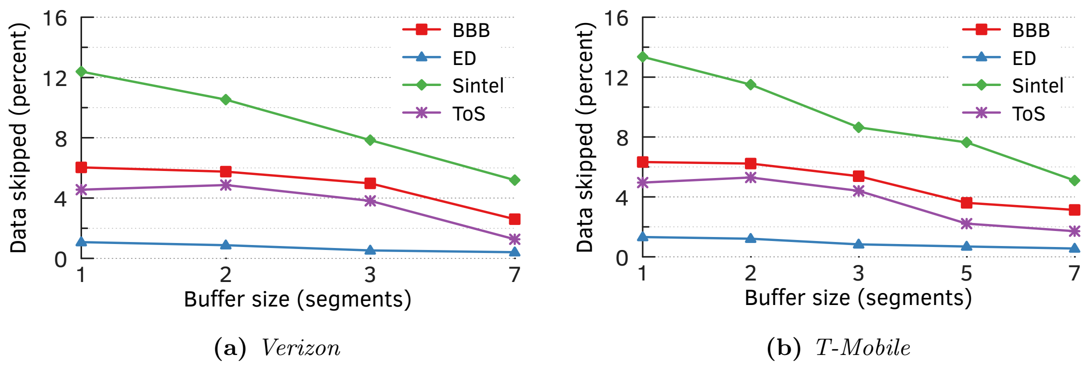
(d) *BBB - T-Mobile; VOXEL less aggressive*

Figure 6.14: *SSIMs of selected videos while streaming with BOLA, BETA, and VOXEL over different networks. VOXEL is superior in SSIM while reducing bufRatio (see Fig. 6.9). In (a), where neither protocol shows rebuffering, VOXEL better utilizes the available bandwidth, (b) performed best in SSIM in with on par or better bufRatio, and traded SSIM only with BOLA in (c) and (d), for a vastly lower bufRatio as both BETA and BOLA.*

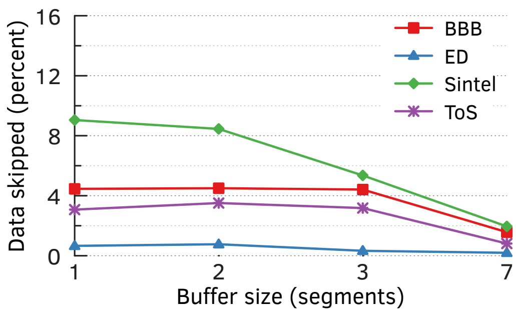
VMAF, or *PSNR*. PSNR, in Fig. 6.10c, shows very similar performance to VMAF and SSIM.

VOXEL achieves low rebuffering by exploiting the virtual quality levels, obtained by "skipping", i.e., not downloading, less important frames. BETA relies, in contrast, on dropping a percentage of unreferenced *B*-Frames (or *b*-Frames) to counter sudden fluctuations in bandwidth [105]. *VOXEL*, thus, vastly increases the decision space by considering *P*-Frames as well as referenced *B*-Frames. Our experiments show that we had to drop frames in 9% of segments on average. In 85% of those cases it was not sufficient to only drop *b*-Frames; we also dropped 46% of all referenced frames, still resulting in *minimal* SSIM degradation.

Fig. 6.15 shows the percent of data dropped, as a function of buffer size. With an increasing playback buffer size, the amount of data dropped is reduced as the large buffer likely absorbs the variations in network conditions and, thus, makes it unnecessary to drop frames. The differences in the data dropped for the same buffer size for different videos stems from the differences in the scaling impact on SSIM scores. Said differently,



(a) *Verizon* **(b)** *T-Mobile*



(c) *AT&T*

Figure 6.15: *The percentage of data dropped or "skipped" by VOXEL when streaming different videos over (a) Verizon, (b) T-Mobile and (c) AT&T as a function of buffer size.*

if the SSIM difference between two bitrate quality levels is small, we will not insert SSIM quality steps (or virtual quality levels) in between.

In the AT&T LTE network trace experiment (Fig. 6.14a), *VOXEL* can even improve SSIMs. When we experimented with a two-segment-long playback buffer, none of the three systems experienced any rebuffering (Fig. 6.9a), while *VOXEL* is still delivering comparable or higher average bitrates (Fig. 6.19b), even with small buffers. Yet, *VOXEL*'s distribution of SSIM scores (Fig. 6.14a) is below (i.e., *better than*) BOLA and BETA, indicating that *VOXEL* used the available bandwidth more efficiently than others. When streaming over the Verizon network, *VOXEL* outperforms BETA and BOLA in *bufRatio* for all videos (refer Fig. 6.9c for the *bufRatio* of all four videos for all playback buffer sizes). BOLA outperforms, however, *VOXEL* and BETA in terms of SSIM (refer to ED in Fig. 6.14c). BETA and *VOXEL*, however, only seemingly lose against BOLA, since this is part of the trade-off which significantly reduces *bufRatio*. For 3G (*bufRatio* in Fig. 6.9b) and bitrates in Fig. 6.19a, and FCC (Figs. 6.20a, 6.20b) *VOXEL* is not able to remove rebuffering entirely, but still delivers comparable or better performance than the state-of-the-art. Overall, our evaluation shows that in most cases *VOXEL* is clearly superior to BOLA as well as BETA.

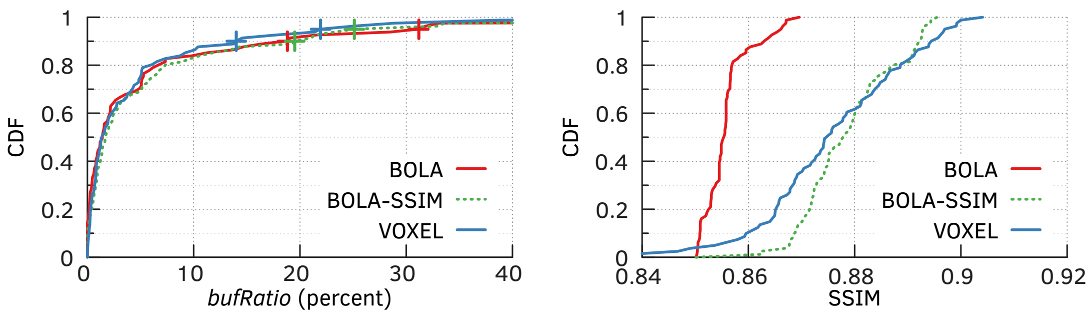
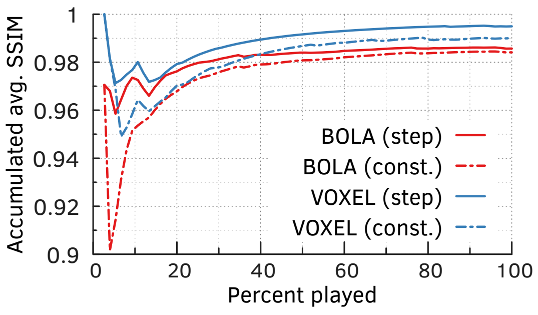


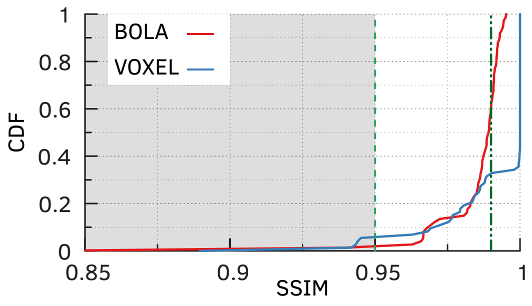
Figure 6.16: *BOLA, BOLA-SSIM and VOXEL have 7.9%, 8.2% and 5.1% mean bufRatio, respectively, when streaming BBB over 86 3G traces with a 1-segment buffer. Pointing out the 90th and 95th, the bufRatio difference mostly lies in the upper percentiles.*

One exception to the above performance of *VOXEL* is the T-Mobile result in Fig. 6.9d. We found, however, that *VOXEL* was tuned too aggressively in its quest to optimize for SSIMs. Such aggressive tuning can lead to suboptimal decisions in network scenarios with large bandwidth fluctuations such as the T-Mobile trace. When we tuned *VOXEL* to be less aggressive, i.e., to slightly underestimate the available throughput, we once again outperform BETA not only in SSIM (Fig. 6.14d) but also in *bufRatio* (Fig. 6.9d). This tuning required changing a single parameter, the *bandwidth-safety* factor, that is applied to the estimated bandwidth. The results for an untuned *VOXEL* can be found in the Appendix in Fig. 6.19d for SSIM and Fig. 6.19c for *bufRatio*. This ability to balance the tradeoff between visual quality and rebuffering can easily be leveraged dynamically by a more advanced ABR algorithm.

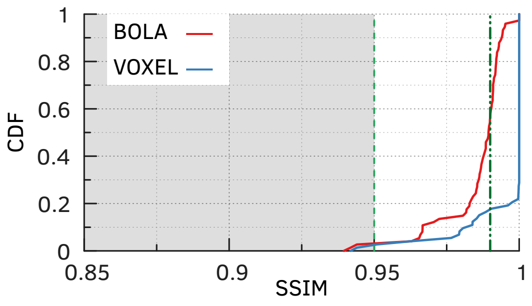
Fig. 6.16 isolates the effect of the two updates to BOLA described in §6. We evaluated BOLA, BOLA-SSIM, and *VOXEL* by streaming BBB over 86 3G traces collected by Riiser et al. [131] using a 1-segment buffer. The low average bandwidth of these 3G traces helps to stress-test the ABR algorithms (Fig. 6.14b). BOLA-SSIM's average SSIM score is 0.02 higher than that of BOLA, but this increase comes at the cost of 4% more rebuffering. *VOXEL* has, however, 35% less rebuffering when compared with BOLA while also enjoying a 0.02 SSIM-score advantage. BOLA-SSIM obtains its SSIM advantage by optimizing for SSIM and by using available bandwidth more aggressively, and with more download options, than BOLA. *VOXEL* reduces rebuffering through smart segment abandonment enabled by QUIC*. Smart abandonment is particularly useful during periods of low bandwidth. We repeated the experiments with a 7-segment buffer and obtained similar results, with *VOXEL* obtaining an even lower *bufRatio*. With the larger buffer, BOLA, BOLA-SSIM, and *VOXEL* have mean *bufRatio*s (SSIM scores) of 7.1% (0.865), 7.1% (0.898), and 2.8% (0.895), respectively.



(a) *BBB; SSIM progression*



(b) *BBB; SSIM distribution*



(c) *BBB; SSIM distribution*

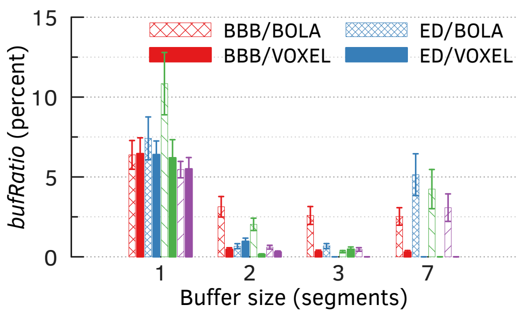
Figure 6.17: *(a) SSIM progression of BOLA and VOXEL while streaming BBB over a constant 10.5 Mbps trace with the CDF of SSIMs in (b) and a step trace that drops from 10.75 Mbps to 10.5 Mbps after 70 s with the CDF of SSIMs in (c).*

6.6.2 In-lab Trials with Long Network Queues

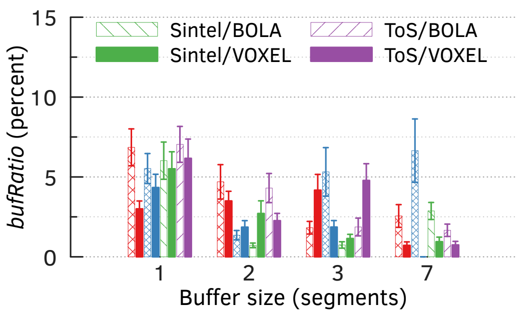
Video content, cached on premise of LTE providers, will traverse an LTE network path with typically very long network queues. To acknowledge this LTE queue behavior for popular on-site-cached content, we ran additional network trace experiments, now with a 750-packets-long queue (see Fig. 6.18). The 1-segment buffer is, again, the largest hurdle but *VOXEL* still has a slight edge over BOLA, even with the more challenging T-Mobile trace. Larger buffer sizes, and the less challenging Verizon trace widen the gap between the two. Looking at larger buffers and Verizon in Fig. 6.18b, we do see *VOXEL* occasionally performing worse than BOLA. This can be attributed to the QUIC version of *VOXEL*, which relies on CUBIC as its congestion control (CC). Large network queues pose a challenge for loss-based CC, thus, in future work, *VOXEL* should be evaluated with a delay based CC.

6.6.3 In-lab Trials without Partial Reliability

To quantify the benefits of partial reliability, we, again, ran trials with network traces, though, now without unreliable streams, i.e., we enforced fully reliable transfers. We

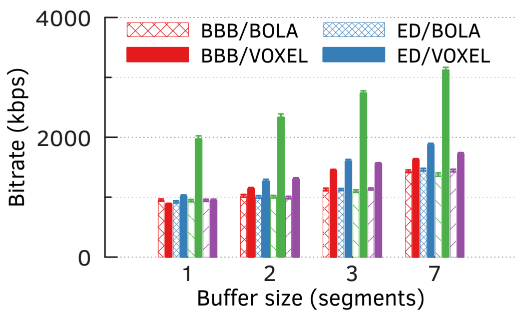


(a) *T-Mobile*

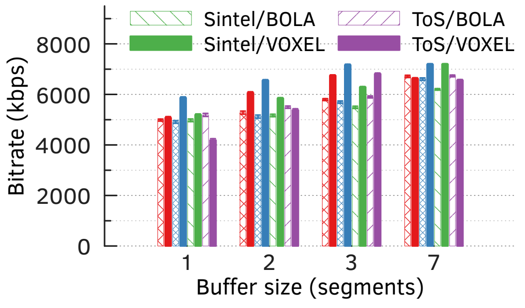


(b) *Verizon*

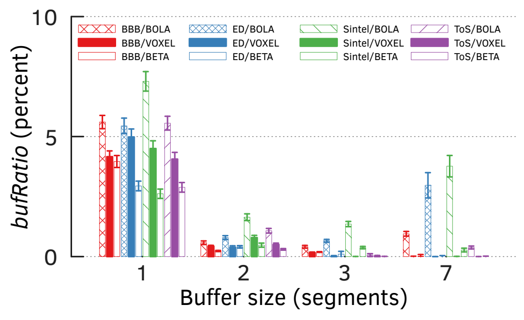
Figure 6.18: *bufRatios for a 750-packets-long network queue.*



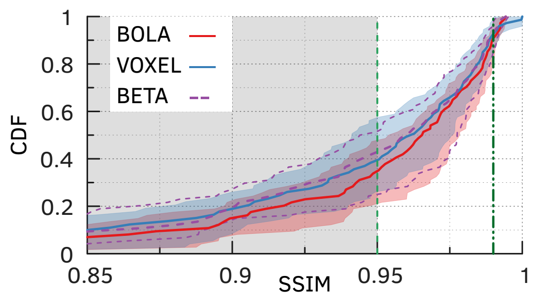
(a) *3G; Average bitrates*



(b) *AT&T; Average bitrates*



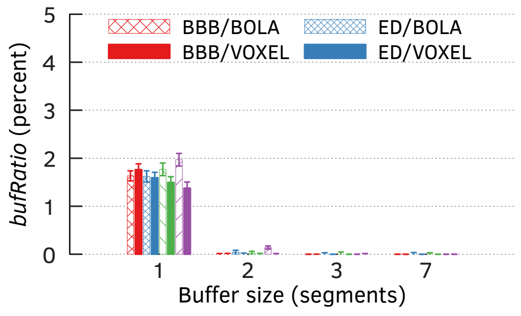
(c) *T-Mobile; VOXEL too aggressive*



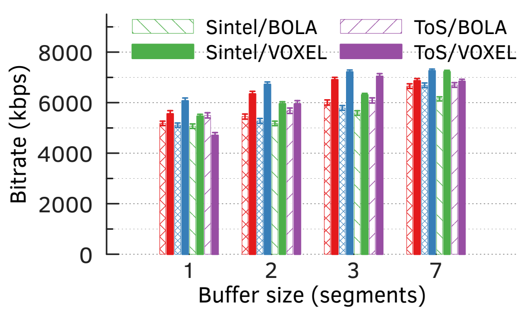
(d) *BBB - T-Mobile; VOXEL too aggressive*

Figure 6.19: *Average bitrates while streaming with VOXEL over (a) 3G and (b) AT&T. A too aggressively tuned VOXEL losing against BETA in bufRatio in (c) while outperforming BETA in SSIM (d).*

kept all remaining features of *VOXEL* intact. With the Verizon trace (Fig. 6.21b) the *bufRatio* doubled without partial reliability across all buffer sizes. Even when streaming under the more challenging T-Mobile trace Fig. 6.21a), *VOXEL* outperforms a reliable streaming system in all cases, except for ED with a 1-segment buffer. In summary, enabling partial reliability significantly reduces the *bufRatio* in all but one cases.

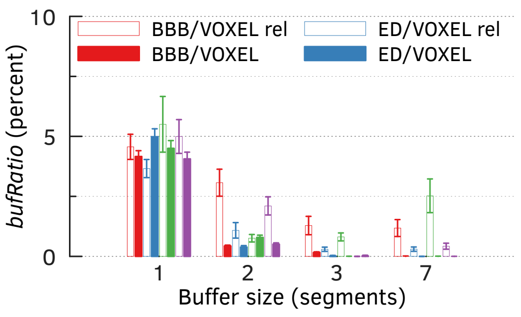


(a) *FCC; bufRatio*

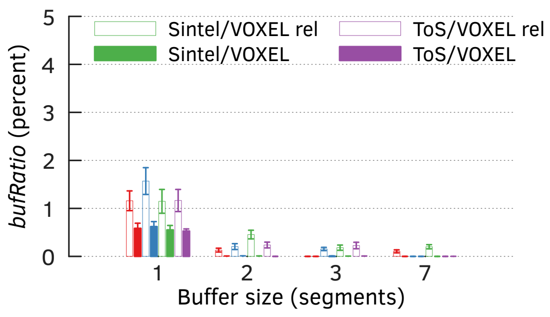


(b) *FCC; Average bitrates*

Figure 6.20: *(a) bufRatio and (b) average bitrates while streaming with VOXEL utilizing the FCC trace.*



(a) *T-Mobile; optional partial reliability*



(b) *Verizon; optional partial reliability*

Figure 6.21: *A bufRatio comparison between VOXEL without partial reliability enabled, denoted as "VOXEL rel", and VOXEL itself, over (a) T-Mobile and (b) Verizon.*

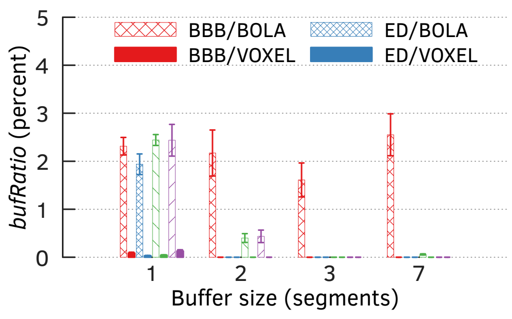
6.6.4 In-lab Trials with Cross Traffic.

VOXEL outperforms the state-of-the-art implementations even in the presence of a substantial volume of cross-traffic. Fig. 6.22a shows that *VOXEL* experiences virtually no rebuffering even in the presence of an average of 20 Mbps of cross-traffic. As all streams in *VOXEL* are congestion-controlled, we have no flow-fairness concerns. We omit results for fairness and lower cross-traffic volumes due to space constraints.

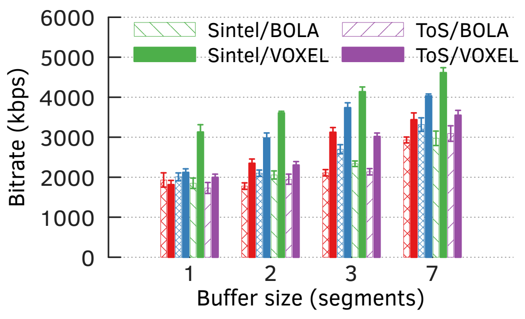
The average bitrates sustained by *VOXEL* in Fig. 6.22b reveal that we do not compromise the bitrates even while virtually getting rid of rebuffering. The performance of *VOXEL*, particularly when using very small playback buffers, attests to the benefit of the cross-layer optimization for improving the status-quo in video streaming.

6.6.5 In-lab Trials with Synthetic Network Traces

To dissect *VOXEL*'s performance improvements, we conducted controlled experiments utilizing synthetic traces. We compared the SSIM progression of streaming BBB on



(a) *bufRatio; 20 Mbps cross-traffic*



(b) *Average bitrates; 20 Mbps cross-traffic*

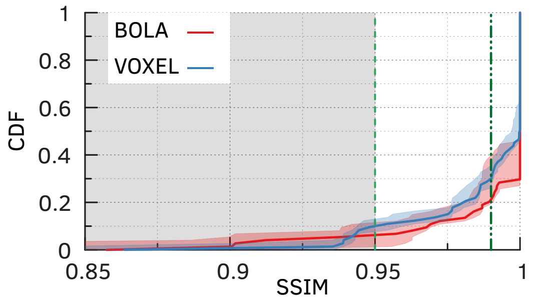
Figure 6.22: *VOXEL outperforms BOLA in (a) bufRatio and (b) bitrate in the presence of 20 Mbps cross-traffic. Even when using a 1-segment playback buffer, VOXEL offers low or near zero rebuffering rates.*

BOLA with *VOXEL* using (a) a constant throughput of 10.5 Mbps and (b) a step trace starting at 10.75 Mbps and dropping to 10.5 Mbps after 70 s (see Fig. 6.17a). To avoid handicapping BOLA, we use a playback buffer of 28 s. In the initial phase, where both ABR algorithms are filling their buffer, *VOXEL*'s SSIM never drops below 0.95 giving it a quality head start compared to BOLA which drops down to 0.90. In steady-state, *VOXEL* outperforms BOLA again with overall higher SSIMs throughout the experiment. Both ABR algorithms run under the same stable conditions, i.e., with constant available throughput, but *VOXEL* utilizes the available resources more efficiently. Fig. 6.17b shows that *VOXEL* obtains an SSIM score of 1.0 for 65% of the segments. BOLA, in contrast, does not get any perfect scores.

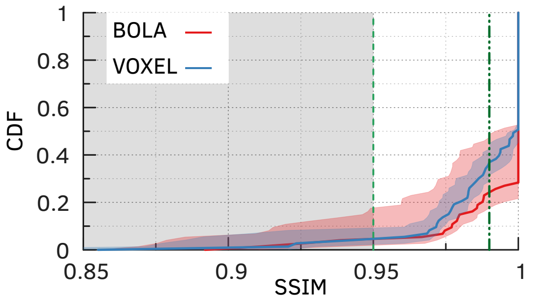
We conduct a second experiment where we start at a marginally higher 10.75 Mbps and, after 70 s, drop down to 10.5 Mbps, the same throughput used for the first experiment. *VOXEL* has greater freedom in selecting a suitable quality and, unsurprisingly, outperforms BOLA again. The finite set of quality levels BOLA can chose from do not capture the network conditions well, and results in a perfect delivery of only 3% of the segments (Fig. 6.17c). In contrast, *VOXEL* copes well with the network conditions, delivering 80% of the segments with a perfect 1.0 SSIM score.

6.6.6 In-the-wild Trials

To complement our in-lab trials, we streamed video, inside of Europe, from a server in a datacenter in France to a client behind a university WiFi in Germany, for verifying the real-world performance of *VOXEL*. The videos were streamend throughout the day, alternating between BOLA and *VOXEL* using the four videos from Tab. 6.1 with a small 1-segment and a large 7-segment playback buffer. While BOLA and *VOXEL* achieve low



(a) *BBB*



(b) *ToS*

Figure 6.23: *In-the-wild trials: SSIM score distribution when streaming with a 1-segment buffer.*

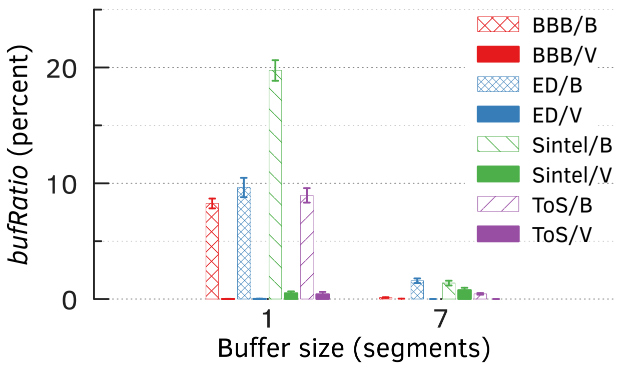


Figure 6.24: *Comparison of bufRatio of BOLA/QUIC, labelled "B", and VOXEL, labelled "V" in in-the-wild trials with VOXEL dominating BOLA in small and outperforming it in large buffer scenarios.*

rebuffering for large buffers, *VOXEL* outperforms BOLA significantly for small buffers (Fig. 6.24). When looking at the average bitrates in Fig. 6.25, we can see that *VOXEL* streams with equal or marginally lower bitrates compared to BOLA. The slight reduction in average bitrate is, though, not at the expense of SSIM. When we compare the SSIM scores in the low-buffer experiment (Fig. 6.23), *VOXEL* performs comparably and, thus, does not unnecessarily trade off rebuffering or average bitrate for visual quality. To ensure similar conditions for both *VOXEL* and BOLA/QUIC, we measured the clients' available bandwidths during the experiment, and they varied, on average, by less than 200 Kbps.

6.7 Real User Survey

We conducted a real user study with 54 participants recruited from different universities showing that *VOXEL* is superior for both objective and subjective metrics. We used one-minute-long video clips extracted from our in-lab experiments; we chose videos streamed

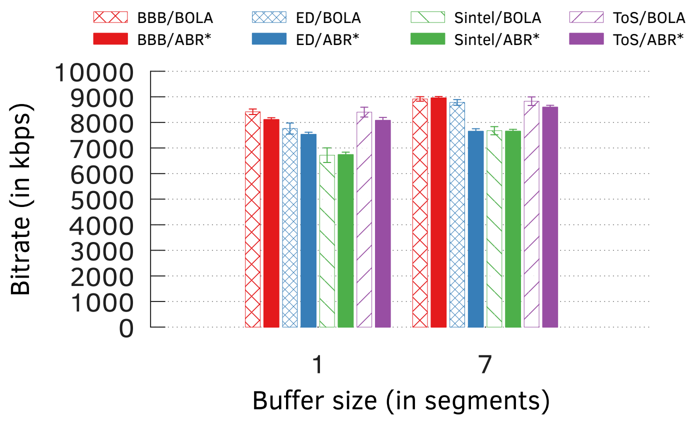


Figure 6.25: *Comparison of the average bitrates of BOLA/QUIC in the in-the-wild trials with VOXEL streaming with equal or marginally lower bitrates.*

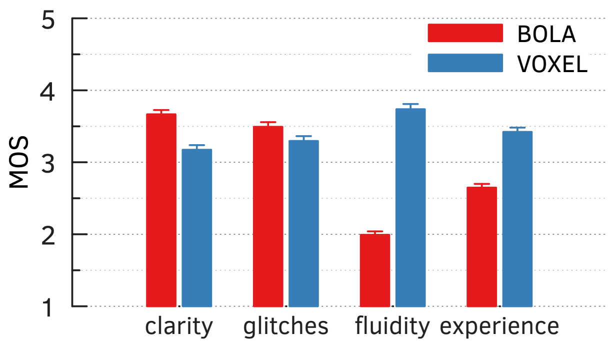


Figure 6.26: *MOS along 4 dimensions: Clarity (i.e., visual quality), glitches (i.e., noticeable artifacts), fluidity (i.e., rebuffering) and overall viewing experience.*

in challenging network conditions (e.g., scenarios where network throughput was as low as 0.3 Mbps). Of the 54 study participants 84% preferred *VOXEL* to BOLA, i.e., they would rather watch the videos streamed using the former than the latter. We asked users if they would have stopped watching the clips: 31% of users indicated that they would have stopped the BOLA streams, if they were permitted, while only 10% said they have stopped the *VOXEL* streams. If the short videos were representative of what to expect in longer videos, 74% indicated that they would not watch the BOLA streams compared to 36.7% for *VOXEL*.

The preference for *VOXEL* is also reflected in the MOS values given for the questions along four dimensions (Fig. 6.26). The playback fluidity was important and rated 1.7 points higher for *VOXEL*. This fluidity is traded for a slightly lower score in terms visual quality, i.e., noted here as clarity −0.49 and glitches −0.19, though, the overall experience was preferred with a 0.77 points higher score.

6.8 Summary

We designed and implemented *VOXEL*, a cross-layer video streaming optimization, and showed that in our evaluations it significantly outperforms the state-of-the-art. *VOXEL*'s design builds upon the insights that (i) videos can tolerate some drops without significant impact on QoE—motivating us to use a partially reliable transport protocol, (ii) we can identify less "important" frames—motivating us to alter the frame sequence in the manifest, (iii) we can leverage QoE utilities—motivating us to introduce virtual quality levels through frame-drops. Skipping a non-trivial amount of streaming data while still delivering high quality video has huge monetary implications for both Content Delivery Networks (CDNs) and content providers—a novel use case for *VOXEL* that we leave to future work.

To make the design and evaluation of *VOXEL* a "call to arms" for the video-streaming community to investigate this optimization landscape, we made our implementation publicly available on GitHub [135].

On being QoE-metric agnostic. *VOXEL* opens up a new design space that is not yet fully supported by current quality metrics. Since SSIM may not be the preferred metric to assess the visual quality of video, we designed *VOXEL* to be QoE-metric agnostic. We show, in Fig. 6.13, relatively good rebuffering performance when using VMAF or PSNR [136, 137]—other widely used QoE metrics in the literature. That being said, we invite the community to consider evaluating new metrics that can accurately measure the QoE impact of imperfect segments and

Managing the overheads. Lastly, *VOXEL*'s frame-prioritization computation introduces some overheads: In our *unoptimized* implementation, enriching the manifest incurred at most 5-times higher cost than that of encoding a video. Beyond simply optimizing the code, the content provider can also drop a few encoding levels and thereby reduce the storage costs. The computation only changes the manifest; video files remain *as is*. Hence, in a typical video streaming scenario via a distributed platform (e.g., CDN), this manifest-only update can be deferred (until later when the provider has empirical observations on streaming conditions) and easily synchronized between servers (owing to their small size). Still, it is a *one-time* computation, i.e., once enriched the manifest can be reused indefinitely.

Chapter 7

Summary and Outlook

In this thesis we studied how to share information between the transport and application layer to improve today's content delivery systems. We started with Socket Intents in combination with MPTCP to allow applications an informed decision about how, i.e., over which combination of network interfaces, its content shall be delivered. We showed performance improvements in 50% of the cases, with our event-based web transfer simulator. Although the Socket Intents approach is rather coarse grained, it is shown to be also suitable for streaming video content [11].

Our video transfer design analysis showed, though, that an even tighter and more fine grained coupling between transport and application is beneficial for reducing rebuffering, the most severe impairment in video streaming. We, thus, created *VOXEL*, our complete and easily deployable video streaming system. With *VOXEL* we can not only determine how a video segment is transferred but on frame granularity decide which frames to transfer, given the current network conditions. Our extensive, and with a real user survey confirmed, evaluation showed significant improvements in avoiding rebuffering.

It follows an outline of how we could apply *VOXEL* to popular and emerging video streaming scenarios, i.e., live streaming and 360° video, as well as what benefits could be gained from integrating a multi-path transport, e.g., MP-QUIC [138], into *VOXEL*.

Live video streaming is more widespread than ever, e.g., Twitch served, on average, over 2 million concurrent viewers in 2021 [139], while still suffering from the same rebuffering issues as VoD streams. Rebuffering makes live streams an ideal target for *VOXEL*. *VOXEL* does, although designed with live-streaming like, low latency streams in mind, not natively support actual live streams without modifications. Currently, the frame analysis is done offline, which is not be possible in a live stream. One approach, to do the analysis in real time, is to train a machine learning model to recognize the connection between the characteristics of a video segment and the resulting visual quality score. In other words, a model is trained, offline, with a diverse set of video segments with known visual qualities, to map the influence of dropped frames to the visual quality. With smart phones already being equipped with machine learning hardware [140], such a trained model could be executed, in real time, while a video segment is generated by the live video encoder. A frame importance order can be generated from the mapping and be attached to the continuously updated manifest. Equipped with this information a *VOXEL* client has all information required to selectively chose which of the frames to download in which order. When there are enough frames received to provide a high QoE for the end-user, the client can even catch up a delayed stream by stopping the current download and moving on.

360° video is an emerging technology to be used with head-mounted-displays head-mounted displays (HMDs). HMDs place displays in front of a users eyes, mounted on a head-gear that tracks the users head movements. Such a system allows a user to look anywhere but it requires the display to present the content equally from all angles the user is looking at. 360° video satisfies this requirement by morphing, or distorting a spherical video onto a two dimensional plane which can then be saved as regular video frames, which can be streamed similar to regular VoD content. As the user can only look at one viewpoint at a time, it is wasteful to transfer content "behind the users back" with the same bandwidth as what is in front of the user. Having different priorities for different regions in the video resulted in the development of tile-based video. A tile-based video splits the content spatially into a grid of smaller videos [141]. Each video can then be transferred at a different quality level, depending on the users' gaze.

Rebuffering, in such a system, is even more detrimental as users will struggle when the immersion breaks because the video suddenly stops. *VOXEL* is ideal in such a case, as we have shown that it can significantly reduce rebuffering. *VOXEL* can prioritize the video tiles even more efficient with the knowledge of where the user is looking at and distribute available throughput accordingly. On top of that, the inherent ability of *VOXEL* to selectively stop a segment download at a designated point, i.e., a defined number of frames is downloaded, is an ideal candidate for 360° video. This designated point can be chosen low for tiles that are behind the user, i.e., downloading a high quality segment but requesting as few frames as possible, but changed mid download to receive more frames if the user is moving their head, making the unimportant tile now very important.

Beyond looking at specific applications for *VOXEL*, it is also of interest to incorporate a Multi-Path capable transport. QUIC itself jump-started the development of *VOXEL*, and with MP-QUIC [138] there is a transport candidate with the same advantages. Introducing a Multi-Path transport could bring *VOXEL*'s split of important and unimportant data to the next level, as now the priority of data could be matched to the quality of a link. For example, the frame header information of a video is crucial to decode an imperfectly transported video segment, but very small in size. Header information could, thus, be transported over a low capacity but low latency link. The large image information of a video segment could then be transferred in parallel over a high capacity link where the latency is less critical.

The described applications and the potential to extend *VOXEL* itself show the versatility of the system and how it could change many aspects of how we transport content. *VOXEL* is designed to prioritize content and separate it into essential and optional parts. Wherever such a distinction of importance is possible, while needing to conserve bandwidth, *VOXEL* would be a prime candidate to be integrated.

Glossary

ABR	Adaptive Bitrate	1
API	Application Programming Interface	35
BBB	Big Buck Bunny	43
BETA	bandwidth-efficient temporal adaptation	40
BOLA	Buffer Occupancy based Lyapunov Algorithm	13
CC	congestion control	83
CDN	Content Delivery Network	1
DASH	Dynamic Adaptive Streaming over HTTP	41
DNS	Domain Name System	9
DOM	Document Object Model	24
DSL	Digital subscriber line	4
EAF	Earliest Arrival First	18
EAF-MPTCP	Earliest Arrival First with MPTCP	18
ED	Elephants Dream	43
FCC	Federal Communications Commission	71
HAR	HTTP Archive	28
HLS	HTTP live streaming	12
HMD	head-mounted display	92

IPC	inter-process communication	19
LTE	Long-Term Evolution	4
MAM	Multi-Access Management	19
MOS	mean opinion score	53
MPC	Model predictive control	13
MPEG	Moving Picture Experts Group	52
MPTCP	Multi-path TCP	7
PR-SCTP	Partial Reliability Stream Control Transmission Protocol	42
PSNR	Peak signal-to-noise ratio	14
QoE	Quality of Experience	2
QUIC	Quick UDP Internet Connections	4
RTT	round-trip time	10
SCTP	Stream Control Transmission Protocol	37
SRTT	smoothed round-trip times	21
SSIM	structural similarity	14
ToS	Tears of Steel	43
TLS	Transport Layer Security	10
VBR	variable bitrate	54
VMAF	Video Multi-Method Assessment Fusion	14
VoD	Video on Demand	43
VOXEL	Video-streaming Optimization Across Enriched Layers	14
VQ	visual quality	42

Bibliography

[1] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. Voxel: Cross-layer optimization for video streaming with imperfect transmission. CoNEXT '21, page 359–374, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450390989. doi: 10.1145/3485983.3494864. URL https://doi.org/10.1145/3485983.3494864.

[2] Bill Gates. Content is king. https://web.archive.org/web/20010126005200/www.microsoft.com/billgates/columns/1996essay/essay960103.asp, March 1996.

[3] DataReportal. Digital 2022: April global statshot report. https://www.statista.com/statistics/617136/digital-population-worldwide/, April 2022.

[4] Cisco. Visual Networking Index: Forecast and Trends, 2017-2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf, February 2019.

[5] Sandvine. The Global Internet Phenomena Report. https://www.sandvine.com/press-releases/sandvine-releases-2019-global-internet-phenomena-report, September 2019.

[6] Saksena, Rachit and Lu, Dave and Celik, Ilyas. Ericsson Mobility Report. https://www.ericsson.com/4ad7e9/assets/local/reports-papers/mobility-report/documents/2021/ericsson-mobility-report-november-2021.pdf, November 2021.

[7] Presse- und Informationsamt der Bundesregierung. So will die Bundesregierung Funklöcher schließen. https://www.bundesregierung.de/breg-de/aktuelles/funkloecher-und-5g-1841896, January 2021.

[8] Bob O'Donnell. Real-World 5G Speeds Are Slower Than Expected. https://www.forbes.com/sites/bobodonnell/2019/11/22/real-world-5g-speeds/, November 2019.

[9] Morris, Sam. Are Your Country's Cellphone Plans a Rip-off? https://themarkup.org/2020/09/03/cost-speed-of-mobile-data-by-country, September 2020.

[10] Philipp S. Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. Socket intents: Leveraging application awareness for multi-access connectivity.

CoNEXT '13, page 295–300, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321013. doi: 10.1145/2535372.2535405. URL https://doi.org/10.1145/2535372.2535405.

[11] Theresa Enghardt, Thomas Zinner, and Anja Feldmann. Using informed access network selection to improve http adaptive streaming performance. MMSys '20, page 126–140, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368452. doi: 10.1145/3339825.3391865. URL https://doi.org/10.1145/3339825.3391865.

[12] Ross Benes. Buffering Is the Streaming Snafu that Won't Go Away. https://www.emarketer.com/content/buffering-is-the-streaming-snafu-that-won-t-go-away, March 2018.

[13] Deepti Ghadiyaram, Alan C Bovik, Hojatollah Yeganeh, Roman Kordasiewicz, and Michael Gallant. Study of the effects of stalling events on the quality of experience of mobile streaming videos. In *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*. IEEE, 2014.

[14] T. De Pessemier, K. De Moor, W. Joseph, L. De Marez, and L. Martens. Quantifying the Influence of Rebuffering Interruptions on the User's Quality of Experience During Mobile Video Watching. *IEEE Transactions on Broadcasting*, 59(1), March 2013. ISSN 0018-9316. doi: 10.1109/TBC.2012.2220231.

[15] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *2012 Fourth International Workshop on Quality of Multimedia Experience*, 2012.

[16] Akamai Technologies Inc. Bit Rate and Business Model - The science behind how our bodies react to video quality. https://www.akamai.com/us/en/multimedia/documents/white-paper/bit-rate-and-business-model.pdf, 2017.

[17] D. Ghadiyaram, A. C. Bovik, H. Yeganeh, R. Kordasiewicz, and M. Gallant. Study of the effects of stalling events on the quality of experience of mobile streaming videos. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2014.

[18] Sandvine. The Mobile Internet Phenomena Report. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2021/Phenomena/MIPR%20Q1%202021%2020210510.pdf, 2021.

[19] Sunghwan Ihm and Vivek S Pai. Towards understanding modern web traffic. In *ACM IMC*, pages 295–312. ACM, 2011.

[20] Michael Butkiewicz, Harsha V Madhyastha, and Vyas Sekar. Understanding website complexity: measurements, metrics, and implications. In *ACM IMC*, pages 313–328. ACM, 11 2011. doi: 10.1145/2068816.2068846.

[21] Derek J. de Solla Price. *Little science, big science.* Columbia Univ. Press, New York, 1963. ISBN 9780231085625.

[22] Philipp S. Tiesel, Theresa Enghardt, Mirko Palmer, and Anja Feldmann. Socket intents: Os support for using multiple access networks and its benefits for web browsing, 2018. URL https://arxiv.org/abs/1804.08484.

[23] Philipp S. Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. Socket intents: Leveraging application awareness for multi-access connectivity. In *ACM CoNEXT*, pages 295–300. ACM, 2013. ISBN 978-1-4503-2101-3. doi: 10.1145/2535372.2535405. URL http://doi.acm.org/10.1145/2535372.2535405.

[24] Philipp Tiesel, Theresa Enghardt, and Anja Feldmann. Socket intents. Internet-Draft draft-tiesel-taps-socketintents-01, IETF Secretariat, 10 2017. URL https://www.ietf.org/archive/id/draft-tiesel-taps-socketintents-01.txt.

[25] Philipp S. Tiesel, Bernd May, and Anja Feldmann. Multi-homed on a single link: Using multiple ipv6 access networks. In *Proceedings of the 2016 Applied Networking Research Workshop*, ANRW '16, pages 16–18. ACM, 2016. ISBN 978-1-4503-4443-2. doi: 10.1145/2959424.2959434. URL http://doi.acm.org/10.1145/2959424.2959434.

[26] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *SIGCOMM HotNets*, pages 6:1–6:6. ACM, 2010. doi: 10.1145/1868447.1868453.

[27] Philipp Richter, Nikolaos Chatzis, Georgios Smaragdakis, Anja Feldmann, and Walter Willinger. Distilling the internet's application mix from packet-sampled traffic. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement*, volume 8995 of *Lecture Notes in Computer Science*, pages 179–192. Springer International Publishing, 2015. doi: 10.1007/978-3-319-15509-8_14.

[28] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), Jan 2013. URL http://www.ietf.org/rfc/rfc6824.txt.

[29] Costin Raiciu, Christoph Paasch, Sébastien Barré, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be?

Designing and Implementing a Deployable Multipath TCP. In *USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA)*, 2012.

[30] Luca Boccassi, Marwan M. Fayed, and Mahesh K. Marina. Binder: A System to Aggregate Multiple Internet Gateways in Community Networks. In *Proceedings of the 2013 ACM MobiCom Workshop on Lowest Cost Denominator Networking for Universal Access*, LCDNet '13, pages 3–8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2365-9. doi: 10.1145/2502880.2502894. URL http://doi.acm.org/10.1145/2502880.2502894.

[31] Linux Foundation. *Netlink(7) Linux Programmer's Manual.* URL http://man7.org/linux/man-pages/man7/netlink.7.html.

[32] Mirko Palmer. Implementation and evaluation of multi-access policies for MPTCP path management in user-space. Master's thesis, TU Berlin, 2015.

[33] Iraj Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4), October 2011.

[34] Roger Pantos and William May. HTTP Live Streaming. RFC 8216, August 2017.

[35] Bitmovin. Bitmovin Video Developer Report 2019. https://bitmovin.com/bitmovin-2019-video-developer-report-av1-codec-ai-machine-learning-low-latency/, 2019.

[36] encoding.com. Global Media Format Report 2018. https://www.encoding.com/files/2018-Global-Media-Formats-Report.pdf, 2018.

[37] Ben Juurlink, Mauricio Alvarez-Mesa, Chi Ching Chi, Arnaldo Azevedo, Cor Meenderinck, and Alex Ramirez. *Understanding the Application: An Overview of the H.264 Standard.* SpringerBriefs in Computer Science, 2012.

[38] 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), April 2014.

[39] A. Beben, P. Wiśniewski, J. Mongay 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, 2016.

[40] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of ACM SICOMM '14*, 2014.

[41] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of ACM SIGCOMM '15*, 2015.

[42] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. BOLA: near-optimal bitrate adaptation for online videos. *IEEE/ACM Transactions on Networking*, 28 (4), 2020.

[43] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From qos to qoe: A tutorial on video quality assessment. *IEEE Communications Surveys Tutorials*, 17(2):1126–1165, 2015. doi: 10.1109/COMST.2014.2363139.

[44] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.*, 13 (4), April 2004.

[45] ITU. P.1203 : Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport. https://www.itu.int/rec/T-REC-P.1203, October 2017.

[46] Netflix. Toward A Practical Perceptual Video Quality Metric. https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652, June 2016.

[47] ITU. per-second audiovisual quality scores always 5.0 even with corrupted video file. https://github.com/itu-p1203/itu-p1203/issues/5, May 2018.

[48] IHS Markit. How to delight consumers in a connected world? https://gpc.ifa-berlin.com/media/ifagpc/ifagpc_dl_en/ifagpc_dl_en_powerbriefings/power_briefing_2019/IHSM_IFA_GPC.pdf, April 2019.

[49] YouTube. The best experience on YouTube Signature Devices. https://devicereport.youtube.com, April 2018.

[50] YouTube. Video resolution & aspect ratios. https://support.google.com/youtube/answer/6375112, August 2017.

[51] Linux Foundation. Netlink. http://www.linuxfoundation.org/collaborate/workgroups/networking/netlink, November 2009. Accessed: 10. Jan. 2015.

[52] S. Egger, T. Hossfeld, R. Schatz, and M. Fiedler. Waiting times in quality of experience for web based services. In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pages 86–96. IEEE, July 2012. doi: 10.1109/QoMEX.2012.6263888.

[53] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *USENIX NSDI*, volume 17, pages 545–559. Usenix, 2017.

[54] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. Increasing TCP's Initial Window. RFC 6928 (Experimental), Apr 2013. URL http://www.ietf.org/rfc/rfc6928.txt.

[55] Jörg Wallerich, Holger Dreger, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. A methodology for studying persistency aspects of internet flows. *ACM CCR*, pages 23–36, 2005. doi: 10.1145/1064413.1064417.

[56] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *USENIX NSDI*. Usenix, Mar 2016.

[57] Srikanth Sundaresan, Walter De Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: a view from the gateway. In *ACM CCR*, volume 41, pages 134–145. ACM, 2011.

[58] Enric Pujol, Philipp Richter, Balakrishnan Chandrasekaran, Georgios Smaragdakis, Anja Feldmann, Bruce MacDowell Maggs, and Keung-Chi Ng. Back-office web traffic on the internet. In *ACM IMC*, pages 257–270. ACM, 2014.

[59] Junaid Qadir, Anwaar Ali, Kok-Lim Alvin Yau, Arjuna Sathiaseelan, and Jon Crowcroft. Exploiting the power of multiplicity: a holistic survey of network-layer multipath. *CoRR*, abs/1502.02111, 2015. URL http://arxiv.org/abs/1502.02111.

[60] M. Wasserman and P. Seite. Current Practices for Multiple-Interface Hosts. RFC 6419 (Informational), Nov 2011. URL http://www.ietf.org/rfc/rfc6419.txt.

[61] E. Nordmark, S. Chakrabarti, and J. Laganier. IPv6 Socket API for Source Address Selection. RFC 5014 (Informational), Sep 2007. URL http://www.ietf.org/rfc/rfc5014.txt.

[62] D. Thaler, R. Draves, A. Matsumoto, and T. Chown. Default Address Selection for Internet Protocol Version 6 (IPv6). RFC 6724 (Proposed Standard), Sep 2012. URL http://www.ietf.org/rfc/rfc6724.txt.

[63] Robert Kiefer, Erik Nordström, and Michael J Freedman. From feast to famine: managing mobile network resources across environments and preferences. In *Proceedings of the 2014 International Conference on Timely Results in Operating Systems*, pages 7–7. Usenix, 2014.

[64] H. Abbasi, C. Poellabauer, K. Schwan, G. Losik, and Richard. A quality-of-service enhanced socket api in gnu/linux. In *Real-Time Linux Workshop*, 2002.

[65] B. D. Higgins, A. Reda, T. Alperovich, J. Flinn, T. J. Gi uli, B. Noble, and D. Watson. Intentional networking: opportunistic exploitation of mobile network diversity. In *ACM MobiCom*, pages 73–84. ACM, 2010.

[66] Shuo Deng, Anirudh Sivaraman, and Hari Balakrishnan. All your network are belong to us: A transport framework for mobile network selection. In *ACM HotMobile*, pages 19:1–19:6. ACM, 2014. doi: 10.1145/2565585.2565588.

[67] M. Welzl, S. Jorer, and S. Gjessing. Towards a protocol-independent internet transport api. In *ICC*, pages 1 –6, 2011.

[68] Brian Trammell, Colin Perkins, and Mirja Kühlewind. Post sockets: Towards an evolvable network transport interface. In *Workshop on Future of Internet Transport (FIT 2017)*, 2017.

[69] Adnan Aijaz, Hamid Aghvami, and Mojdeh Amani. A survey on mobile data offloading: technical and business perspectives. *Wireless Communications, IEEE Transactions on*, 20(2):104–112, 2013.

[70] Kyunghan Lee, Joohyun Lee, Yung Yi, Injong Rhee, and Song Chong. Mobile data offloading: how much can wifi deliver? In *ACM CoNEXT*, 2010.

[71] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. Augmenting mobile 3g using wifi. In *ACM MobiSys*, pages 209–222, 2010.

[72] Narseo Vallina-Rodriguez, Vijay Erramilli, Yan Grunenberger, Laszlo Gyarmati, Nikolaos Laoutaris, Rade Stanojevic, and Konstantina Papagiannaki. When david helps goliath: the case for 3g onloading. In *SIGCOMM HotNets*, pages 85–90. ACM, 2012.

[73] Joohyun Lee, Yung Yi, Song Chong, and Youngmi Jin. Economics of wifi offloading: Trading delay for cellular capacity. *Wireless Communications, IEEE Transactions on*, 13(3):1540–1554, 2014.

[74] Cisco Systems, Inc. Architecture for mobile data offload over wi-fi access networks (whitepaper), 2012. URL http://www.cisco.com/en/US/solutions/collateral/ns341/ns524/ns673/white_paper_c11-701018.html.

[75] Philipp S. Schmidt, Ruben Merz, and Anja Feldmann. A first look at multi-access connectivity for mobile networking. In *ACM workshop on Capacity sharing*, pages 9–14. ACM, 2012. doi: 10.1145/2413219.2413224.

[76] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. A measurement-based study of multipath tcp performance over wireless networks. In *ACM IMC*, pages 455–468. ACM, 2013.

[77] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? Designing and implementing a deployable multipath TCP. In *USENIX NSDI*, pages 29–29. Usenix, 2012.

[78] Shuo Deng, Ravi Netravali, Anirudh Sivaraman, and Hari Balakrishnan. Wifi, lte, or both?: measuring multi-homed wireless internet performance. In *ACM IMC*, pages 181–194. ACM, 2014.

[79] Bo Han, Feng Qian, Shuai Hao, Lusheng Ji, and NJ Bedminster. An anatomy of mobile web performance over multipath tcp. In *ACM CoNEXT*, 2015.

[80] Ashkan Nikravesh, Yihua Guo, Feng Qian, Z. Morley Mao, and Subhabrata Sen. An in-depth understanding of multipath tcp on mobile devices: Measurement and system design. In *ACM MobiCom*. ACM, 2016.

[81] Lance Hartung and M Milind. Policy driven multi-band spectrum aggregation for ultra-broadband wireless networks. In *Dynamic Spectrum Access Networks (DySPAN)*, pages 82–93. IEEE, 2015.

[82] Thomas Dreibholz, Robin Seggelmann, Michael Tüxen, and Erwin Paul Rathgeb. Transmission scheduling optimizations for concurrent multipath transfer. In *Proceedings of the 8th International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, volume 8, 2010.

[83] Akamai Technologies. Hotstar And Akamai Set Global Streaming Record During VIVO IPL 2018. https://www.akamai.com/uk/en/about/news/press/2018-press/hotstar-and-akamai-set-global-streaming-record-during-vivo-ipl-2018.jsp, April 2018.

[84] YouTube. Mission complete: Red Bull Stratos lands safely back on Earth. https://youtube.googleblog.com/2012/10/mission-complete-red-bull-stratos-lands.html, October 2012.

[85] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of ACM SIGCOMM '11*, 2011.

[86] Stefan Lederer. Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length. https://bitmovin.com/mpeg-dash-hls-segment-length/, April 2015.

[87] Sam Liang and David Cheriton. TCP-RTM: Using RTP for Real Time Multimedia Applications. "http://gregorio.stanford.edu/sliang/rtm.pdf", 2002.

[88] Ashvin Goel, Charles Krasic, and Jonathan Walpole. Low-latency Adaptive Streaming over TCP. *ACM Trans. Multimedia Comput. Commun. Appl.*, September 2008.

[89] Eli Brosh, Salman Abdul Baset, Dan Rubenstein, and Henning Schulzrinne. The Delay-friendliness of TCP. In *Proceedings of SIGMETRICS '08*, 2008.

[90] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. Netw.*, 14(6), December 2006.

[91] B. Mukherjee and T. Brecht. Time-lined TCP for the TCP-friendly delivery of streaming media. In *Proceedings 2000 International Conference on Network Protocols*, 2000.

[92] M. Claeys, N. Bouten, D. De Vleeschauwer, K. De Schepper, W. Van Leekwijck, S. Latré, and F. De Turck. Deadline-aware TCP congestion control for video streaming services. In *2016 12th International Conference on Network and Service Management (CNSM)*, Oct 2016.

[93] Stephen McQuistin, Colin Perkins, and Marwan Fayed. TCP Goes to Hollywood. In *Proceedings of NOSSDAV '16*, 2016.

[94] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. Leith, and M. Medard. Network Coded TCP (CTCP). *ArXiv e-prints*, December 2012.

[95] Huahui Wu, Mark Claypool, and Robert Kinicki. Adjusting Forward Error Correction with Temporal Scaling for TCP-friendly Streaming MPEG. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(4), November 2005.

[96] Nick Feamster and Hari Balakrishnan. Packet Loss Recovery for Streaming Video. In *12th International Packet Video Workshop*, April 2002.

[97] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-draft, IETF, May 2018.

[98] Igor Lubashev. Partially Reliable Message Streams for QUIC. Internet-Draft draft-lubashev-quic-partial-reliability-03, Internet Engineering Task Force, May 2018.

[99] Tommy Pauly, Eric Kinnear, and David Schinazi. An Unreliable Datagram Extension to QUIC. Internet-Draft draft-ietf-quic-datagram-06, Internet Engineering Task Force, October 2021.

[100] Philipp S. Tiesel, Mirko Palmer, Balakrishnan Chandrasekaran, Anja Feldmann, and Joerg Ott. Considerations for Unreliable Streams in QUIC. Internet-draft,

IETF, October 2017. URL https://datatracker.ietf.org/doc/html/draft-tiesel-quic-unreliable-streams-01.

[101] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, 2012.

[102] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proceedings of CoNEXT '12*, 2012.

[103] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of ACM SIGCOMM '16*, 2016.

[104] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of ACM SIGCOMM '17*, 2017.

[105] Cyriac James, Mea Wang, and Emir Halepovic. Beta: Bandwidth-efficient temporal adaptation for video streaming over reliable transports. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, 2019.

[106] Cloudflare. The QUICening. https://blog.cloudflare.com/the-quicening/, September 2018.

[107] Akamai Technologies. Community Blog, FAQ: QUIC Native Platform Support for Media Delivery Products. https://community.akamai.com/customers/s/article/FAQ-QUIC-Native-Platform-Support-for-Media-Delivery-Products?language=en_US, March 2018.

[108] Cloudflare. HTTP/3: the past, the present, and the future. https://blog.cloudflare.com/http3-the-past-present-and-future/, September 2019.

[109] LiteSpeed Technologies. What is HTTP/3 Check? https://http3check.net/about, 2020.

[110] Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, and Anja Feldmann. The QUIC Fix for Optimal Video Streaming. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, EPIQ'18, 2018.

[111] Divyashri Bhat, Amr Rizk, and Michael Zink. Not So QUIC: A Performance Study of DASH over QUIC. In *Proceedings of NOSSDAV '17*, 2017.

[112] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon. HTTP/2-based Frame Discarding for Low-Latency Adaptive Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(1), February 2019.

[113] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758 (Proposed Standard), May 2004. URL http://www.ietf.org/rfc/rfc3758.txt.

[114] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proceedings of USENIX NSDI '18*, 2018.

[115] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural Adaptive Content-Aware Internet Video Delivery. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, 2018.

[116] A. C. Brooks, Xiaonan Zhao, and T. N. Pappas. Structural similarity quality metrics in a coding context: Exploring the space of realistic distortions. *Trans. Img. Proc.*, August 2008.

[117] Piotr Romaniak and Lucjan Janowski. How to build an objective model for packet loss effect on high definition content based on ssim and subjective experiments. In Sherali Zeadally, Eduardo Cerqueira, Marília Curado, and Mikołaj Leszczuk, editors, *Future Multimedia Networking*. Springer Berlin Heidelberg, 2010.

[118] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hossfeld. Impact of frame rate and resolution on objective qoe metrics. In *Proc. Second Int. Workshop Quality of Multimedia Experience (QoMEX)*, pages 29–34, June 2010. doi: 10.1109/QOMEX.2010.5518277.

[119] MPEG. Media presentation description and segment formats. https://mpeg.chiariglione.org/standards/mpeg-dash/media-presentation-description-and-segment-formats, March 2011.

[120] Akamai Technologies Inc. Guidelines for Implementation: DASH-IF Interoperability Points. https://dashif.org/docs/DASH-IF-IOP-v4.2-clean.htm, 04 2018.

[121] Limelight Networks. The State of Online Video 2020. https://www.limelight.com/resources/market-research/state-of-online-video-2020, August 2020.

[122] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: Improving bitrate adaptation in the DASH reference player. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(2s), July 2019.

[123] Dash Industry Forum. A reference client implementation for the playback of MPEG DASH. https://github.com/Dash-Industry-Forum/dash.js, June 2021.

[124] Christian Kreuzberger, Daniel Posch, and Hermann Hellwagner. A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP. In *Proceedings of the 6th ACM Multimedia Systems Conference*, MMSys '15, 2015.

[125] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*, Sept 2014.

[126] Stefan Lederer, Christopher Müller, and Christian Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12. ACM, 2012.

[127] Yanyuan Qin, Shuai Hao, K. R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. ABR Streaming of VBR-encoded Videos: Characterization, Challenges, and Solutions. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, 2018.

[128] youtube-dl. Download videos from YouTube. https://ytdl-org.github.io/youtube-dl/about.html, 2020.

[129] Netflix. Per-Title Encode Optimization. https://medium.com/netflix-techblog/per-title-encode-optimization-7e99442b62a2, December 2015.

[130] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX, 2013.

[131] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: Analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, 2013.

[132] Federal Communications Commission. Raw Data - Measuring Broadband America. https://www.fcc.gov/reports-research/reports/measuring-broadband-america/raw-data-measuring-broadband-america-2016, December 2016.

[133] Cisco. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017-2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html, February 2019.

[134] Joel Sommers, Hyungsuk Kim, and Paul Barford. Harpoon: A Flow-level Traffic Generator for Router and Network Tests. In *Proceedings of SIGMETRICS '04/Performance '04*, 2004.

[135] Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. VOXEL-enabled server and client implementation. https://github.com/derbroti/VOXEL, 2021.

[136] Quan Huynh-Thu and Mohammed Ghanbari. The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommunication Systems*, 06 2012.

[137] Q. Huynh-Thu and M. Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics Letters*, 44(13), 2008.

[138] Quentin De Coninck and Olivier Bonaventure. Multipath quic: Design and evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 160–166, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354226. doi: 10.1145/3143361.3143370. URL https://doi.org/10.1145/3143361.3143370.

[139] Iqbal, Mansoor. Twitch Revenue and Usage Statistics (2022). https://www.businessofapps.com/data/twitch-statistics/, May 2022.

[140] Apple Inc. Machine Learning. https://developer.apple.com/machine-learning/, 2022.

[141] Jeroen Van der Hooft, Maria Torres Vega, Stefano Petrangeli, Tim Wauters, and Filip De Turck. Tile-based adaptive streaming for virtual reality video. *ACM Trans. Multimedia Comput. Commun. Appl.*, 15(4), December 2019. ISSN 1551-6857. doi: 10.1145/3362101. URL https://doi.org/10.1145/3362101.

List of Figures

List of Tables

List of Listings