
# VOXEL: Cross-layer Optimization for Video Streaming with Imperfect Transmission

Mirko Palmer
Max-Planck-Institut für Informatik
mpalmer@mpi-inf.mpg.de

Malte Appel
Max-Planck-Institut für Informatik
IIJ
malte@iij-ii.co.jp

Kevin Spiteri*
University of Massachusetts Amherst
kspiteri@google.com

Balakrishnan Chandrasekaran
Vrije Universiteit Amsterdam
b.chandrasekaran@vu.nl

Anja Feldmann
Max-Planck-Institut für Informatik
anja@mpi-inf.mpg.de

Ramesh K. Sitaraman
University of Massachusetts Amherst
Akamai Tech
ramesh@cs.umass.edu

## ABSTRACT

Delivering videos under less-than-ideal network conditions without compromising end-users' quality of experiences is a hard problem. Virtually all prior work follow a piecemeal approach—either "tweaking" the fully reliable transport layer or making the client "smarter." 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.

## CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Information systems** → **Multimedia streaming**.

**ACM Reference Format:**
Mirko Palmer, Malte Appel, Kevin Spiteri, Balakrishnan Chandrasekaran, Anja Feldmann, and Ramesh K. Sitaraman. 2021. VOXEL: Cross-layer Optimization for Video Streaming with Imperfect Transmission. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3485983.3494864

*Now at Google.


*CoNEXT '21, December 7–10, 2021, Virtual Event, Germany*

ACM ISBN 978-1-4503-9098-9/21/12.
https://doi.org/10.1145/3485983.3494864

## 1 INTRODUCTION

Video traffic constitutes the majority of Internet traffic [9, 59]. Researchers also forecast that video will account for 82% of all IP traffic by 2022 [9]. The need to stream video traffic 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 [19, 21, 27]. Our focus in this paper is 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 [2, 74].

A video stream is split into *segments*—sequence of bytes spanning a predetermined (typically, 2-10 s) time duration [37]. Streaming these segments over the Internet without interrupting playback at the client (i.e., video player) is a notoriously hard problem. It is exacerbated by the fact that each video segment has an implicit *deadline* within which it must be delivered to the client; the client will, otherwise, *stall* playback to wait for the segment to arrive. Such playback interruptions have a discernible impact on end users and degrade their video-watching experiences, as measured via QoE metrics [15]. There is a rich literature on video streaming, but virtually all of them focus on either "fine-tuning" TCP or making clients "smarter" by using adaptive bitrate (ABR) algorithms.

Prior efforts on fine-tuning TCP for video streaming include sidestepping packet losses [40], supporting real-time delivery [7, 20, 40, 68], adding deadline-awareness [10, 47], using retransmissions for sending new data [45], and using coding techniques for loss recovery [35, 70]. They all use TCP, a reliable transport, although video streaming can tolerate *some* packet loss [17]. Upgrading or improving TCP is also hard. While the QUIC protocol makes it easy to extend the transport [31], 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 [23, 33, 43, 63, 64, 73], 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

[1]There have been a few proposals to support unreliable delivery in QUIC (e.g., [42, 53, 66], but none have been *standardized* or implemented in Google QUIC, as of this writing.

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.

In this paper we propose *VOXEL*, a 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 [17, 32].

★ *First, 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, BETA [32], 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, 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 [73] and BOLA [63]) as well as recent work, e.g., BETA (see §5).

★ *VOXEL is more than a simple research prototype. VOXEL* works seamlessly with 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 [1, 11]) and client side (e.g., in major web browsers [12, 65]). In our evaluations, *VOXEL* streams videos in the 90$^{\text{th}}$-percentile with up to 97% less rebuffering than the state-of-the-art without sacrificing end-users' QoEs.

★ *VOXEL is QoE-metric-agnostic.* We use the all-component SSIM [67] 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 [49] and PSNR.

***Summary of our contributions.***

★ We describe an offline, server-side algorithm that rank orders frames, within each segment, based on the impact of their loss on QoE. The 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).

★ 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.

★ In our tests, *VOXEL* (ABR* with QUIC*) 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.

★ We conducted a real user survey with 54 users, 84% of whom indicated that they prefer videos streamed using *VOXEL* compared to the state-of-the-art.

Our implementation is publicly available on GitHub [50].

## 2 BACKGROUND

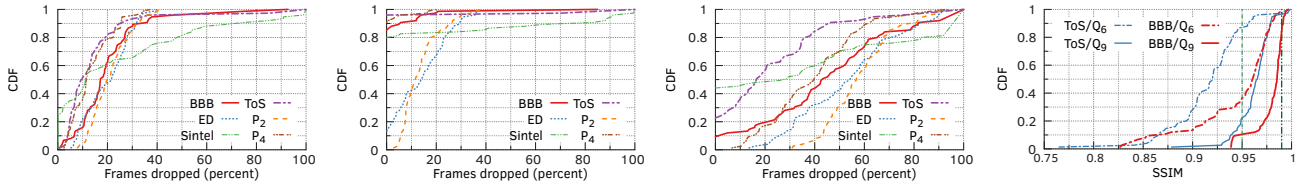
Streaming video over HTTP typically entails using either dynamic adaptive streaming over HTTP (DASH) [60] or HTTP live streaming (HLS) [52]. Although DASH and HLS have similar requirements regarding the video format, we restrict our attention to the codec-agnostic DASH. The video data itself is usually encoded as H.264, the most widely used video codec [5, 16], 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 [60]. To handle varying network conditions, clients utilize an ABR algorithm to determine which of the available quality levels of a segment to download.

***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 [34]. Since frames in our videos consist of only one slice, we use the generic term, frames.

***Adaptive bitrate (ABR) algorithms.*** 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 [39]), base their decision, on estimated network throughput. Time-based ABR algorithms, (e.g., ABMA+ [3]), use segment download times, while buffer-based algorithms rely solely on playback buffer occupancy [24]. Recent hybrid ABR algorithms (e.g., MPC [73] and BOLA [63]) combine throughput and buffer-based approaches to improve performance.

***QoE metrics.*** The structural similarity index (SSIM) is a widely used *full-reference* QoE metric that captures the visual quality of a video stream [67]. To calculate *SSIM scores*, each frame is compared




**(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 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.*

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 [29] and, the more recent, VMAF [49], 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 [29, 49], which is also confirmed by the authors of p.1203 [30]. We focus, hence, on SSIM in our tests, but *also* use VMAF and PSNR, wherever feasible, to demonstrate that *VOXEL is QoE-metric agnostic.*

***Reference quality level.*** More than half of the TV's shipped in 2018 are native 4K [44] and half of YouTube's recommended mobile devices have ≥1440 p screens [76]. YouTube recommends uploading content in up to 4K [75]. 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 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. The quality score, hence, indicates how close the stream's quality is to the highest feasible quality.

## 3 INSIGHTS

Although early academic work on streaming used UDP, recent work and all HTTP streaming use a reliable transport (i.e., TCP). Feamster et al. [17] and, more recently, Palmer et al. [51] demonstrated, however, that video streaming will also 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 [32]. Unlike *VOXEL*, BETA considers loss of only unreferenced B-Frames, uses TCP, and implements only a subset of features of *VOXEL*.

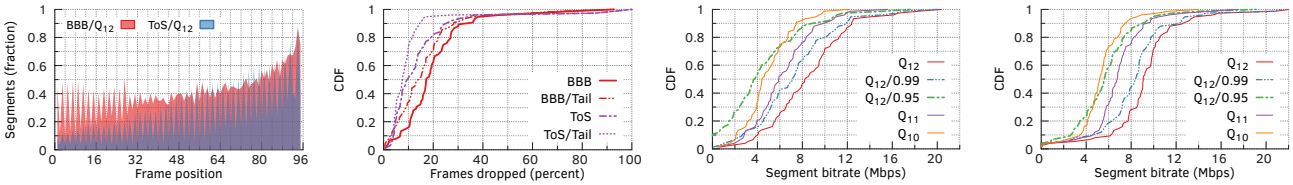
Our goal is to systematically identify which frames can be dropped *a priori*, i.e., in an offline process, while still delivering a high QoE to end users. We now describe the key insights (highlighting similarities and differences with relevant prior work) that underlie our cross-layer approach for 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. 1 in §A 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. 3 in §C) that were retrieved following an approach similar to that of Yeo et al. [72]. We restrict our focus to only the 4 videos in Tab. 1 (in §A) and 2 randomly selected videos from Tab. 3 (in §C) to simplify the plots and explanations. We include a detailed discussion of the analyses of all the YouTube videos in §C. For each video, we selected a 5-minute section comprising 75, 4 s long segments. Each video is transcoded at 13 different bitrates (Tab. 2 in §A) 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.

***1) Drop frames while still delivering a high QoE.***

Typical video content can tolerate some dropped frames without a significant impact on QoE [17, 32, 71]. Fig. 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. [6] 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. 1b repeats the experiment at the lower quality $Q_9$ at 4.3 Mbps (refer Tab. 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. 1d, an SSIM score less than




**(a)** *Frame Distribution* **(b)** $Q_{12}$, *SSIM 0.99* **(c)** *BBB* **(d)** *ToS*

**Figure 2:** *(a) Frames that can be dropped while guaranteeing a specific SSIM are distributed throughout the segments (b) 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—(c) BBB, and (d) ToS.*

0.99. Fig. 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 [57] with perceptible but not detrimental impairment.

This idea of tolerating frame losses was explored in [17] 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 §4.1) than relying on the relative positions of frames (as in [17]) or simply tagging only unreferenced *B*-Frames as "unimportant" [32]. Unlike *VOXEL*, prior work either use a complex solution involving RTP/RTSP [17] or rely on TCP [32].

***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. 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. 2b), 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.

Fig. 2a 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 observed unimportant frames to be distributed throughout the segment in all videos, but we omitted these plots due to space constraints.

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 5.2). To effect the reordering BETA modifies the video files, whereas *VOXEL* only changes the manifest (refer 4.1). 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 7).

***3) Fine-grained quality switching via frame-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. [57] show that perceptual-quality-impairing artifacts caused by packet loss, or frame-drops, induce structural changes that can be captured by SSIMs. Figs. 2c and 2d 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 §4.1) 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 §5.2).

# 4 SYSTEM DESIGN

Streaming video using *VOXEL* is similar to traditional systems, but with several important differences.

★ We prepare a video for streaming offline, by identifying the "important" frames in each video segment. We order the frames in the manifest to prioritize "important" frames (i.e., to download them before the "unimportant" ones) and mark their influence, or lack thereof, towards the QoE.

★ At the transport layer, we build on the QUIC extensions from [51] 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.

This section shows *VOXEL*'s backwards compatibility with existing clients and means for incremental deployment.

## 4.1 Preparing the Video Content

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 [46]. 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 are downloaded.

The prioritization effort does not affect *I*-Frames as they have the highest importance and are always downloaded first. For all other frames, we investigate three different prioritization orders.

① ***Original order***, in which we retain the frames in the same order as generated by the (MPEG) encoder.[2]

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, minimal degradation. This order closely resembles BETA's approach.

③ ***Order by inbound references***, in which we rank order frames based on inbound references. We take both direct and transitive references of a given frame into account for this rank ordering. "Unimportant" frames in the tail end of the segment will be the ones with fewer inbound references than 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.

*Finding the best among the three 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 (calculated from 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 three orderings. Lastly, we pick the ordering with the minimal number of bytes that achieves the required QoE score. A client may fetch bytes beyond this threshold, if conditions permit.

*Extending the manifest.* We update the manifest with frame-level details (see Listing 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 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 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. 1 in §A, 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 [28].

**Listing 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.

## 4.2 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

[2] 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.

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. [51], 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 [51].

*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 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 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 §5) show that *VOXEL* significantly reduces rebuffering, particularly in scenarios with smaller buffer sizes. Confirming, as of 2020, rebuffering to be the most frustrating [41, 58], our user survey shows that an overwhelming majority of participants prefer trading off buffering for quality.

### 4.3 ABR*: Enhancing the ABR Algorithm

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

***Optimize for QoE.*** ABR algorithms such as MPC [73] and BOLA [63] 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.

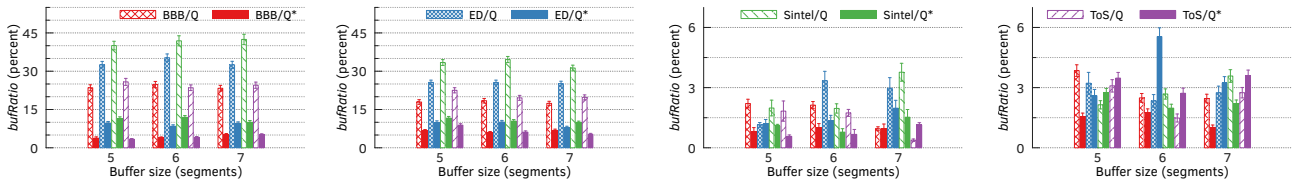
***Support partial-segment downloads.*** Traditional ABR algorithms choose from a limited set of bitrates. *VOXEL* allows partial segment downloads while ensuring frame-header integrity (§4.2) and presents the respective QoE metrics for different download subsets (Listing 1), thereby significantly increasing 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 [32].

***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 §3).

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 [62]. BOLA allows for a custom utility function by design, and has only two tuning parameters $\gamma$ and $V$. Before streaming, *VOXEL* automatically tunes $\gamma$ and $V$ for the video's bitrate ladder characteristics following a calculation described in [63]. 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 [14].

We developed ABR* by extending BOLA-E, a variant of BOLA, described in [62], 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




**(a)** *MPC - T-Mobile* **(b)** *MPC - Verizon* **(c)** *BOLA - T-Mobile* **(d)** *BOLA - Verizon*

**Figure 3:** *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.* ***Due to space constraints, we distribute the plot legends across neighboring plots, in this and all following similar figures.***

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.

## 5 EVALUATIONS

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.

***Videos.*** Of the 14 videos we used earlier in §3, we restrict our attention to 4 widely used videos (Tab. 1 in §A) in video-streaming research, e.g., [22, 36, 38], for the evaluations. 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.15 in §A). Following the encoding procedure outlined in [54], 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. 2. Transcoding was done using 2-pass encoding, preset *slow* and no custom encoding settings. Unlike [54], we used 4 s segments, which are a good balance between encoding quality and fast quality switching [37]. 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 §A.

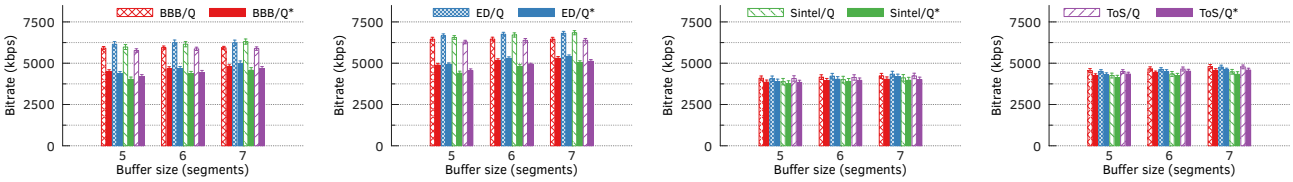
***Network testbed.*** We ran all experiments on a testbed consisting of multiple sets of three bare-metal Linux machines running Linux (Debian 9) with the *4.19* kernel. Each triplet emulates a one-hop network—a server and client connected via an intermediate host (or router). We *shape* the traffic 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 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 (see §B, due to space constraints). We configured a 30 ms delay on the router-to-client link, to emulate typical "last mile" latencies.

***Network traces.*** We used 5 different network traces: 3 LTE (4G) traces from [69], a 3G trace collected in Norway from [56], and a fixed-line broadband network trace from the FCC dataset [13]. We primarily focus on cellular networks for two reasons: (a) Video traffic constitutes a substantial portion of IP traffic on mobile networks [8], 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 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.

***ABR algorithms.*** In our evaluations we compare four ABR algorithms against ABR*. They include BOLA [63] and MPC [73], two state-of-the-art ABR algorithms, and, the more recent, BETA [32] 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.[3] 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., [33, 43, 63]), but small buffers are crucial for supporting low-latency or live-streaming-like applications.

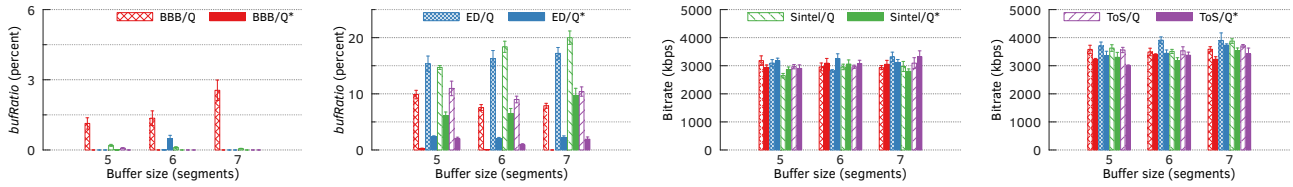
***Experiments.*** An *experiment* involves streaming a video from a server to a client via the router, under a fixed *configuration*. A configuration specifies the ABR algorithm, buffer size, video, and network trace. Unless otherwise stated, we repeat each experiment 30 times and report the aggregate 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.15 in §A). To evaluate the performance of each trial, we instrumented our video streaming software to obtain segment-level timing information with packet-level precision.

[3] We implemented BETA from scratch, to the best of our ability, based on the details in their paper [32], since it is not publicly available.




**(a)** *MPC - T-Mobile* **(b)** *MPC - Verizon* **(c)** *BOLA - T-Mobile* **(d)** *BOLA - Verizon*

**Figure 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)** *BOLA; 20 Mbps* **(b)** *MPC; 20 Mbps* **(c)** *BOLA; 20 Mbps* **(d)** *MPC; 20 Mbps*

**Figure 5:** *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).*

## 5.1 ABR Algorithms with QUIC*

Below we evaluate the implications of an incremental deployment and, most importantly, ascertain the need for a cross-layer, coordinated approach for optimizing video streaming. To this end, we 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.

***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, or 5 through 7 segments—are a representative lower bound of prior work [39, 43, 63]. Fig. 3 shows the 90$^{\text{th}}$-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. 3a) and Verizon (Fig. 3b) traces. The inferences hold for the AT&T, 3G, and FCC traces (omitted to conserve space).

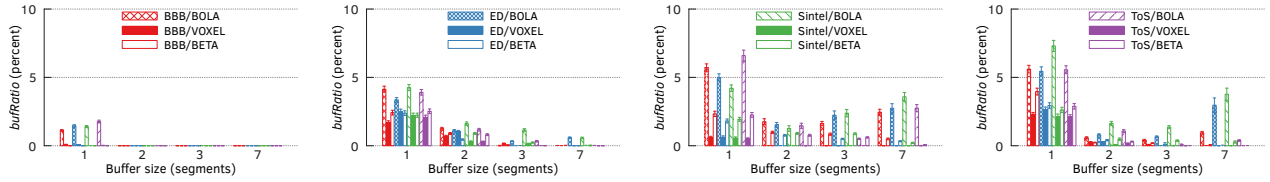
The 90$^{\text{th}}$-percentile improvements in *bufRatio*s for MPC is on average 71.7% larger than that for BOLA (9.2%), mainly because MPC's network-throughput prediction performs poorly for our traces. To shed light on how QUIC* lowers the *bufRatio*s we estimate the average bitrates across all trials for each video under different configurations. Unmodified ABR algorithms seem to trade off bitrates (Fig. 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. 3c & 3d), 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.

***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 [61] 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.

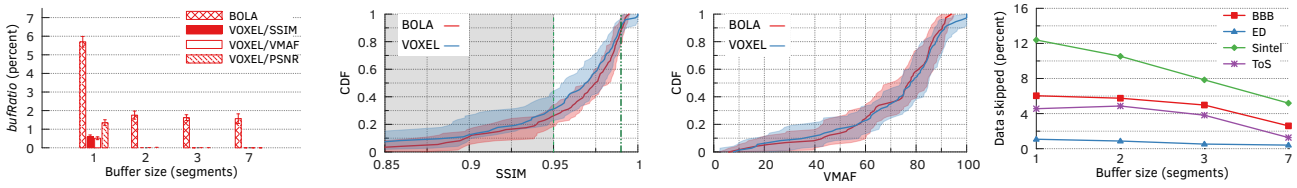
The link capacity in all scenarios was 20 Mbps. We measure, for each value of $\mathcal{T}$, the 90$^{\text{th}}$-percentile of *bufRatio*s and average bitrates across five trials for the three ABRs as before. ABR algorithms using QUIC* experience much less rebuffering than when using QUIC (Fig. 5). Even though ABR algorithms experience a slight reduction in average bitrates, the improvements in *bufRatio*s are substantial. MPC, again, shows with 82% 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 rebuffering under varying network conditions. Without a meticulous redesign, unsurprisingly, ABR performance might suffer under some network conditions.



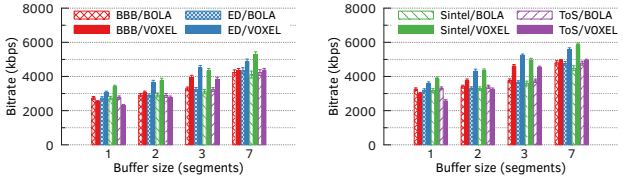

**(a)** *AT&T* **(b)** *3G* **(c)** *Verizon* **(d)** *T-Mobile; VOXEL less aggressive*

**Figure 6:** *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. 17c in §D.*




**(a)** *BBB - Verizon; diff. QoE metrics* **(b)** *BBB - Verizon; SSIM - distribution* **(c)** *BBB - Verizon; VMAF - distribution* **(d)** *Percent of segment data dropped*

**Figure 7:** *(a) bufRatio, (b) SSIM, and (c) VMAF measurements while streaming BBB over Verizon show that VOXEL outperforms BOLA independent of the quality metric, emphasizing its agnostics. (d) the percentage of data dropped or "skipped" by VOXEL when streaming different videos over Verizon as a function of buffer size.*




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

**Figure 8:** *VOXEL outperforms BOLA/QUIC even in bitrates under different network conditions.*

## 5.2 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 (§3). 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 against *VOXEL* in our highly varying network traces as well as challenging cross-traffic scenarios.

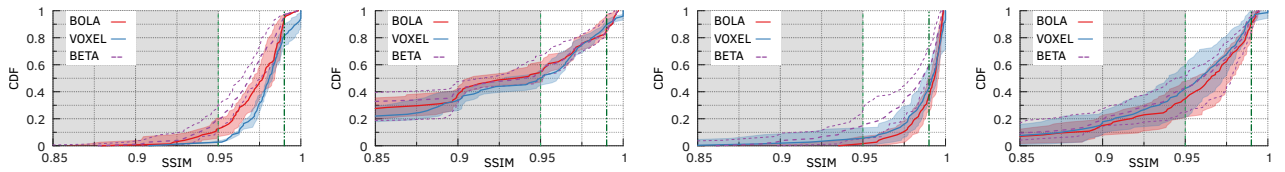
***In-lab trials with network traces.*** In our evaluations, under all network conditions emulated using the different trace files, *VOXEL* experienced either substantially low or virtually zero rebuffering. We mainly focus, due to space constraints, on the plots for AT&T (Fig. 6a), 3G (Fig. 6b), Verizon (Fig. 6c) and T-Mobile (Fig. 6d). We observe similar results for FCC (Fig. 18a) in Appendix D. *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 90$^{th}$ 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. 8 reveals the reason: With a large buffer, BOLA aggressively requests higher quality segments but fails to deliver them in 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, 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. 7a, 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. 8. 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. 6c). When comparing the CDFs of all streamed segments' SSIMs for both BOLA/QUIC and *VOXEL* (see Fig. 7b), 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. 7c); *VOXEL* even




**(a)** *ToS - AT&T; 2-segment buffer* **(b)** *Sintel - 3G* **(c)** *ED - Verizon* **(d)** *BBB - T-Mobile; VOXEL less aggressive*

**Figure 9:** *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). 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.*

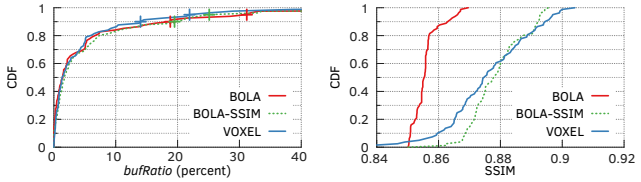



**Figure 10:** *BOLA, BOLA-SSIM and VOXEL have 7.9%, 8.2% and 5.1% mean bufRatio, respectively, when streaming BBB over 86 3G traces with a 1-segment buffer. Pointing out the 90$^{th}$ and 95$^{th}$, the bufRatio difference mostly lies in the upper percentiles.*

achieves perfect scores for several segments. The same holds for *bufRatio*, in Fig. 7a: We show that *VOXEL* almost eliminates rebuffering with either *SSIM*, *VMAF*, or *PSNR*. We omit plotting PSNR, as it 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 [32]. *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. 7d 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, if the SSIM difference between two bitrate quality levels is small, we will not insert SSIM quality steps (or virtual quality levels) in between.

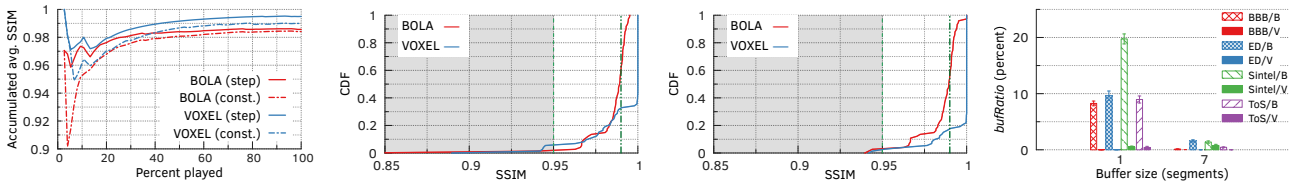
In the AT&T LTE network trace experiment (Fig. 9a), *VOXEL* can even improve SSIMs. When we experimented with a two-segment-long playback buffer, none of the three systems experienced any rebuffering (Fig. 6a). Yet, *VOXEL*'s distribution of SSIM scores (Fig. 9a) is below (i.e., *better than*) BOLA and 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. 6c 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. 9c). BETA and *VOXEL*, however, only seemingly lose against BOLA, since this is part of the trade-off which significantly reduces *bufRatio*. Overall, our evaluation shows that in most cases *VOXEL* is clearly superior to BOLA as well as BETA.

One exception to the above performance of *VOXEL* is the T-Mobile result in Fig. 6d. 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. 9d) but also in *bufRatio* (Fig. 6d). 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. 17d for SSIM and Fig. 17c 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. 10 isolates the effect of the two updates to BOLA described in §4. We evaluated BOLA, BOLA-SSIM, and *VOXEL* by streaming BBB over 86 3G traces collected by Riiser et al. [56] using a 1-segment buffer. The low average bandwidth of these 3G traces helps to stress-test the ABR algorithms (Fig. 9b). 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 abandonments enabled by QUIC*. The smart abandonments are 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.

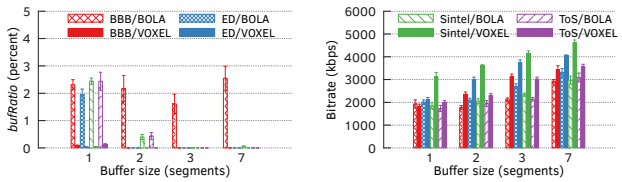
***In-lab trials without partial reliability.*** To quantify the benefits of partial reliability, we repeated the trials with network traces, and kept all of *VOXEL* features intact except for unreliable streams, i.e., we enforced fully reliable transfers. With the Verizon trace (Fig. 18d in §D) the *bufRatio* doubled without partial reliability across all buffer sizes. Even when streaming under the more challenging T-Mobile trace (§D, Fig. 18c), *VOXEL* outperforms a reliable streaming system in all cases, except for ED with a 1-segment buffer.

***In-lab trials with cross traffic.*** *VOXEL* outperforms the state-of-the-art implementations even in the presence of a substantial




**(a)** *BBB; SSIM progression* **(b)** *BBB; SSIM distribution* **(c)** *BBB; SSIM distribution* **(d)** *In-the-wild trials; bufRatio*

**Figure 11:** *(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); and comparison of bufRatio (d) of BOLA/QUIC, labelled "B", and VOXEL, labelled "V" in in-the-wild trials.*




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

**Figure 12:** *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.*

volume of cross-traffic. Fig. 12a 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. 12b 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.

***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 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. 11a). 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. 11b shows that *VOXEL* obtains an SSIM score of 1.0 for 65% of the segments. BOLA, in contrast, does not get any perfect scores.

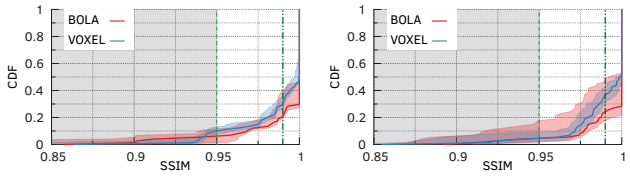
We conduct a second experiment where we start at a marginally higher 10.75 Mbps and, after 70 s, drop down to 10.5 Mbps, the same throughput used for the first experiment. *VOXEL* has greater freedom in selecting a suitable quality and, unsurprisingly, outperforms BOLA again. The finite set of quality 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. 11c). In contrast, *VOXEL* copes well with the network conditions, delivering 80% of the segments with a perfect 1.0 SSIM score.

***In-the-wild 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*. We streamed videos throughout the day alternating between BOLA and *VOXEL* using all four videos with a small 1-segment and a large 7-segment playback buffer. While BOLA and *VOXEL* achieve low rebuffering for large buffers, *VOXEL* outperforms BOLA significantly for small buffers (Fig. 11d). If we compare the SSIM scores in the low-buffer experiment (Fig. 13), *VOXEL* performs comparably and, thus, does not unnecessarily trade off rebuffering 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.

## 5.3 Real User Survey

We conducted a real user study with 54 participants recruited from different universities. We used one-minute-long video clips extracted from our in-lab experiments; we chose videos streamed 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. 14). The playback fluidity was important and rated 1.7 points higher for *VOXEL*.




**(a)** *BBB* **(b)** *ToS*

**Figure 13:** *In-the-wild trials: SSIM score distribution when streamed with a 1-segment buffer.*

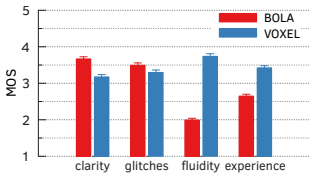



**Figure 14:** *Mean Opinion Scores along 4 dimensions: Clarity (i.e., visual quality), glitches (i.e., noticeable artifacts), fluidity (i.e., re-buffering) and overall viewing experience.*

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 RELATED WORK

Almost all prior work on ABR algorithms (e.g., [24, 33, 39, 43, 64, 73]) leave the underlying transport—typically, TCP—intact. Bhat et al., show that porting such ABR algorithms to QUIC does not suffice [4].

Prior work also looked at "tweaking" or "tuning" TCP for video streaming. TCP variants such as TCP-RTM [40] and TL-TCP [47] either ignore retransmissions or avoid retransmitting data that have already missed the deadline. Both complicate application design, making deployment impractical, if not impossible. Brosh et al. [7] suggest optimizations to make TCP more friendly for delivering real-time media. In a similar vein, Goel et al. [20] 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. [45] propose a TCP variant that uses retransmissions to deliver new data. This idea alleviates some but not all of the overhead.

There is work but seldomly a full system on dropping frames. Yahia et al. [71] propose video delivery schemes that exploit HTTP/2 mechanisms to drop frames during low-latency live streaming. They use, however, only single-bitrate videos without ABR algorithms. Stewart et al. [55] present PR-SCTP, a partial reliability extension to SCTP. It allows to specify, on a per-message basis, if a retransmission should take place. As a transport protocol, it does not have a notion of video frames, and thus, would require an application on top of it to determine which frames can be dropped.

Feamster et al. [17] explore the effect of selective reliability for streaming video via RTP, necessitating substantial changes to the network stack. *VOXEL*, in contrast, requires minimal changes and can be deployed incrementally. Fouladi et al. [18] designed *Salsify*, a system for video conferencing, that adjusts the encoding quality per video frame. This approach is less suitable for video-on-demand 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. [32] 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. 1 in §A, 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.

## 7 CONCLUDING REMARKS

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 CDNs and content providers—a novel use case for *VOXEL* that we leave to future work.

*Managing the overheads.* *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). Lastly, it is a *one-time* computation, i.e., once enriched the manifest can be reused indefinitely.

*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. 7a, relatively good rebuffering performance when using VMAF or PSNR [25, 26]—other widely used QoE metrics in the literature. We, thus, invite the community to consider evaluating new metrics that can accurately measure the QoE impact of imperfect segments.

We have released our implementation as an open-source artifact to motivate exploration of other ABR designs and also foster reproducible research (refer §E for details). We hope the design and evaluation of *VOXEL* serves as a "call to arms" for the video-streaming community to investigate this optimization landscape.

## ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grants No. CNS-1763617 and No. CNS-1901137.

# REFERENCES


[1] Akamai Technologies. 2018. Community Blog, FAQ: QUIC Native Platform Support for Media Delivery Products. https://tinyurl.com/yab23e4f.

[2] Akamai Technologies. 2018. Hotstar And Akamai Set Global Streaming Record During VIVO IPL 2018. https://bit.ly/2MtJLbP.

[3] A. Beben, P. Wiśniewski, J. Mongay Batalla, and P. Krawiec. 2016. ABMA+: Lightweight and Efficient Algorithm for HTTP Adaptive Streaming. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*.

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

[5] Bitmovin. 2019. Bitmovin Video Developer Report 2019. https://bit.ly/2toCyW1.

[6] A. C. Brooks, Xiaonan Zhao, and T. N. Pappas. 2008. Structural Similarity Quality Metrics in a Coding Context: Exploring the Space of Realistic Distortions. *Trans. Img. Proc.* (Aug. 2008).

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

[8] Cisco. 2019. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017-2022 White Paper. https://bit.ly/366EFO5.

[9] Cisco. 2019. Visual Networking Index: Forecast and Trends, 2017-2022 White Paper. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

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

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

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

[13] Federal Communications Commission. 2016. Raw Data - Measuring Broadband America. https://bit.ly/35UgFgJ.

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

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

[16] encoding.com. 2018. Global Media Format Report 2018. https://bit.ly/2HXfSxn.

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

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

[19] D. Ghadiyaram, A. C. Bovik, H. Yeganeh, R. Kordasiewicz, and M. Gallant. 2014. 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)*.

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

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

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

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

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

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

[26] Quan Huynh-Thu and Mohammed Ghanbari. 2012. The accuracy of PSNR in predicting video quality for different video scenes and frame rates. *Telecommunication Systems* (06 2012).

[27] Akamai Technologies Inc. 2017. 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.

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

[29] ITU. 2017. 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.

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

[31] Jana Iyengar and Martin Thomson. 2018. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft. IETF.

[32] Cyriac James, Mea Wang, and Emir Halepovic. 2019. BETA: Bandwidth-Efficient Temporal Adaptation for Video Streaming over Reliable Transports. In *Proceedings of the 10th ACM Multimedia Systems Conference (MMSys '19)*.

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

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

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

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

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

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

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

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

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

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

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

[44] IHS Markit. 2019. How to delight consumers in a connected world? https://bit.ly/3A9CpTY.

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

[46] MPEG. 2011. Media presentation description and segment formats. https://bit.ly/3ot9W4w.

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

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

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

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

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

[52] Roger Pantos and William May. 2017. HTTP Live Streaming. RFC 8216.

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

[54] Yanyuan Qin, Shuai Hao, K. R. Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. 2018. 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)*.

[55] Michael A. Ramalho, Qiaobing Xie, Randall R. Stewart, Michael Tüxen, and Phillip Conrad. 2004. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758.

[56] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys '13)*.

[57] Piotr Romaniak and Lucjan Janowski. 2010. How to Build an Objective Model for Packet Loss Effect on High Definition Content Based on SSIM and Subjective Experiments. In *Future Multimedia Networking*, Sherali Zeadally, Eduardo Cerqueira, Marília Curado, and Mikołaj Leszczuk (Eds.). Springer Berlin Heidelberg.

[58] Ross Benes. 2018. Buffering Is the Streaming Snafu that Won't Go Away. https://bit.ly/3h0slUf.

[59] Sandvine. 2019. The Global Internet Phenomena Report. https://bit.ly/37TtuaD.

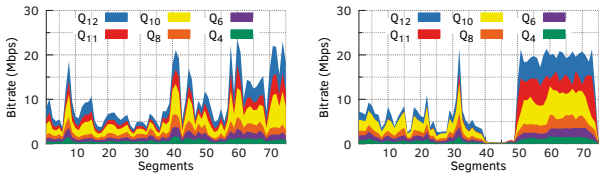
[60] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia* 18, 4 (Oct. 2011).

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

[62] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2019. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM Trans. Multimedia Comput. Commun. Appl.* 15, 2s (July 2019).
[63] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2020. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking* 28, 4 (2020).
[64] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. 2016. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of ACM SIGCOMM '16*.
[65] LiteSpeed Technologies. 2020. What is HTTP/3 Check? https://http3check.net/about.
[66] Philipp S. Tiesel, Mirko Palmer, Balakrishnan Chandrasekaran, Anja Feldmann, and Joerg Ott. 2017. *Considerations for Unreliable Streams in QUIC*. Internet-Draft. IETF. https://datatracker.ietf.org/doc/html/draft-tiesel-quic-unreliable-streams-01
[67] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.* 13, 4 (April 2004).
[68] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. 2006. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. Netw.* 14, 6 (Dec. 2006).
[69] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. 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.
[70] Huahui Wu, Mark Claypool, and Robert Kinicki. 2005. Adjusting Forward Error Correction with Temporal Scaling for TCP-friendly Streaming MPEG. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 4 (Nov. 2005).
[71] Mariem Ben Yahia, Yannick Le Louedec, Gwendal Simon, Loutfi Nuaymi, and Xavier Corbillon. 2019. HTTP/2-based Frame Discarding for Low-Latency Adaptive Video Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 15, 1 (Feb. 2019).
[72] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural Adaptive Content-Aware Internet Video Delivery. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*.
[73] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of ACM SIGCOMM '15*.
[74] YouTube. 2012. Mission complete: Red Bull Stratos lands safely back on Earth. https://youtube.googleblog.com/2012/10/mission-complete-red-bull-stratos-lands.html.
[75] YouTube. 2017. Video resolution & aspect ratios. https://bit.ly/3ac4m10.
[76] YouTube. 2018. The best experience on YouTube Signature Devices. https://devicereport.youtube.com.
[77] youtube-dl. 2020. Download videos from YouTube. https://ytdl-org.github.io/youtube-dl/about.html.

**Table 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 |




**(a)** *ED* **(b)** *Sintel*

**Figure 15:** *Variations in segment sizes across a subset of the 13 available quality levels for two selected videos.*

## A VIDEOS FROM PRIOR WORK

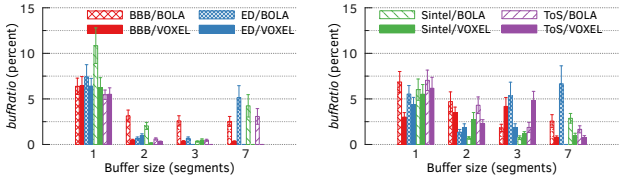
For both demonstrating the key insights and evaluating the design of *VOXEL*, we chose 4 videos widely used in video-streaming research (see Tab. 1), namely, *Big Buck Bunny (BBB)*, *Elephants Dream (ED)*, *Sintel*, and *Tears of Steel (ToS)*. Since the actual videos are quite long in duration—ranging 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. 1). 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. 2. As ED was not available in 4K, $Q_{11}$ 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* [77], and Netflix [48]. These capped VBR videos have a peak bitrate of at most 200% the average bitrate (see Fig. 15) and 24 fps. The clips exhibit (in Fig. 15), 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 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 |

## B LONG NETWORK QUEUES

We ran network trace experiments with a 750-packets-long queue (see Fig. 16), to acknowledge typical commercial-LTE behavior for popular on-site-cached content. Video content, cached on premise of an LTE provider, will traverse an LTE network path with typically very long network queues. 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. 16b, 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.




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

**Figure 16:** *bufRatios for a 750-packets-long network queue.*

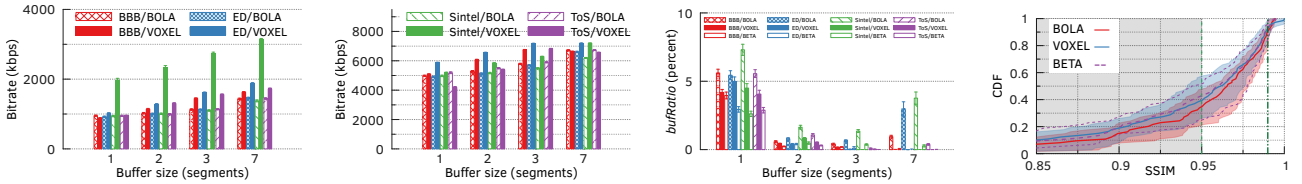
## C PUBLIC YOUTUBE VIDEOS

To confirm whether the insights in §3 hold for a broader spectrum of videos, and not only for the, although, widely used videos in Tab. 1, we analyzed a set of 10 publicly available videos from Youtube (see Tab. 3). In Fig. 19, we present the analyses of some of the videos that were not already presented in §3. 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. 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. 19c). 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.

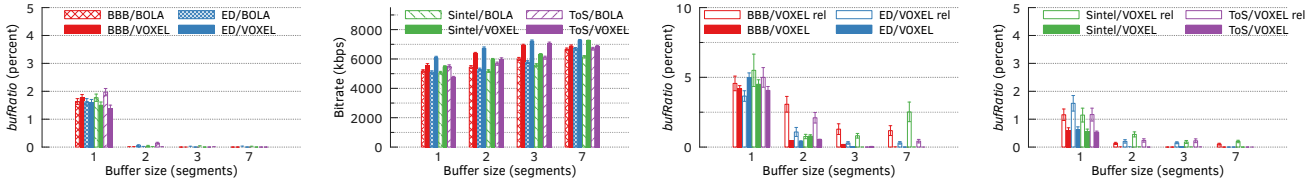
**Table 3:** *Overview of public YouTube videos used to generalize our insights to a diverse set ot Internet videos.*

| *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 |



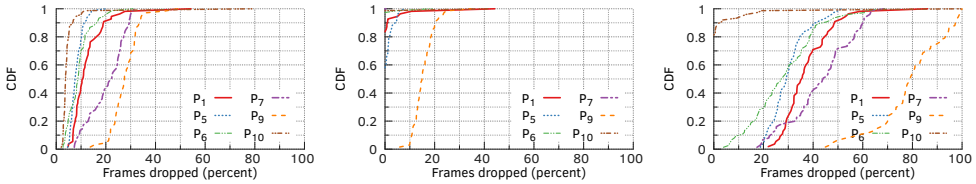

**(a)** *3G; Average bitrates* **(b)** *AT&T; Average bitrates* **(c)** *T-Mobile; VOXEL too aggressive* **(d)** *BBB - T-Mobile; VOXEL too aggressive*

**Figure 17:** *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).*




**(a)** *FCC; bufRatio* **(b)** *FCC; Average bitrates* **(c)** *T-Mobile; optional partial reliability* **(d)** *Verizon; optional partial reliability*

**Figure 18:** *bufRatio and average bitrates while streaming with VOXEL utilizing the FCC trace (a-b), and a bufRatio comparison between VOXEL without partial reliability enabled, denoted as "VOXEL rel", and VOXEL itself, over (c) T-Mobile and (d) Verizon.*




**(a)** $Q_{12}$, *SSIM 0.99* **(b)** $Q_9$, *SSIM 0.99* **(c)** $Q_9$, *SSIM 0.95*

**Figure 19:** *The insights from §3 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.*

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.

## D VOXEL

*VOXEL* is able to almost completely remove rebuffering for the AT&T trace (recall Fig. 6a in the main paper), while still delivering comparable or higher average bitrates (Fig. 17b), even with small buffers. For 3G (recall *bufRatio* in Fig. 6b) and bitrates in Fig. 17a, and FCC (Figs. 18a, 18b) *VOXEL* is not able to remove rebuffering entirely, but still delivers comparable or better performance than the state-of-the-art. To determine the influence of partial reliability on the overall performance of *VOXEL*, we ran the same trace experiments with *VOXEL* on T-Mobile (Fig. 18c) and Verizon (Fig. 18d) but disabled unreliable streams. Enabling partial reliability significantly reduces the *bufRatio* in all but one cases.

## E ARTIFACTS

The source code of the *VOXEL* implementations is available at https://github.com/derbroti/VOXEL and includes:

- the network traces used in the experiments,
- scripts to build the system and perform a video stream,
- a README on how to use the provided scripts.

To compact the repository it holds only the modified files, the remaining Chromium code base is retrieved during the initial setup. The system is headless, thus, the streamed video is not rendered but it and extensive logging is written to disk.

The video files used in the experiments and the extended manifests required to use *VOXEL*'s features are available at https://nextcloud.mpi-inf.mpg.de/index.php/s/e8e3C977wg2Kkty.

Each video folder[4] contains the video files, encoded in thirteen quality levels, and the extended manifest (`*.mpd`).

[4]The Chromium server requires the served files to include an HTTP-response-header.