The QUIC Fix for Optimal Video Streaming

Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, Anja Feldmann
Max-Planck-Institut für Informatik
{mpalmer, tkrueger, balac, anja}@mpi-inf.mpg.de

ABSTRACT

Within a few years of its introduction, QUIC has gained traction: a significant chunk of traffic is now delivered over QUIC. The networking community is actively engaged in debating the fairness, performance, and applicability of QUIC for various use cases, but these debates are centered around a narrow, common theme: how does the new reliable transport built on top of UDP fare in different scenarios? Evaluation of unreliable delivery in QUIC remains largely unexplored.

The option for delivering content unreliably, as in a best-effort model, deserves the QUIC designers' and the QUIC community's attention. We propose extending QUIC to support unreliable streams and discuss a simple use case of video streaming—an application that dominates the overall Internet traffic—that can leverage the unreliable streams and potentially bring immense benefits to network operators and content providers. We demonstrate, using controlled-environment trials, how to combine reliable and unreliable streams to outperform TCP and QUIC in video streaming.

CCS CONCEPTS

• **Networks → Transport protocols**; *Network protocol design*;

KEYWORDS

Video Streaming, Partial Reliability, QUIC

ACM Reference Format:
Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, Anja Feldmann. 2018. The QUIC Fix for Optimal Video Streaming. In *Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18), December 4, 2018, Heraklion, Greece.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3284850.3284857

1 INTRODUCTION

Google's Quick UDP Internet Connections (QUIC) protocol offers TCP-like properties at the application layer on top of UDP [16, 26]. Although the protocol was designed and made public only recently, in 2013, it is rapidly gaining adoption: nearly 6% of the global Internet traffic flows over QUIC, and many CDNs and content providers already support the protocol [26]; Google, unsurprisingly, leads the Internet in QUIC adoption and delivers more than 40% of its traffic via QUIC [39]. Given the browser support, notably with the Google Chrome browser even enabling the protocol by default, together with the popularity of Google's services—the infrastructure of which support QUIC—these adoption statistics will quickly and significantly increase.

Although QUIC seems to deliver data in a reliable, secure, and fast manner, this fixation on *only* the reliable-delivery aspect of the protocol (and, consequently, the lack of support for unreliable delivery) needs a closer examination. Naturally, we ask the following questions: (a) Is the lack of unreliable streams in QUIC really an issue? (b) Is there a clear use case for a *selectively or partially reliable* transport, where an application can seamlessly multiplex reliable and unreliable streams over a single connection? (c) Is it practical to extend QUIC to offer a partially reliable transport?

To highlight a need to reconsider the strict adherence to reliable transport, we focus on one class of traffic delivered, today, via QUIC—video streaming. Video traffic constitutes a significant share of traffic delivered using QUIC [4, 26].[1] The inherent challenges in streaming "real-time" video traffic [21, 40] over varying, and sometimes less than ideal, network conditions are only exacerbated by the choice of a *reliable* transport—so far, TCP. It is well known that TCP is not suited for video streaming: the rich body of prior work on optimizing and extending TCP, and adaptive bitrate (ABR) selection attest to this observation [11, 19, 22, 31, 50]. TCP retransmissions of lost packets in a video stream, inadvertently lead to *stalls* in the video stream. TCP also performs poorly when it encounters packet losses that are not due to congestion. By shunning unreliable delivery, QUIC, thus, falls trap to most, if not all, of TCP's problems for video streaming; in some instances, QUIC has been shown to perform even worse than TCP for video streaming [3].

The rationale for streaming video via TCP (or, generally, the fixation on reliable transport), today, is rooted in the economics and feasibility of streaming infrastructure deployment. More than 52% of today's Internet traffic is delivered by content delivery networks (CDNs) [6]. When we consider the massive, distributed infrastructure and mature software stack that CDNs have already deployed for delivering Web traffic, the idea of streaming video over HTTP, using dynamic adaptive streaming over HTTP (DASH) or HTTP live streaming (HLS) sounds appealing and practical. This choice of HTTP, unfortunately, ties video streaming to TCP. But with CDNs (e.g., Akamai) and popular Web browsers (e.g., Google Chrome) already supporting QUIC, it is worth revisiting the status quo in video streaming [1, 35, 39, 48].

We share a simple observation to highlight that reliable transports are ill-suited for video streaming: video data consists of different types of frames, some types of which do *not* require reliable delivery. The loss of some types of frames has minimal or no impact (since such losses can be recovered) on the end-user quality

[1] Although this video traffic over QUIC is only from Google, its YouTube video streaming service is one of the largest video serving platforms in the Internet.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EPIQ'18, December 4, 2018, Heraklion, Greece
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6082-1/18/12…$15.00
https://doi.org/10.1145/3284850.3284857

of experience (QoE) [9]. Therefore, by adding support for unreliable streams in QUIC and offering a selectively reliable transport, wherein not all video frames are delivered reliably, we can optimize video streaming and improve end-user experiences. This approach has several advantages: (a) it builds atop QUIC that is rapidly gaining adoption; and (b) it involves only a simple, backward compatible, incrementally deployable extension—support for unreliable streams in QUIC. These advantages taken together make this approach safe, easy, and practical to deploy.

We propose a simple extension to QUIC: the addition of unreliable streams. To demonstrate the benefits of this extension for video streaming and address the non-trivial challenges of combining both unreliable and reliable transport, we present *ClipStream*. [2] Our approach is motivated by a simple observation: not all frames in a video encoding scheme, such as the widely used H.264, are equally "important"; some frames (e.g., *I*-Frames) are more "important" than others (e.g., *B*- and *P*-Frames). "importance" refers to the implications of the loss of a frame, contained in a video stream, for the QoE that an end user attributes when watching that video.

Our streaming solution, *ClipStream*, thus, uses reliable transport for the important frames and unreliable transport, for all other frames. To tackle losses in the unreliable stream, *ClipStream* uses forward error correction (FEC), as required. Supporting such a partially reliable stream, however, introduces other non-trivial challenges, e.g., synchronization of the streams. Demonstrating that the partially reliable stream fares well compared to TCP and QUIC using controlled-environment trials, and addressing the challenges in using it for video streaming is the central theme of this paper. We summarize our contributions as follows.

- ⋆ We propose the addition of unreliable streams to QUIC. We discuss the ease of implementation of this extension and its implications for applications.
- ⋆ We motivate the extension of QUIC through a simple, practical use case: video streaming. To this end, we present *ClipStream*, a hybrid transport protocol that offers selective (or partial) reliability; *ClipStream* provides reliable transport for frames that explicitly request it, and unreliable, best effort transport (protected by FEC) for the rest.
- ⋆ We present preliminary evaluations—using experiments in controlled environments—that show *ClipStream* outperforms other solutions by a significant margin: even under 1.28% of loss, our approach delivers the video stream without compromising video quality, i.e., users see little or no visible quality degradation when viewing the video.

2 THE STATUS QUO

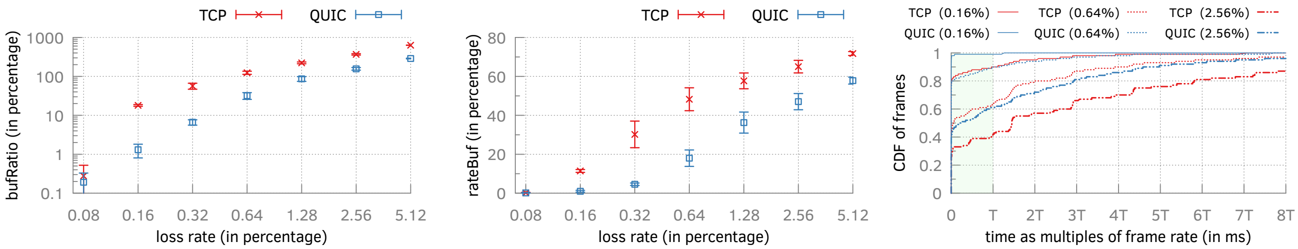
Streaming video over a reliable transport has remained the status quo for a long time, but this scheme suffers to sustain a high end-user QoE when the network conditions are less than ideal. To highlight some of the problems with current video streaming solutions we performed a simple experiment where we, in the lab, repeatedly streamed the "Big Buck Bunny" video (described in Tab. 1) across a lossy link. For details on the setup refer §5. We varied the loss rates from 0.08% to 5.12%, and set the link bandwidth to 20 Mbps and delay to 30 ms. We repeated the experiment with several other choices for network parameters and also using other videos, and observed similar results (not shown).

To assess the performance of a streaming solution, we rely on commonly used, industry-standard metrics—e.g., *buffering ratio (bufRatio)* and *rate of buffering (rateBuf)* [7]. *bufRatio* is defined as the ratio of time spent in re-buffering to the total video duration, and *rateBuf* is the ratio of the frequency of re-buffering events to the total number of video frames. The former captures for each instance of a re-buffering event the duration for which it lasted and affected end users' experiences, while the latter only captures the rate at which end users are interrupted in the course of watching a video.

TCP Despite its shortcomings for streaming videos over the Internet, *TCP* is still the dominant transport protocol for video streaming, due to the widespread use of DASH [38]. The rich body of prior work on optimizing TCP, adaptive bitrate selection algorithms, or TCP variants highlights TCP's shortcomings [11, 19, 22, 31, 50]. TCP retransmits lost packets without considering if these retransmissions are "useful" for the video player; unnecessary retransmissions introduce stalls and degrade the quality of the video stream. Besides, it is well-known that TCP performs poorly when it encounters packet losses that are not due to congestion. Fig. 1(a) shows *bufRatio* as a function of loss, and, per this figure, even at a loss rate of 0.16%—lower than that typically observed in the Internet [44]—the video player spends 20% of the total video time in stalls (i.e., in waiting for the lost packets to arrive at the playback buffer). To put this bufRatio in perspective, note that a 1% of *bufRatio* can reduce user engagement by more than 3 minutes [7]. The rate of re-buffering events in Fig. 1(b) is also high: at 0.64% loss TCP introduces on average 105% of re-buffering. A recent study indicates that traffic policing is highly prevalent world-wide and induces, globally, an average loss rate of over 20% [10]: streaming video over TCP under such loss rates is infeasible.

QUIC Google's QUIC protocol [16] takes a positive, albeit small, step forward towards improving the status quo. QUIC vastly improves connection establishment times, which might lower the initial video buffering times, but Ghasemi et al. empirically show that the impact of throughput on end-users' video quality is higher than that of latency [14]. QUIC packs support for better bandwidth estimation and pluggable congestion control mechanisms, and its transport streams allow applications to seamlessly multiplex several requests or data exchanges on a single connection to avoid head-of-line blocking. The current design, however, demands the use of reliable transport even though, in principle, unreliable transport options and error correction schemes could be supported. Due to this strict adherence to reliable transport, QUIC inherits some of TCP's issues: Fig. 1(c) shows that even at a loss rate of 0.64%, QUIC fails to deliver 10% of the video frames, i.e., these frames arrive much later than when they were required, thereby causing stalls. Our experiments in a controlled environment show, typically (i.e., in the median), a relatively high *bufRatio*, in Fig. 1(a), and *rateBuf*, in Fig. 1(b), even at a loss rate of 0.64%.

[2] *ClipStream*, our hybrid approach, has no relation to the online video platform with a nearly identical name.



(a) Percentage of playback time spent in buffering.

(b) Percentage of frames where stall events occurred.

(c) CDF of frame arrival times.

Figure 1: ***TCP and QUIC are not well suited for video streaming. Even at a loss rate of*** 0.64% ***TCP (QUIC) encounters, in the median,*** 105% ***(***30%***) buffering, per Fig. (a), with*** 50% ***(***19%***) of stall events, per Fig. (b). At this loss rate, Fig. (c) shows that TCP (QUIC) delivers*** 64% ***(***90%***) of frames before the deadline (region shaded in green).***

While ABR schemes help in alleviating some of the issues, they are still akin to "band-aids": they are designed to fix transient problems that the underlying transport fails to handle; besides, switching bitrates has implications for the end-user QoE [12, 18]. In case of QUIC, surprisingly, prior work also show that ABR schemes ported to QUIC operate poorly compared to TCP [3]. Simply switching to UDP for video streaming also does not suffice. The inherent unreliability of UDP necessitates the use of coding or error-correction techniques to recover lost packets. *Blindly* coding every packet, in an application-agnostic manner, to recover from losses poses problems: error correction schemes have a significant overhead, and unrestricted use of such schemes even by a small fraction of the users on a network will add significant load (or traffic) on the network. Besides, without proper congestion control, the UDP streams will not share network resources equitably with other TCP flows.

3 A PRIMER ON STREAMING

Today, video streams are being delivered typically via HTTP using either DASH [42] or HLS [34]. While both, DASH and HLS, have similar requirements regarding the video format, we restrict our attention to the codec-agnostic DASH. When streaming a video via DASH, the client first requests a *manifest* file [42]. The manifest specifies the quality levels at which the video can be delivered, the details of the encoding, and metadata on the actual video (e.g., name of files and locations) stored on the server.

Today, the most widely used video codec in the Internet is H.264 [8]. To encode a video using H.264 and stream it via DASH, the video data is split into *chunks*, each of which contain the same number of video frames,[3] as illustrated in Fig. 2. Often the video is encoded at different qualities (i.e., at different bitrates and/or resolutions) to enable ABR switching at the receiver or video player; in case of congestion, for instance, the video player might fetch the next chunk at a lower quality and avoid stalling the video stream. To allow fast switching, the chunk duration is commonly in the range of 1 s to 10 s.

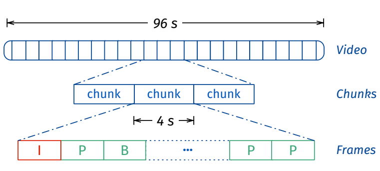


Figure 2: ***Components of a video file encoded using the H.264 codec for streaming via DASH. Video is split into equally sized chunks, each of which comprises one I-Frame and, depending on the length, several B- and P-Frames.***

The H.264 codec defines three types of *slices*: *I*-, *P*-, and *B*-*slices* [23]. We simplify, however, the H.264 specification's terminology in that we do not use the term *slice* explicitly. Each frame, in our terminology, consists of only one (*I*-, *P*-, or *B*-) slice; an *I*-Frame, for instance, refers to a frame consisting of only H.264 *I-slices*.

To help video players *instantly* start playback upon receiving a chunk (or after buffering enough chunks), each chunk needs to start with an *I*-Frame. Since *I*-Frames are *independent* frames, they can be rendered instantaneously. This lack of dependence on other frames results in the *I*-Frames being significantly large in size, and, hence, they should be used sparingly to keep the size of the video file small. To seamlessly switch between the different quality levels, we need, however, an *I*-Frame at the start of each chunk. In contrast, *P*-Frame depends on one or more previous frames, which can be of any type, and *B*-Frames depend on both previous as well as following frames. These inter-dependencies confirm a simple observation: *I*-Frames are essential and, therefore, should be well protected against loss while *P*- and *B*-Frames are less essential [2, 9, 41, 49].

4 THE QUIC FIX

The design of an optimal transport for video streaming hinges on two simple observations: *(a) I-Frames should be reliably streamed*, and *(b) It is relatively easy to recover from B- and P-Frame losses.*

We *require* an *I*-Frame to start video playback, and, hence, this frame should be reliably delivered; the playback of the remaining frames (of the concerned chunk) depend on it. Since the remaining frames encode only the *deltas* or differences with reference to the starting *I*-Frame, the loss of the *I*-Frame renders the deltas of no use, resulting in significant implications for the QoE. Regarding losses, a recent study [33] shows that the impact of *B*- and *P*-Frame losses on end-user QoE is less severe than that of *I*-Frame losses. In DASH streaming, we can quickly recover from losses after each

[3] Except, perhaps, the last chunk, which might contain fewer frames.

chunk, which is at most a few seconds long, if we transfer the *I*-Frame of each chunk *reliably*. If sufficient *I*-Frames are available (at brief-enough intervals) the impact of QoE should *not* be significant, despite losses in other frames. In practice, we can also use forward error correction (FEC) mechanisms, while carefully measuring the overheads introduced, to correct for losses in *B*- and *P*-Frames.

QUIC offers a good starting point for redesigning video transport. QUIC supports multiple streams within a single association and decouples congestion control from retransmission. In particular, QUIC's congestion control and acknowledgments operate on a per-packet basis while retransmissions are realized on a per-stream basis. This feature enables the sender to selectively retransmit or to introduce FEC on a per-stream basis, and, thus, allows, in principle, reliable and unreliable streams within the same association. Extending QUIC to support such selective delivery of the video frames over either reliable or unreliable streams, as required based on the frame type, introduces several non-trivial challenges.

• *Adding unreliable streams to QUIC.* Streams in QUIC offer a light-weight, in-order byte-stream abstraction [20]; they are individually flow-controlled and subject to congestion control. Streams, however, only offer reliable delivery. Indeed, QUIC makes, quoting the current IETF Internet draft [20], *"no specific allowance for partial reliability. Endpoints MUST be able to deliver stream data to an application as an ordered byte-stream."* This limitation makes it challenging to add support for unreliable streams and ensuring such changes are backward compatible, i.e., do not break QUIC's flow control and congestion control logic. We exploit a simple insight to solve this problem: to support unreliable streams we need to change *only* the way retransmissions are handled. More concretely, at the sender, we choose to replace retransmission of missed data with *opportunistic* transmission of the next byte range, i.e., the set of next QUIC frames. At the receiver we do not change the acknowledgment strategy: all packets, including out-of-order packets, are acknowledged using selective ACKs. The sender, hence, receives the feedback on lost packets to adjust its congestion window, but it sends *new* rather than the lost data. This approach ensures that transmission can continue without breaking flow or congestion control. We also leverage the existing re-order buffer at the receiver: an out-of-order packet is inserted into the byte-stream within this buffer unless the data has already been consumed by the application. If the application tries to consume "missing" byte-ranges the byte-stream is filled with zeros.

• *Negotiating appropriate streams.* The choice of reliable as well as unreliable QUIC streams leads to an obvious follow-up question: what data should be delivered reliably? Based on prior work on the impact of losses of different types of frames on video quality [9, 33], we deliver *I*-Frames over a reliable stream and the other kinds —*B*- and *P*-Frames—over unreliable streams. Since unreliable streams are initiated (or requested) by the client, we reuse the QUIC handshake mechanism, which includes the capabilities of the sender or receiver, to advertise and negotiate support for unreliable streams.

• *Selectively enabling reliability.* We can either provide a *meta* stream within QUIC that dictates how to selectively offer reliability—by tagging individual QUIC frames as reliable or unreliable—or implement an interface in QUIC that facilitates a client (e.g., Web browser or video player) in opening reliable as well as unreliable streams. In either case video frames are sent via the appropriate

Table 1: Video file characteristics: Resolution (Res); Bitrate, in Mbps (Br); Duration (Dur); Size, in MB; #I-Frames (#I); and #B-/P-Frames (#B/P).[4]

Video	*Res*	*Br*	*Dur*	*Size*	*#I*	*#B/P*
Big Buck Bunny	1080p	5	296.21	176	75 (1%)	7,031
Sintel	1080p	5	296.21	182	75 (1%)	7,031
Tears of Steel	1080p	5	296.21	182	75 (1%)	7,031

streams based on application-offered insights into reliability. The receiver may also use this meta-information to de-multiplex the streams and deliver the data to the application.

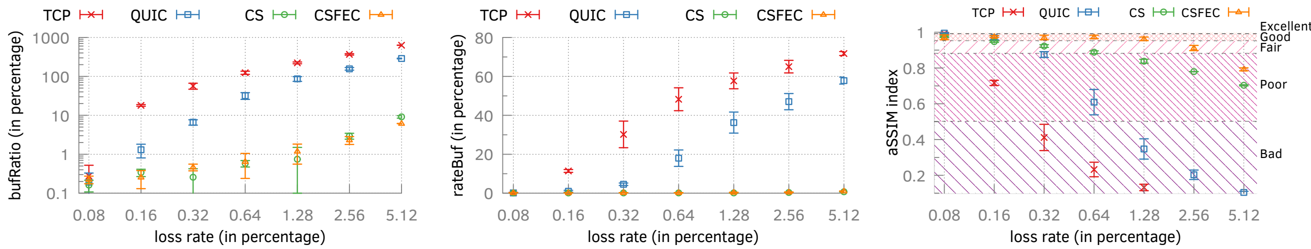
• *Synchronizing partially reliable QUIC streams.* Multiplexing the video frames over reliable and unreliable streams introduces another challenge: *How will the receiver (of a video stream) combine the frames from the different streams into the appropriate order in the playback buffer?* To this end, we can add a reliable *control stream* to signal multiplexing and demultiplexing information for the different streams to the client. The control stream helps the client to orchestrate its reads from the different streams and, thus, re-assemble the video file. Lastly, an issue that arises in case of out-of-order delivery is that the receiver will be unable to determine the end of the transmission on a stream; suppose, for instance, that the last (QUIC) frame is lost. To cope with this issue, one solution, which we choose, is to reliably transfer end-of-stream markers.

• *Tagging each video frame with reliability markers.* The sender of the video stream needs to tag each frame as reliable or unreliable to deliver it via the appropriate stream. Naturally, the sender has to parse and decode the video file, and mark each video frame to indicate whether it requires reliability. *Is it feasible for the sender to decode and tag frames?* This need to decode the video, in contrast to treating it as an opaque object, induces some overhead, but it is either a one-time cost or incurs only a small overhead. Indeed, to enable the widely used industry practice of supporting multiple resolutions as well as bitrate selections by clients, e.g., via DASH [42], video files are typically encoded *a priori* at different (predefined) resolutions. In case of live streams, videos are transcoded on demand. The tagging or reliability information (i.e., marking of frames), in either case, can be seamlessly integrated into this encoding process. The only remaining overhead is that the server must parse these reliability tags to choose the appropriate QUIC stream. We can, however, add these reliability tags to the DASH manifest files allowing clients to initiate the appropriate streams and deliver data corresponding to each without any additional overhead.

4.1 Prototype

We developed our prototype based on `quic-go` [37], specifically the version with commit ID *c852814* from Oct. 2017. The implementation of *ClipStream* comprises a *shim* layer application and our modifications to QUIC. The latter involves only *~200* lines of code. We realized unreliable streams by instructing the server (or sender) to avoid retransmissions in case of loss. We implemented a new interface for unreliable streams for allowing the clients (e.g., video player, or Web browser) to explicitly specify the required stream

[4] We shortened the videos to have the same length: The number of *I*-Frames (one per 96 frames or 4 s) and the combined number of *P*- and *B*-Frames is the same across the videos. We do not discuss control frames, they account for 0.05% of the video.



(a) Playback time spent in buffering.

(b) Relative frequency of stall events.

(c) Adjusted SSIM of reference & received frames.

Figure 3: *ClipStream (CS) and ClipStreamFEC (CSFEC) outperform TCP and QUIC across a wide range of loss values.*

type. In addition, we added a reliable *control stream* to signal multiplexing and demultiplexing information for the different streams to the client so that it can orchestrate its reads from the streams and, thus, re-assemble the video. The shim processes this control stream: on the server side, it receives *untagged* video files and marks the video frames, as reliable or not, on the fly; on the client side, it reassembles the video frames before feeding it to the video player.

When the client attempts to consume data that has not yet arrived or that has been lost, it will receive a buffer of zeros;[5] the buffer is sized to match the missing QUIC frames. We currently transmit the last byte of each frame reliably. The shim compensates for some of the loss, in unreliable streams, by using Reed-Solomon erasure coding technique (an implementation of which is available as a Go library [36]) for each of the video frames. This FEC coding scheme is well-suited for our needs as it can deal with various kinds of byte errors including bursts. We configured the FEC scheme to deliver each video frame with an overhead of 1/3 of the frame size as redundant data; more concretely, we split each video frame into 18 shards, compute 6 parity shards, and deliver the 24 shards.

5 EVALUATION

To compare and contrast the performance of *ClipStream* (and *ClipStreamFEC*, which adds FEC support) with QUIC [20] and TCP, we streamed videos from one host to another through an intermediate host, called the *shaper*. The hosts are physical machines running Debian Linux (version 9) with kernel version *4.9.91.1*. We used the `tc` utility in Linux for emulating specific link capacities and delays. We set the link capacities to 20 Mbps, which is large enough to accommodate the video streams and FEC overheads, and we sized buffers to hold 1000 packets, chiefly to accommodate QUIC's burstiness [45]. To emulate typical "last mile" latencies, we configured a 30 ms delay on the link between the client and shaper. Lastly, we captured packet traces using `tcpdump` and instrumented the server-side and client-side video streaming software for obtaining frame-level timing data.

Data Set. We selected videos (Tab. 1) that are deemed standard [25] and widely used in the literature: "Big Buck Bunny", for instance, was used in [27], "Sintel" in [52], and "Tears of Steel" in [18]. We re-encoded these videos using the `ffmpeg` utility to adhere to a frame rate of 24 fps. To simplify evaluation, the original videos were cut to be of uniform length spanning 296.21 s. The videos require a *minimum* bandwidth (B_{min}) of approximately 5 Mbps, and nearly 1% of the frames in the video file are I-Frames.

SSIM & *aSSIM*. In addition to *bufRatio* and *rateBuf*, we compute the *structural similarity (SSIM)* [47] index values to objectively estimate the stream quality. SSIM index looks at the quality of the received frames, but ignores the time at which the frames were delivered. When a video frame arrives after its deadline the client encounters a stall, significantly degrading the perceived quality of the video. SSIM is, hence, *not a good metric* for evaluating either TCP or QUIC. To capture the effect of these stalls, we compute an *adjusted* SSIM (*aSSIM*) score wherein each frame period (i.e., $1/f_T$, where f_T is the duration or time span of a frame) over the duration of the stall is assigned an SSIM index of zero. In assigning these *aSSIM* scores, we are still being generous in the evaluation of the reliable transports: we assume that despite the stalls the end user will watch the video rather than abandoning the stream, which seems to be the norm according to prior work [7, 12, 13, 17, 51]. To estimate the subjective video quality, we map the *aSSIM* values to Mean Opinion Score (MOS) values (based on [53]); the MOS values, e.g., "excellent", "good", and "bad", reflect the subjective measure of quality perceived by the user.

Controlled-Environment Trials. We streamed the video files under different loss rates, repeating 10 times for each loss rate. We computed the mean, median, and standard deviations of the three performance metrics, *bufRatio*, *rateBuf*, and *aSSIM*. We repeated the experiments with several combinations of the network parameters—bandwidth, buffer size, and delays; we omit some of the plots in the interest of space, but discuss the relevant results in the text. Since prior work shows that switching between quality levels has a negative impact on QoE [12, 18], we only use a single quality level in our experiments. The evaluations, hence, show the ability of *ClipStream* to sustain the same quality level under varying loss rates; more quality levels allow *ClipStream* more freedom (although each switch affects QoE), and we leave evaluation with multiple quality levels to future work.

Under no loss, *bufRatio* and *rateBuf* for all four transport protocols is rather small—less than 0.25%. Overall, TCP was the worst protocol for both metrics, and both *ClipStreamFEC* and *ClipStream* outperform QUIC. Per Fig. 3(a) and 3(b) we observe that the *bufRatio* and *rateBuf* for both *ClipStream* and *ClipStreamFEC* (abbreviated as CS and CSFEC, respectively, in the figures) absolutely dominate

[5] Modern video players, e.g., VLC, are capable of decoding zero-padded streams without any issue.

that of TCP and QUIC. The *rateBuf* values for both *ClipStream* and *ClipStreamFEC* are very close to 0%, with the maximum being 0.012%. These low *rateBuf*s are due to *ClipStream* streaming only a small percentage (approx. 1% by count or 12% by size) of the overall video stream reliably; the potential for stalls, hence, is rather small. *ClipStream*, hence, imposes the *bare minimum* load, even at loss rates as high as 5.12%.

The plot of *aSSIM* values as a function of loss rate, in Fig. 3(c), also shows that *ClipStream* performs better than the rest. The QoE for TCP drops very quickly from "excellent" to "bad"; even at a low loss rate of 0.32%, TCP delivers a typical *aSSIM* value that is less than 0.5, far below what is typically considered "acceptable" quality. The QoE for QUIC stays above "fair" quality for loss rates smaller than 0.32%, but drops to "bad" above 1% loss. *ClipStream* sustains "fair" quality video until 0.64% loss and does not reach "bad" quality even at 5.12% loss. *ClipStreamFEC* significantly improves upon *ClipStream*, owing to the use of FEC, delivering "good" quality till 1.28% and "fair" until 2.56%.

6 RELATED WORK

There exists a large body of prior work on video streaming. Several studies have, for instance, looked at factors affecting QoE [7, 14] and on designing optimal streaming infrastructures [21, 29]. In this section we briefly discuss only those most relevant to our work.

Adaptive bitrate schemes. Buffer-based and rate-based schemes that dynamically adapt the video bitrate [19, 22, 30, 43, 50] suffer invariably from the limitations of the underlying transport: these schemes simply operate on top of an existing transport protocol that does not discriminate between the different types of frames in the video stream. While they help in improving end-user QoE, simply porting over ABR to QUIC offers poor performance [3].

TCP variants & "tweaks". TCP variants such as TCP-RTM [28] and TL-TCP [32] either ignore retransmissions or avoid retransmitting data that have already missed the deadline. The former needs support for loss recovery to be built into the application and the latter requires application's cooperation to obtain the deadlines: both complicate application design, making deployment impractical, if not impossible. Brosh et al. [5] suggest optimizations to make TCP more friendly for delivering real-time media. In a similar vein, Goel et al. [15] tune TCP's send buffer for mitigating delays. While these optimizations are important, they will be even more beneficial when applied selectively to only the portion of data that requires reliability in the first place.

Partial reliability. McQuistin et al. [31] propose a novel TCP variant that uses retransmissions to deliver new data, instead of the lost data. The idea of using the retransmissions to send new data alleviates some but not all of the overhead; *B*- and *P*-Frames that have not missed their deadlines will still be retransmitted. [9] explores the effect of selective reliability for streaming MPEG-4 video via RTP, necessitating substantial changes to the network stack. *ClipStream* requires minimal changes and can be deployed incrementally.

Error-correction schemes. Kim et al. [24] propose CTCP, which codes data in an application-agnostic manner, to improve performance in lossy channels. CTCP's indiscriminate coding of all video frames by a significant number of users might, under certain conditions, overwhelm the network capacity. *ClipStream* can benefit, however, from using CTCP's adaptive coding scheme for delivering *B*- and *P*-Frames.

7 SUMMARY & OUTLOOK

The increasing adoption of QUIC on the server side (e.g., CDNs) as well as the client side (e.g., Google Chrome browser) offers us the unprecedented opportunity to rethink about an ideal transport protocol for video streaming. We show that such an ideal transport, exploiting partial reliability, can be realized simply through the addition of unreliable streams to QUIC. We already submitted a draft to the QUIC Working Group [46] to add support for unreliable streams, and plan on following up with insights and observations from our experience of implementing unreliable streams in QUIC and leveraging it in *ClipStream* for use in video streaming. While our preliminary evaluation of the selective use of reliability for video streaming shows our approach to be better than TCP and QUIC, we envision conducting real-world experiments (i.e., over the Internet) and comparing our approach with ABR schemes.

REFERENCES

[1] Akamai Technologies. Community Blog, FAQ: QUIC Native Platform Support for Media Delivery Products. https://tinyurl.com/yab23e4f, March 2018.
[2] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42(6), Nov 1996.
[3] D. Bhat, A. Rizk, and M. Zink. Not So QUIC: A Performance Study of DASH over QUIC. In *Proceedings of NOSSDAV '17*, 2017.
[4] S. Bouzas. Why the Meteoric Rise of Google QUIC is Worrying Mobile Operators. https://tinyurl.com/y927mctq, January 2018.
[5] E. Brosh, S. A. Baset, D. Rubenstein, and H. Schulzrinne. The Delay-friendliness of TCP. In *Proceedings of SIGMETRICS '08*, 2008.
[6] Cisco. The Zettabyte Era: Trends and Analysis. https://bit.ly/2h3jXbJ, June 2017.
[7] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proceedings of ACM SIGCOMM '11*, 2011.
[8] encoding.com. Global Media Format Report 2018. https://bit.ly/2HXfSxn, 2018.
[9] N. Feamster and H. Balakrishnan. Packet Loss Recovery for Streaming Video. In *12th International Packet Video Workshop*, April 2002.
[10] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan. An Internet-Wide Analysis of Traffic Policing. In *Proceedings of ACM SIGCOMM '16*, 2016.
[11] S. Fouladi, J. Emmons, E. Orbay, C. Wu, R. S. Wahby, and K. Winstein. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *Proceedings of USENIX NSDI '18*, 2018.
[12] M. . Garcia, F. D. Simone, S. Tavakoli, N. Staelens, S. Egger, K. Brunnström, and A. Raake. Quality of experience and HTTP adaptive streaming: A review of subjective studies. In *International Workshop on Quality of Multimedia Experience (QoMEX)*, Sept 2014.
[13] D. Ghadiyaram, A. C. Bovik, H. Yeganeh, R. Kordasiewicz, and M. Gallant. Study of the effects of stalling events on the quality of experience of mobile streaming videos. In *Signal and Information Processing (GlobalSIP), 2014 IEEE Global Conference on*. IEEE, 2014.
[14] M. Ghasemi, P. Kanuparthy, A. Mansy, T. Benson, and J. Rexford. Performance Characterization of a Commercial Video Streaming Service. In *Proceedings of ACM IMC '16*, 2016.
[15] A. Goel, C. Krasic, and J. Walpole. Low-latency Adaptive Streaming over TCP. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(3), Sept. 2008.
[16] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Internet-draft, Transport Area Working Group, January 2016.
[17] T. Hossfeld, S. Egger, R. Schatz, M. Fiedler, K. Masuch, and C. Lorentzen. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *International Workshop on Quality of Multimedia Experience*, pages 1–6, July 2012.
[18] T. Hoßfeld, M. Seufert, C. Sieber, and T. Zinner. Assessing effect sizes of influence factors towards a QoE model for HTTP adaptive streaming. In *Sixth International Workshop on Quality of Multimedia Experience (QoMEX)*, Sept 2014.
[19] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service.

In *Proceedings of ACM SICOMM '14*, 2014.

[20] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-draft, IETF, May 2018.

[21] J. Jiang, V. Sekar, I. Stoica, and H. Zhang. Shedding Light on the Structure of Internet Video Quality Problems in the Wild. In *Proceedings of CoNEXT '13*, 2013.

[22] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proceedings of CoNEXT '12*, 2012.

[23] B. Juurlink, M. Alvarez-Mesa, C. C. Chi, A. Azevedo, C. Meenderinck, and A. Ramirez. *Understanding the Application: An Overview of the H.264 Standard*. SpringerBriefs in Computer Science, 2012.

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

[25] C. Kreuzberger, D. Posch, and H. Hellwagner. A Scalable Video Coding Dataset and Toolchain for Dynamic Adaptive Streaming over HTTP. In *Proceedings of ACM MMSys '15*, 2015.

[26] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of ACM SIGCOMM '17*, 2017.

[27] S. Lederer, C. Müller, and C. Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of ACM MMSys '12*, 2012.

[28] S. Liang and D. Cheriton. TCP-RTM: Using TCP for real time multimedia applications. In *International Conference on Network Protocols*, 2002.

[29] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *Proceedings of ACM SIGCOMM '12*, 2012.

[30] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of ACM SIGCOMM '17*, 2017.

[31] S. McQuistin, C. Perkins, and M. Fayed. TCP Goes to Hollywood. In *Proceedings of NOSSDAV '16*, 2016.

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

[33] P. Orosz, T. Skopkó, and P. Varga. Towards estimating video QoE based on frame loss statistics of the video streams. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015.

[34] R. Pantos and W. May. HTTP Live Streaming. RFC 8216, Aug. 2017.

[35] M. Ponec. QUIC @ Akamai, IETF 96. https://tinyurl.com/y7ysw8js, July 2016.

[36] K. Post. Reed-Solomon Erasure Coding in Go. https://tinyurl.com/oszpooz, November 2017.

[37] QUIC-go. A QUIC implementation in pure Go. https://tinyurl.com/hp9r3sm, November 2017.

[38] D. Robinson and D. De Vleeschauwer. TCP: From Data to Streaming Video. https://goo.gl/SsZcbr, March 2011.

[39] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld. A First Look at QUIC in the Wild. In *Passive and Active Measurement*, 2018.

[40] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE J.Sel. A. Commun.*, 13(7), Sept. 2006.

[41] K. Shimamura, Y. Hayashi, and F. Kishino. Variable-Bit-Rate Coding Capable Of Compensating For Packet Loss. In *Proceedings Vol. 1001, Visual Communications and Image Processing '88: Third in a Series*, 1988.

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

[43] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of ACM SIGCOMM '16*, 2016.

[44] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband Internet Performance: A View from the Gateway. In *Proceedings of ACM SIGCOMM '11*, August 2011.

[45] I. Swett. QUIC: Congestion Control and Loss Recovery. https://tinyurl.com/y9szq6k8, July 2015.

[46] P. S. Tiesel, M. Palmer, B. Chandrasekaran, A. Feldmann, and J. Ott. Considerations for Unreliable Streams in QUIC. Internet-draft, IETF, Oct. 2017.

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

[48] M. Yakan and A. Jayaprakash. Introducing QUIC for Web Content. https://tinyurl.com/y93uqryr, October 2018.

[49] X. Yang, C. Zhu, Z. G. Li, X. Lin, and N. Ling. An unequal packet loss resilience scheme for video over the Internet. *IEEE Transactions on Multimedia*, 7(4), Aug 2005.

[50] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of ACM SIGCOMM '15*, 2015.

[51] T. Zhao, Q. Liu, and C. W. Chen. QoE in video transmission: A user experience-driven strategy. *IEEE Communications Surveys & Tutorials*, 19(1), 2016.

[52] W. Zia, T. Stockhammer, and K. Walker. Demonstrating ATSC ROUTE-DASH Delivery. In *Proceedings of MMSys '16*, 2016.

[53] T. Zinner, O. Hohlfeld, O. Abboud, and T. Hossfeld. Impact of frame rate and resolution on objective QoE metrics. In *Second International Workshop on Quality of Multimedia Experience (QoMEX)*, June 2010.