End-to-End Transport for Video QoE Fairness

Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki
Mehrdad Khani, Prateesh Goyal, Mohammad Alizadeh
MIT Computer Science and Artificial Intelligence Laboratory
{vikramn,vibhaa,addanki,khani,prateesh,alizadeh}@csail.mit.edu

Abstract

The growth of video traffic makes it increasingly likely that multiple clients share a bottleneck link, giving video content providers an opportunity to optimize the experience of multiple users jointly. But today's transport protocols are oblivious to video streaming applications and provide only connection-level fairness. We design and build Minerva, the first end-to-end transport protocol for multi-user video streaming. Minerva uses information about the player state and video characteristics to adjust its congestion control behavior to optimize for *QoE fairness*. Minerva clients receive no explicit information about other video clients, yet when multiple of them share a bottleneck link, their rates converge to a bandwidth allocation that maximizes QoE fairness. At the same time, Minerva videos occupy only their fair share of the bottleneck link bandwidth, competing fairly with existing TCP traffic. We implement Minerva on an industry standard video player and server and show that, compared to Cubic and BBR, 15-32% of the videos using Minerva experience an improvement in viewing experience equivalent to a jump in resolution from 720p to 1080p. Additionally, in a scenario with dynamic video arrivals and departures, Minerva reduces rebuffering time by an average of 47%.

CCS Concepts

• **Networks → Application layer protocols**;

Keywords

video streaming, quality of experience, DASH, rate control

ACM Reference Format:
Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, Mohammad Alizadeh. 2019. End-to-End Transport for Video QoE Fairness. In *SIGCOMM '19: 2019 Conference of the ACM Special Interest Group on Data Communication, August 19–23, 2019, Beijing, China.* ACM, Beijing, China, 16 pages. https://doi.org/10.1145/3341302.3342077

1 Introduction

HTTP-based video streaming traffic has grown rapidly over the past decade. Video traffic accounted for 75% of all Internet traffic in 2017, and is expected to rise to 82% by 2022 [9]. With the prevalence of video streaming, a significant body of research over the past decade has developed robust adaptive bitrate (ABR) algorithms and transport protocols to optimize video quality of experience (QoE) [3, 7, 13, 18, 27, 40, 44]. The majority of this research focuses on QoE for a *single* user in isolation. However, due to the fast-paced growth of video traffic, it is increasingly likely that multiple video streaming clients will share a bottleneck link. For example, a home or campus WiFi network may serve laptops, TVs, phones, and tablets, all streaming video simultaneously. In particular, the median household in the U.S. contains five streaming-capable devices, while one-fifth of households contain at least ten [36].

Video content providers today are beholden to the bandwidth decisions made by existing congestion control algorithms: all widely-used protocols today (e.g. Reno [4] and Cubic [16]) aim to achieve *connection-level fairness*, giving competing flows an equal share of the link's capacity on average. Therefore, content providers running these protocols can only optimize for user viewing experience in isolation, e.g. by deploying ABR algorithms. They miss the bigger picture: allocating bandwidth carefully between video clients can optimize the overall utility of the system. The opportunity to optimize viewing experience collectively is particularly relevant for large content providers. Netflix, for example, occupies 35% of total Internet traffic at peak times and may therefore control a significant fraction of the traffic on any given bottleneck link [38].

Specifically, there are two problems with standard transport protocols that split bandwidth evenly between video streams. First, they are blind to user experience. Users with the same bandwidth may be watching a variety of video genres in a range of viewing conditions (e.g. screen size), thereby experiencing significant differences in viewing quality. Today's transport protocols are unaware of these differences, and cannot allocate bandwidth in a manner that optimizes QoE.

Second, existing congestion protocols are blind to the dynamic state of the video client, such as the playback buffer size, that influences the viewer's experience. For example, knowing that a client's video buffer is about to run out would allow the transport to temporarily send at a higher rate to build up the buffer, lowering the likelihood of rebuffering. Protocols like Cubic, however, ignore player state and prevent clients from trading bandwidth with each other.

We design and implement Minerva, an *end-to-end transport protocol for multi-user video streaming*. Minerva clients dynamically and independently adjust their rates to optimize for *QoE fairness*, a measure of how similar the viewing experience is for different users. Minerva clients require no explicit information about other competing video clients, yet when multiple of them share a bottleneck link, their rates converge to a bandwidth allocation that maximizes QoE fairness. Crucially, throughout this process, Minerva clients together occupy only their fair share of the link bandwidth, which ensures fairness when competing with non-Minerva flows (including other video streams). Since clients operate independently, Minerva is easy to deploy, requiring changes to only the client and server endpoints but not to the network. A content provider can deploy Minerva today to optimize QoE fairness for its users, without buy in from other stakeholders.

Central to Minerva are three ideas. First is a technique for deriving *utility functions* that capture the relationship between network bandwidth and quality of experience. Defining these functions is challenging because standard video streaming QoE metrics are expressed in terms of video bitrates, rebuffering time,

SIGCOMM '19, August 19–23, 2019, Beijing, China

ACM ISBN 978-1-4503-5956-6/19/08…$15.00
https://doi.org/10.1145/3341302.3342077

smoothness, and other application-level metrics, but not network link rates. We develop an approach that reconciles the two and exposes the relationship between them.

Second is a new distributed algorithm for achieving fairness between clients using these utility functions. Our solution supports general notions of fairness, such as max-min fairness and proportional fairness [22]. Each client computes a dynamic *weight* for its video. The transport layer then achieves a bandwidth allocation for each video that is proportional to its weight. A client's weight changes throughout the course of the video, based on network conditions, the video's utility function, and client state, but independently of other clients. Collectively, the rate allocation determined by these weights converges to the optimal allocation for QoE-fairness.

Third is a weight normalization technique used by Minerva to compete fairly with standard TCP. This step allows clients to converge to a set of rates that simultaneously achieves QoE fairness while also ensuring fairness with non-Minerva flows on average.

We implement Minerva on top of QUIC [23] and adapt an industry-standard video player to make its application state available to the transport. For deployability, our implementation uses Cubic [16] as its underlying congestion control algorithm for achieving weighted bandwidth allocation. We run Minerva on a diverse set of network conditions and a large corpus of videos and report the following:

(1) Compared to existing video streaming systems running Cubic and BBR, Minerva improves the viewing quality of between 15-32% of the videos in the corpus by an amount equivalent to a bump in resolution from 720p to 1080p.
(2) By allocating bandwidth to videos at risk of rebuffering, Minerva is able to reduce total rebuffering time by 47% on average in a scenario with dynamic video arrivals and departures.
(3) We find that Minerva competes fairly with videos and emulated wide area traffic running Cubic, occupying within 4% of its true fair share of the bandwidth.
(4) Minerva scales well to many clients, different video quality metrics, and different notions of fairness.
(5) We run Minerva over a real, residential network and find that its benefits translate well into the wild.

This work does not pose any ethical concerns.

2 Motivation

Central to Minerva is the realization that connection-level fairness, where competing flows get an equal share of link bandwidth, is ill-suited to the goals of video providers like Netflix and YouTube. In particular, connection-level fairness has two undesirable effects from a video provider's point of view.

First, it is oblivious to the bandwidth requirements of different users. Different videos require different amounts of bandwidth to achieve the same viewing quality. For example, a viewer will have a better experience streaming a given video on a smartphone at 1 Mbit/s than a large 4K TV at 1 Mbit/s [26]. Further, discrepancies in viewing experience extend beyond differences in screen size. User studies show that the perceived quality of a video is influenced by its content as well, e.g., its genre or degree of motion. Fig. 1 plots the average video quality, as rated by viewers, at several bitrates for a diverse set[1] of videos [24]. A client watching the video "V3", for example, would require a higher bandwidth than a client watching "V19" to sustain the same viewing quality. If both received the same bandwidth, "V3" would likely appear blurrier and less visually satisfying. However, current transport protocols that provide

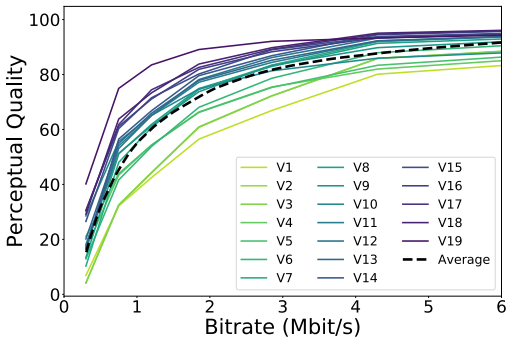
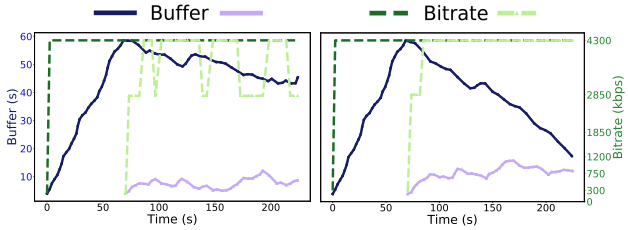


Figure 1: Perceptual quality for a diverse set of videos based on a Netflix user study [24]. Also shown is the "average" perceptual quality (dotted), which is Minerva's normalization function (§5.4). For context, the average qualities at 720p and 1080p are 82.25 and 89.8.



(a) Cubic **(b) Minerva**

Figure 2: To demonstrate buffer pooling, we start one video client, allow it to build up a large buffer, and then introduce a second video after 70 seconds. (a) With Cubic, the first video maintains a large buffer, causing the second video to sacrifice its quality to avoid rebuffering. (b) Minerva trades the large buffer of the first video to fetch higher quality chunks for the second.

connection-level fairness are unaware of these differences in videos, and thus they relegate some viewers to a worse viewing experience.

Second, protocols that split bandwidth evenly between connections are blind to the state of the video client, such as the playback buffer size, so they cannot react to application-level warning signs. In particular, a video client with a low buffer, e.g. at video startup, has a higher likelihood of rebuffering, which prevents the ABR algorithm from requesting high-quality chunks. Consider such a video sharing the network with other clients that have large buffers. Dynamically shifting bandwidth to the resource-constrained video in question would allow it to quickly build up its buffer, mitigating the adverse effects of rebuffering without sacrificing its own viewing experience or that of the other clients. This dynamic reallocation effectively creates a shared *buffer pool*, letting clients with small video buffers tap into the large buffers of other video clients, as illustrated in Fig. 2. Equal bandwidth sharing, on the other hand, isolates clients and prevents them from pooling their resources.

While most Internet traffic expects bandwidth to be shared equally between competing flows, connection-level fairness *between videos* is not ideal: providers should be able to allocate bandwidth between their videos in a way that optimizes their viewers' experience, provided they play fairly with other traffic. In fact, providers have economic incentives to consider video quality when delivering content: to boost user engagement, improving viewing quality at lower bitrates is often more important than at higher bitrates.[2] Connection-level fairness, being blind to video quality,

[1] For a description of each video, see Appendix A.

[2] Corroborated in private communication with a large content provider.

is therefore ill-suited to a video provider's needs. Instead, having the ability to shift bandwidth based on video quality considerations would allow them to directly optimize relevant business objectives.

Many video providers have already abandoned connection-level fairness. Netflix and YouTube typically use three parallel TCP connections to download video chunks, effectively giving their videos larger bandwidth shares and preventing poor viewing experiences [28, 32, 41]. Additionally, Netflix uses a larger number of connections at video startup, when buffer is low, to avoid potential rebuffering events early in the video [41]. These remedies are ad-hoc, coarse (adding or removing only an integral number of connections), heavyweight (incurring startup time for a new connection), and make no effort to be fair to competing web traffic.

Minerva offers video providers a more principled solution. It dynamically allocates bandwidth between videos in a manner that (a) allows fine-grained control over a video's rate share, (b) responds quickly to low buffers and (c) competes fairly with non-video traffic. Importantly, each provider has full control over their use of Minerva. They may deploy Minerva on their client and server endpoints, independently of other providers, and without any change to the network.

Minerva is able to ameliorate the drawbacks of connection-level fairness by dynamically modifying a video's rate allocation to optimize a *QoE fairness* metric. Minerva uses a standard definition of QoE fairness (max-min QoE fairness) that aims to improve the video quality for clients with the worst QoE (§4.1). However, Minerva is flexible and can optimize for a variety of QoE fairness definitions (§8.6). It is beyond the scope of this work to determine the best fairness metric for video providers.

3 Related Work

Minerva is informed by a large body of work focused on improving user experience while streaming video.

Single-user streaming. In single-user video streaming, each video optimizes only within the bandwidth allocation prescribed by the underlying transport. The underlying bandwidth share between videos is not modified in response to video-level metrics, such as perceptual quality or playback buffer. Improvements to single-user streaming include ABR algorithms, which use bandwidth measurements to choose encodings that will improve perceptual quality and minimize rebuffering. State of the art algorithms are typically also aware of client state and may optimize for QoE either explicitly [3, 44] or implicitly, e.g. via a neural network [27].

Further single-user streaming improvements include techniques that correct for the shortcomings of DASH [39] to achieve fairness across multiple users, by improving bandwidth estimation [25] or avoiding idle periods during chunk downloads [45]. Other schemes manage the frequency at which chunks are requested [20] or model the choices of other clients using a game-theoretic framework [6]. As a result, these methods improve utilization, perceptual smoothness, and fairness among competing videos. However, they improve only connection-level fairness and ignore the perceptual quality differences between videos, so they cannot optimize QoE fairness. Furthermore, they are still ultimately bound by the equal-bandwidth allocation prescribed by the underlying transport.

Transport Protocols. Alternate transport protocols may also improve a viewer's experience. PCC has been shown to make better use of available link bandwidth and thus improve a user's QoE [12]. However, PCC is a general purpose transport and is not aware of video quality metrics. Salsify [13] designs real-time video conferencing applications that are network aware; the video encoder uses estimates of available bandwidth to choose its target bitrate. However, Salsify targets the real-time conferencing use case, while Minerva targets DASH-based video-on-demand.

Centralized multi-user streaming. Existing systems that optimize QoE fairness over multiple users only consider *centralized* solutions [8, 43]. They require a controller on the bottleneck link with access to all incident video flows. This controller computes the bandwidth allocation that optimizes QoE fairness and enforces it at the link; clients are then only responsible for making bitrate decisions via traditional ABR algorithms. However, this requires a network controller to run at every bottleneck link and thus presents a high barrier to deployment.

Video quality metrics. Another line of work has focused on defining metrics to better capture user preferences. Content-agnostic schemes [26] use screen size and resolution as predictors of viewing quality. Other efforts [29, 42], including Netflix's VMAF metric [24], use content-specific features to compute scores that better align with actual user preferences. Minerva can support any metric in its definition of QoE.

Decentralized schemes. Minerva clients use a distributed rate update algorithm to converge to QoE fairness. A popular framework for decentralizing transport protocols that optimize a fairness objective is Network Utility Maximization (NUM) [22]. NUM uses link "prices" that reflect congestion to solve the utility maximization problem by computing rates based on prices at each sender; repeated iterations of the price updates and rate computations converges to an allocation that optimizes the fairness objective. In practice, NUM-based rate control schemes can be difficult to stabilize. Another approach solves NUM problems by dynamically deciding a *weight* for each sender, and then using a rate control scheme to achieve rates proportional to those weights. This avoids over-and under-utilization of links [31]. Minerva implements this second approach and is therefore able to simultaneously achieve full link utilization and QoE fairness.

4 Problem Statement

4.1 QoE Fairness

Minerva optimizes for a standard definition of QoE found in the video streaming literature [44]. QoE is defined for the kth chunk c_k based on the previous chunk and the time R_k spent rebuffering prior to watching the chunk:

$$\text{QoE}(c_k, R_k, c_{k-1}) = P(c_k) - \beta R_k - \gamma \|P(c_k) - P(c_{k-1})\|. \quad (1)$$

$P(e)$ denotes the quality gained from watching a chunk at bitrate e, which we term Perceptual Quality (PQ), β is a penalty per second of rebuffering, and γ penalizes changes in bitrate between adjacent chunks (*smoothness*). In general, PQ may vary between videos and clients, based on parameters such as the client's screen size, screen resolution, and viewing distance, as well as the video content and genre. It also typically varies by chunk over the course of a single video: chunks with the same bitrate may have different PQ levels depending on how the content in those chunks are encoded. Minerva can use any definition of PQ that meets the loose requirements in Appendix B, e.g., it is sufficient that $P(e)$ be increasing and concave.[3]

Suppose N clients share a bottleneck for a time period T, during which each client i watches n_i chunks and experiences a total (summed over all chunks) QoE of QoE_i. Our primary goal is *max-min fairness* of the per-chunk average QoE between clients, i.e., to maximize $\min_i \frac{QoE_i}{n_i}$.

[3] This property, standard for utility functions, captures the notion that clients experience diminishing marginal utility at successively higher encodings.

Max-min QoE fairness, a standard notion of fairness, captures the idea that providers may reasonably seek to improve the experience of their worst-performing clients. However, achieving max-min QoE fairness may require very different rate allocations for two videos with substantially different perceptual qualities. Providers who are unhappy with such a rate allocation have two alternatives. First, Minerva allows optimizing for max-min QoE fairness *subject to* the constraint that no video achieves a bandwidth that differs from its fair share by more than a given factor μ. Second, it also supports different definitions of fairness, such as proportional fairness. Both approaches are discussed further in §5.5.

4.2 Goals

Minerva's overarching motivation is to provide a practical mechanism to achieve QoE fairness among competing video flows. In particular, we desire that Minerva

(1) be an *end-to-end* scheme. Deploying Minerva should only require modifications to endpoints, without an external controller or changes to the network.
(2) improve *QoE fairness*. N video flows using Minerva should converge to a bandwidth allocation that maximizes QoE fairness between those N videos. The clients do not know N or any information about other video flows.
(3) ensure *fairness with non-Minerva flows*. The total throughput of N Minerva videos should equal that of N Cubic flows to not adversely impact competing traffic. We design Minerva to compete fairly against Cubic, since it is a widely deployed scheme, but our approach extends to other protocols as well.
(4) be *ABR agnostic*. The ABR algorithm should be abstracted away, so that Minerva can work with any ABR algorithm. Minerva may have access to the ABR algorithm, but can only use it as a black box.

These properties offer benefits to large video providers, like Netflix, to use Minerva. Netflix videos constitute 35% of total Internet traffic, making them likely to share bottleneck links [38]. This large bandwidth footprint motivates using Minerva to improve collective experience for Netflix viewers. Additionally, since Minerva video streams operate independently, it can be deployed without knowing which videos share bottlenecks or where those bottlenecks occur. Further, even a single provider can benefit from Minerva, independently of whether other providers deploy Minerva as well, since Minerva videos achieve a fair total throughput share with other competing traffic.

5 Design

5.1 Approach

Minerva repeatedly updates clients' download rates in a way that increases $\min_i \frac{QoE_i}{n_i}$. However, making rate control decisions in the context of dynamic video streaming is a tricky task. The definition of QoE does not directly depend on the client's download rate, so it is not immediately apparent how to optimize QoE fairness by simply changing the bandwidth allocation. In fact, the effects of rate changes on QoE may not manifest themselves immediately; for example, increasing the download rate will not change the bitrate of the chunk currently being fetched. However, it may have an indirect impact: if a client's bandwidth improves, its ABR algorithm may measure a higher throughput and choose higher qualities for future chunks.

To solve the above optimization problem with a rate control algorithm, Minerva must recast it in a form that depends only on network link rates. This is made difficult by the fact that the QoE is also a function of other parameters, such as buffer level and bitrate. Therefore, Minerva first **formulates** a *bandwidth utility* function for each client that decomposes the optimization problem into a function of only its download rate.

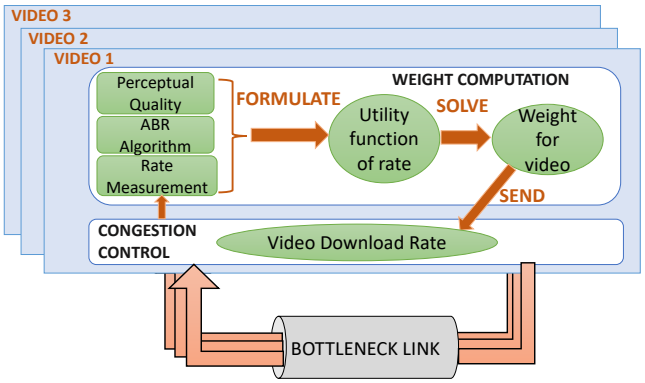


Figure 3: Minerva's high-level control flow. Clients run Minerva's formulate-solve-send process independently and receive feedback only through the rate they measure on the next iteration.

Following formulation, the competing videos must **solve** the QoE fairness maximization problem in a decentralized manner. Since Minerva cannot change the available capacity, it can control only the relative allocation of bandwidth to each video. It determines this allocation by assigning each video a weight based on the solution of the bandwidth utility optimization, using a custom decentralized algorithm. Then in the **send** step, Minerva utilizes existing congestion control algorithms to achieve a bandwidth allocation for each video proportional to its weight, while also fully utilizing the link capacity. Fig. 3 illustrates Minerva's high level control flow.

Each video runs Minerva's three-step formulate-solve-send process once every T milliseconds, where T is tunable parameter. §5.2 details the basic operation of these three steps, including the form of the bandwidth utility functions, the method by which weights are determined, and how existing congestion control algorithms are adapted to achieve those weights. §5.3 discusses a key optimization to improve performance, §5.4 explains how Minerva achieves fairness with TCP, and §5.6 outlines how Minerva can be used on top of a variety of existing congestion control algorithms.

5.2 Basic Minerva

Formulating the bandwidth utility. Given the definition of QoE (Eq. 1), we aim to construct a bandwidth utility function $U(r)$ that is a function of only the client's download rate. $U(r)$ should capture the QoE the client expects to achieve given a current bandwidth of r. In Minerva's basic design, $U(r)$ assumes that the client is able to maintain a bandwidth of r for the rest of the video.

Buffer dynamics provide insight into what this QoE value will be. After downloading a single chunk of duration D and size C, the client loses $\frac{C}{r}$ seconds of download time but gains D seconds from the chunk's addition. In the client's steady state, the average change in the buffer level is close to 0 over long time periods (several chunks). During this time, the bitrate averages to approximately r; if r lies between two available bitrates e_i and e_{i+1}, the client switches between those bitrates such that its average bitrate per chunk is r. Therefore, the expected per-chunk QoE is a linear interpolation of the PQ function between $P(e_i)$ and $P(e_{i+1})$ at r. This observation

yields a formalization of $U(r)$:

$$U(r) = \begin{cases} P(e_i) & \text{if } r = e_i \\ \texttt{Interpolate}(P,r) & \text{if } e_i < r < e_{i+1} \end{cases} \quad (2)$$

assuming the video is available to stream at discrete bitrates $\{e_i\}$. r is the client's average download rate, measured over the past T milliseconds.

This definition of the bandwidth utility only considers the PQ component of the QoE; it does not take into account client state, such as buffer level, nor does it factor in penalties for rebuffering or smoothness. We present a more sophisticated bandwidth utility function in §5.3 that does both.

Solving $U(r)$. Given a bandwidth utility function $U_i(r_i)$, which is an estimate of a client's expected QoE, the QoE fairness optimization problem now becomes:

$$\text{maximize } \min_i U_i(r_i)$$

Each client must find a *weight* w_i, such that the set of all client weights determines a relative bandwidth allocation that optimizes QoE fairness. Assuming the U_i are continuous, the solution to the optimization problem occurs when the U_i are all equal and none can be made larger. Minerva relies on the Send step to ensure that clients achieve full link utilization. Therefore, this step focuses only on distributing the link bandwidth to achieve equality between the bandwidth utility functions.

Reaching equality is complicated by the fact that Minerva is completely decentralized: each client is not aware of the utility functions of the others. However, Minerva uses a decentralized algorithm to achieve max-min utility fairness on a bottleneck link, where each video makes decisions based only on its own state. At every timestep, the weight w_i for the next interval is

$$w_i = \frac{r_i}{U_i(r_i)} \quad (3)$$

This iterative update rule has two key properties. First, provided a fixed capacity link, the optimal rates $\{r_i^*\}$ are a fixed point. To see why, notice that at the optimal rates, $\{U_i(r_i^*)\}$ are equal for all clients. Therefore, if the clients are downloading at the optimal rates, after one weight update, the ratio of the new rates will be

$$\frac{r_i'}{r_j'} = \frac{r_i^*/U_i(r_i^*)}{r_j^*/U_j(r_j^*)} = \frac{r_i^*}{r_j^*}$$

Therefore, the rate ratios remain identical. Minerva assumes the link capacity stays constant in the short term, so identical rate ratios will result in identical rates.

Second, each iteration of (3) moves the rates closer towards their optimal values. Consider two clients with ratio $\rho = \frac{r_1}{r_2}$. If $r_1 < r_1^*$ and $r_2 > r_2^*$, then $U_1(r_1) < U_2(r_2)$. Correspondingly, the ratio in the next iteration is $\rho' = \rho \cdot \frac{U_2(r_2)}{U_1(r_1)} > \rho$. Client 1's share of bandwidth will then increase, and Client 2's will decrease, moving the clients closer to $U_1(r_1) = U_2(r_2)$. See Appendix B for a full proof of convergence.

Setting rates. After the Solve step, clients have weights w_i that they must use to achieve rates proportional to w_i, while still fully utilizing the link capacity. A straightforward method to realize these rates is to emulate w_i connections using a congestion control algorithm that achieves per-flow fairness. This makes Minerva capable of building on top of any transport protocol that accepts such a weight. §5.6 discusses examples of such protocols.

5.3 A Client-Aware Utility Function

The basic bandwidth utility function takes into account only the current download rate and the PQ function. However, while this model of QoE is approximately accurate over long time scales, it is overly simplistic for several reasons. First, it is blind to client state. The client holds valuable information, such as buffer level, that influences its future QoE. For example, having a larger buffer may allow the client to receive less bandwidth in the short term without reducing its encoding level. A more sophisticated utility function should be able to capture the positive value of buffer on a client's expected QoE.

Second, it accounts for only the PQ term in the QoE and ignores the rebuffering and smoothness terms. The basic utility function does not understand that a client with a lower bandwidth or low buffer has a higher likelihood of rebuffering. Additionally, though it expects the client to fetch encodings that average to r, it does not factor in the smoothness penalty between these encodings.

Third, it only looks at future QoE, while ignoring the past. A client that rebuffers early on will afterwards be treated identically to a client that never rebuffered. In order to achieve max-min QoE fairness, the rebuffering client should be compensated with a higher bitrate. Only a utility function that is aware of the QoE of previous chunks can hope to have this capability.

Recognizing the limitations of the basic PQ-aware utility function, we construct a *client-aware* utility function that addresses all three limitations. This new utility function directly estimates the per-chunk QoE using information from past chunks, the current chunk being fetched, and predicted future chunks:

$$U(r) = \frac{\varphi_1(\text{Past QoE}) + \varphi_2(\text{QoE from current chunk}) + V_h(r,b,c_i)}{1+\varphi_1+\varphi_2}$$

where φ_1, φ_2 are positive weights that determines relative importance of the three terms. The QoE of the current chunk is estimated by using the current rate r to determine if the video stream will rebuffer. Suppose that a client with buffer level b is downloading a chunk c_i and has c bytes left to download, Minerva computes the expected rebuffering time $R = [c/r - b]_+$ and estimates the QoE of the current chunk as $QoE(c_i,R,c_{i-1})$, where QoE is defined as in Eq. 1.

One of Minerva's key ideas is V_h, a *value function* computing the expected per-chunk QoE over the next h chunks, where h is a horizon that can be set as desired. It captures the notion that the QoE a client will achieve depends heavily on the ABR algorithm that decides the encodings. In order to accurately estimate future QoE, clients simulate the ABR algorithm over the next h chunks as follows:

```
r ← Measured download rate
S ← client state
do h times:
  e = ABR(S)
  S, rebuf ← client state and
       rebuffer time after chunk
       e is downloaded at rate r
```

The value function uses the chunk encodings and rebuffer times at every iteration to evaluate the expected QoE. Clients need not modify the ABR implementation; they can simply use it as a black box when computing the value function.

Fig. 4 shows the value function for MPC [44] with $h = 5$. Note the dependence on rate and buffer. In particular, a large buffer has diminishing marginal returns. The jump in value due to increasing the buffer from 0s to 4s approximately equals the bump due to increasing the buffer from 4s to 20s. This aligns with intuition: clients with a low buffer are at a higher risk of rebuffing and therefore place a high value on each second of buffer.

In general, the value function depends heavily on the underlying ABR algorithm used to compute it, which has two implications. First,

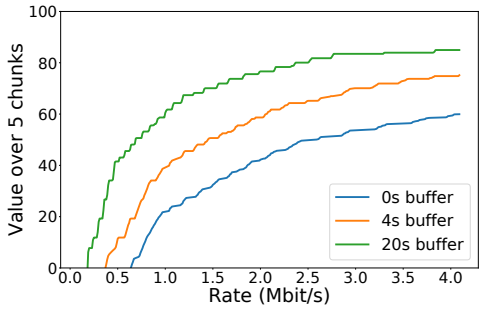


Figure 4: The value function simulated with MPC for a range of buffers and rates. Higher rates and buffers yield a higher value, and larger buffers have diminishing marginal utility.

it may be expensive to compute. Since the value function for a video does not change across sessions, Minerva precomputes the value function before streaming begins (§6). Second, the value function may not exactly satisfy the convergence conditions of Minerva's rate update rule. Appendix C discusses methods to overcome this.

5.4 Normalization: Fairness with TCP

§5.2 describes how Minerva flows competing with each other converge to the correct bandwidth allocation. However, in the wild, they will have to also compete with, and be fair to, other TCP flows (e.g., Cubic), including Minerva flows from different video providers. In this section, we describe how Minerva flows achieve fairness with TCP while optimizing the relative bandwidth allocation between themselves.

As discussed previously, flows updating their weights using Eq. 3 converge to a steady state where the $U_i(r_i)$ are equal. This convergence is not specific to the exact functional form of U_i; in fact, we can replace $U_i(r_i)$ with $f(U_i(r_i))$ if the function f is the same across all Minerva clients. As long as f is monotonically increasing and $f(U_i(r_i))$ satisfy the loose requirements outlined in Appendix B, all clients will still reach a steady state such that the $f(U_i(r_i))$ are equal; the monotonicity of f then implies that the $U_i(r_i)$ are also equal.

We take advantage of this flexibility by choosing an f that allows Minerva flows, in the steady state, to collectively occupy their fair share of the link bandwidth. In particular, for each Minerva flow to occupy the equivalent of α TCP flows on average, we *normalize* their weights, so they average to α:

$$\frac{1}{N}\sum_i \frac{r_i}{f(U_i(r_i))} = \alpha$$

where N is the number of flows on that link, and we typically choose $\alpha = 1$. Since all $U_i(r_i)$ converge to the steady-state value u, the resulting form of f is:

$$f(u) = \frac{1}{\alpha N}\sum_i U_i^{-1}(u)$$

In practical deployments of Minerva, *clients are not aware of N or the utility functions used by competing flows.* Therefore, we compute $f(u)$ using a popularity *distribution*, which video providers can measure. If video i has popularity p_i, where $\sum_i p_i = 1$, then:

$$f(u) = \frac{1}{\alpha}\sum_i p_i U_i^{-1}(u)$$

We call f^{-1} the normalization function instead of f because f^{-1} has the same signature as the U_i and can be compared to them. Fig. 1 shows a diverse set of PQ curves along with their normalization function.

If this popularity distribution over videos is accurate, then on average, N randomly sampled Minerva flows will converge to the same bandwidth as N TCP flows, giving Minerva the property of fairness to non-Minerva flows. The normalization function is also how Minerva isolates videos from different providers: if two providers A and B have videos sharing a bottleneck link, and each provider uses a suitable normalization function based on the popularity distribution of its own videos, then A's videos will occupy their fair share on average, as will B's. Therefore, videos from one provider do not affect the other.

It's important to note that Minerva achieves fairness with competing traffic *in expectation* over its video distribution. At any particular bottleneck, the aggregate bandwidth consumed by Minerva is determined largely by the PQ curves of the videos in question. Consider client i playing a video whose PQ curve lies above f, i.e. $U_i(r_i) > f(r_i)$. This client will compute $f^{-1}(U_i(r_i)) > r_i$, so $w_i < 1$, and it will occupy less than its fair share of bandwidth. The converse holds for videos with PQ curves under f. However, over several samples from the video distribution, Minerva's aggregate bandwidth footprint is fair to competing traffic, by the law of large numbers.

The function f maps utilities back to rates, so the weight computed by each Minerva flow is unitless. As a result, weights from all clients are comparable to each other, *even if* those clients are using different QoE definitions that cannot be compared (e.g., PSNR and SSIM). This property is crucial to the design of Minerva: different video providers using Minerva can properly compete with each other without sharing information about their utility functions.

5.5 Generalizing Max-Min Fairness

Practically, providers may not want any of their videos to consume *too* much more (or less) bandwidth than their equal share. Minerva allows them to optimize max-min fairness *subject to* constraints on the video's bandwidth. This is accomplished by setting upper and lower bounds on the weight w_i of each client. The weight has a straightforward interpretation: it is the ratio of that client's rate to a standard Cubic flow, on average. Keeping it from straying past a bound μ ensures that its rate is never μ times more than its equal link share.

Alternatively, Minerva is able to optimize for different notions of fairness, such as proportional fairness. We relegate further discussion of proportional fairness to Appendix D.

5.6 Using Existing Transport Protocols

§5.2 describes how the first two steps (formulate and solve) result in a weight w_i for each flow that Minerva must translate to a rate proportional to w_i, while ensuring that the flows achieve full link utilization. To accomplish this, Minerva can use any existing transport protocol that accepts such a weight and can achieve a bandwidth share proportional to that weight. Abstracting out the congestion control layer is valuable, since there are many protocols that have this property and can be plugged into Minerva. We discuss two such protocols here: Cubic [16], which is loss-based and allows Minerva to compete with wide area traffic, and FAST [21], a delay-based congestion control scheme.

For the sake of deployability, we choose Cubic [16], as our primary underlying congestion control mechanism. Minerva emulates multiple Cubic connections using a technique similar to MulTCP [10]: a client emulating w connections counts every ACK towards all w flows but counts a loss against only a single flow. Note that w need not be an integer for this to work. The average throughput of a Cubic flow as a function of loss probability p is [19]:

$$T = C\left(\frac{3+\beta}{4(1-\beta)}\right)^{1/4}\left(\frac{RTT}{p}\right)^{3/4},$$

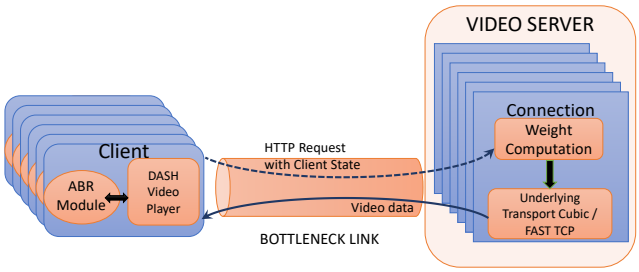


Figure 5: System architecture for Minerva. Clients run MPC and convey their state to the QUIC video server via chunk requests. The server is responsible for setting the download rate.

where β is Cubic's multiplicative decrease factor, i.e., the factor by which Cubic cuts its congestion window after a packet loss. Substituting β with

$$\beta' = \frac{\beta(w+1)+w-1}{\beta(w-1)+w+1}$$

multiplies the throughput by the desired factor of w.

We also describe how to implement Minerva on top of the delay-based FAST, which has been shown to achieve faster convergence times than Cubic [21]. Faster convergence times of the underlying congestion control protocol mean that the measurement interval T can be shortened, resulting in faster convergence of Minerva's decentralized algorithm.

A client with weight w_i sets its congestion window to

$$\text{cnwd}' = \left(\frac{\text{RTT}}{\text{RTT}_{min}} \cdot \text{cwnd} + \kappa w_i\right) \quad (4)$$

This attempts to keep κw_i packets in the bottleneck queue, for some positive constant $\kappa > 1$, and can be shown to achieve an allocation proportional to $\{w_i\}$.

6 Implementation

Minerva is implemented using a DASH Video client [2] and a HTTPS video server running over QUIC [23]. Fig. 5 shows the overall flow of data in Minerva. The QUIC server runs Minerva's weight update algorithm and uses that weight to adjust the download rate (equivalently, the server's sending rate) via the underlying congestion control protocol. In order to send the necessary state from clients to the server, e.g. playback buffer and past QoE, the client piggybacks its state onto the HTTP chunk requests as URL parameters.

6.1 Client

In Minerva, video clients are headless Google Chrome [14] browsers (Version 62) watching their video using a DASH video player. Chrome was configured to send HTTP requests to the server using QUIC. We use a modified version of dash.js (Version 2.4) [2] with a ABR module running MPC [44]. The player requests video chunks via HTTPS, and is responsible for tracking the playback buffer, rebuffer time, QoE from previous chunks, chunk download times (for bandwidth estimation), and other client parameters, such as screen size and device. This information is sent to the ABR module, which responds with the next quality for the client to fetch. Note that the client's bandwidth measurement is only used to make the ABR decision and is not sent to the video server; the server makes its own higher resolution rate measurements. However, all other data listed above are appended on video chunk requests.

6.2 Video Server

Choosing a QUIC-based server made it easy to test rate control changes. In addition to updating congestion control specific parameters, like the congestion window, the video server was also responsible for tracking client state and performing the weight update computation from §5.2. QUIC uses a persistent connection for each client, preserving state across requests made on that connection. Therefore, when video clients request a new chunk, congestion control information, such as congestion window, is carried over from the previous chunk request, preventing clients from having to re-enter slow start. Persistence grants Minerva enough time (several chunks) to converge to the QoE-fair rate allocation.

Rate Measurements. In order to predict the expected QoE from the current chunk and future chunks, the video server requires rate measurements at finer granularity than once per chunk. §5 notes that weight updates are made every T milliseconds, independently for each flow. Following a weight update, the server waits $\frac{T}{2}$ milliseconds to allow the flow to converge to its new rate, and then measures the number of bytes acked over the next $\frac{T}{2}$ milliseconds to estimate the current instantaneous rate r_i. We use $T = 25 * \text{RTT}_{\min}$, to allow time for convergence. In practice, we found that this instantaneous rate measurement can fluctuate and be noisy. Rate measurements that exceed the true rate cause the client to overestimate its utility and drop its weight significantly; this increases the chance of rebuffering. To mitigate this problem, we compute a *conservative* rate estimate $r_{i,cons} = \max(0.8r_i, r_i - 0.5\sigma)$, where σ is the standard deviation of the last 4 rate measurements. Each client uses their conservative rate estimate to calculate their utility.

Specifically, Minerva computes the weight update based on §5.2 with two modifications. First, $w_i = \frac{\tilde{r}_{i,cons}}{f(U_i(\tilde{r}_{i,cons}))}$, where $\tilde{r}_{i,cons}$ is an exponential weighted moving average (EWMA) over the conservative instantaneous rate:

$$\tilde{r}_{i,cons} = 0.1 r_{i,cons} + 0.9 \tilde{r}_{i,cons}$$

Second, we apply another EWMA, $\tilde{w}_i$, to the computed weight:

$$\tilde{w}_i = 0.1 w_i + 0.9 \tilde{w}_i$$

Minerva passes $\tilde{w}_i$ as the weight to the send step. Minerva clamps this weight to the interval [0.5,20], although this is only as a precaution; we do not observe the weight passing these bounds in practice.

Control Flow. Clients only begin updating weights after downloading the first chunk, which is always fetched at the lowest quality. This allows the client's bandwidth share to stabilize before starting Minerva's rate update algorithm. Starting on the second chunk, when a new rate measurement is available after T milliseconds, Minerva allows clients to update their weights. When computing the utility function, we use $\varphi_1 = \frac{1}{N}$, where N is the number of chunks already played, and $\varphi_2 = 1$. This equally weights the contribution from past QoE, current chunk, and value function.

Function Evaluation. Minerva's design involves evaluating both a value function, to estimate the QoE of future chunks, and a normalization function. To save computation time when running the 'Solve' step, these functions are pre-computed before running the video server. Recall that the value function $V(r,b,e_i)$ is a function of the download rate, buffer level, and the bitrate currently being fetched. We precompute the value function for every chunk, for each combination of rates, from 0 to 8 Mbit/s in intervals of 100 kbit/s; buffers, from 0 to 40 seconds in intervals of 0.05 seconds; and previous bitrates. This amounts to 38.4 million values for a single video, which is approximately 153MB. We losslessly compress the value functions by storing them as a series of contiguous line segments, reducing the space overhead to between 6MB and 16MB, depending on the video. We discuss ways to reduce this overhead further in §7.

The normalization function is precomputed from the videos being watched before the videos start, using the PQ curve for each video.

Although normalization ensures that Minerva converges to its fair share of bandwidth, this may not be true during the convergence process. To compensate for the higher bandwidth share occupied during convergence, we use $\alpha = 1.65$ (QUIC's Cubic implementation emulates 2 flows). The normalization function is stored as a table of values and is loaded when the video server launches.

Tracking client state. The QUIC server keeps an estimate of the client's buffer, which it uses to predict the QoE expected from downloading the current chunk. This estimate is updated after every chunk request with the true buffer size, which the client sends as a URL parameter on the HTTP request. We find that DASH incurs an overhead of 0.6 seconds for each chunk, so Minerva decrements the estimate by 0.6 seconds before using it in its weight computation. In addition, the client sends the total QoE it has experienced for all chunks watched so far, which the server incorporates into the utility function.

Overhead. The overhead added by Minerva's computation on the video server is negligible: we observed no quantifiable increase in CPU usage when running Minerva as compared to QUIC Cubic.

7 Discussion

Global vs Local Fairness. Minerva guarantees that videos are fair to Cubic in aggregate over a set of links, provided that the normalization function is crafted from the popularity distribution of videos over those links. However, popularity distributions may vary both geographically and over time. In particular, the global distribution over all videos may differ substantially from the videos being watched in a particular geographic area, and may additionally vary from morning to evening. Therefore, if the provider bases their normalization off a single global popularity distribution, videos will be fair to Cubic on average *globally* but may not achieve *local* fairness. Achieving global fairness at the expense of local fairness may not be desirable, since videos playing in the same region may then achieve significantly more or less than their fair share of bandwidth. Providers have two remedies to retain control over the weights of their videos and prevent them from straying too far from their fair-share allocation.

First, providers can achieve local fairness by using a different normalization function in each geographic region, which accurately captures the popularity of videos in that particular area. The more granular the popularity distribution, the more likely it is that videos in the corresponding area closely match the distribution, and the less Minerva videos stray from their fair-share allocation. Taken to the extreme, if providers knew precisely which sets of videos shared common bottleneck links, they could guarantee that their videos achieve their fair share allocation on every link. In practice, obtaining link-level statistics is difficult; however, it may be more feasible for providers to approach local fairness by specializing their normalization functions to less granular regions.

Second, Minerva exposes the client's weight w_i, so providers may cap it at a particular value to limit how far a video's bandwidth strays from its equal share. After normalization, w_i has a straightforward interpretation: it will occupy w_i times what a standard Cubic flow would occupy. Keeping w_i between 0.5 and 2, for example, ensures that no video grabs less than half or more than twice its fair share, respectively. However, a more stringent cap limits the range in which Minerva can operate, so QoE fairness may suffer. We evaluate this approach in §8.6.

Deployment Considerations. Minerva requires two types of modifications to video servers. First, Minerva requires servers that keep track of application state for each video flow they serve and have the ability to adjust their sending rates according to that application state. This may mean that conventional CDNs, which are stateless and implement a traditional TCP stack, are ill-suited for serving Minerva videos. However, some providers, such as Netflix, already deploy video servers with custom software [33].

Second, Minerva's current implementation precomputes the value functions and stores them on the video server to avoid the overhead of real-time computation. This incurs a memory overhead of 16MB per video, and may be prohibitive when a single server serves a large number of videos simultaneously. There are multiple approaches to reducing this overhead. First, we can compute the value function at a coarser granularity, e.g., for buffers at every 0.5 seconds instead of 0.05 seconds. Second, the video clients can compute the value function on demand and send them to the video server when requesting a chunk. Recall that for any given chunk, the value function $V(r,b,e_i)$ depends on the rate, buffer, and previous bitrate. Since the client knows both its current buffer level B and previously played bitrate E, it sends $V(r,B,E)$, now only a function of r, for a handful of buffer values close to B. We estimate the size of this representation to be around 2kB per chunk.

8 Evaluation

Minerva has two design goals: (1) maximizing QoE fairness while (2) occupying its fair share of the link capacity. We start by isolating the first goal and asking how well Minerva is able to improve QoE max-min fairness in the absence of competing traffic (§8.2), including dynamic environments with videos arriving and leaving at different times (§8.3). To test the second goal, we add cross-traffic, and show that Minerva shares bandwidth fairly with other videos, scales to many clients (§8.4), and adapts to link conditions that vary with time (§8.4, §8.4.1). Third, we show that Minerva works in the wild, over a real residential network (§8.5). Fourth, we explore how Minerva can optimize for alternatives to max-min fairness (§8.6). Fifth, we consider the case of two providers competing with Minerva (§8.7).

8.1 Setup

8.1.1 System Details All clients use dash.js (version 2.4) running on Google Chrome (version 62), with QUIC support enabled. Each client runs in a headless Chrome browser, on an Amazon AWS r5a.4xlarge EC2 instance. The server is based off the HTTP server distributed with Chromium, modified only to implement Minerva's rate control scheme.

Clients choose the bitrate of the next chunk by contacting an ABR server that implements MPC. The ABR server is colocated on the same machine as the clients and is shared by all of them. The overhead of a client's request to the ABR server, including latency to/from the server and computation time, is 70ms seconds, which is small relative to the length of a video chunk download (4 seconds). Unless otherwise noted, clients begin watching their videos at the same time, and fairness is computed over a 200 second interval from the start of the video.

The bulk of our evaluation considers Minerva videos sharing a single bottleneck link, emulated with a Mahimahi [34] shell. The link has a minimum RTT of 20ms and a buffer with a capacity of 1.5 bandwidth delay products (BDPs). The link runs a PIE [35] AQM scheme for both Minerva and baseline flows, with a target delay of 15ms.[4] However, we also evaluate over droptail buffers. Since all video traffic travels from server to clients, our evaluation only alters

[4] The PIE AQM scheme is a part of the DOCSIS3.1 standard and widely deployed in residential areas: Comcast has already deployed DOCSIS3.1 compliant routers to 75% of their customers [15, 30].

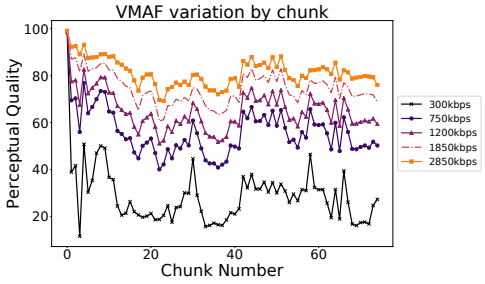


Figure 6: VMAF scores for all chunks in video V9, shown for the 5 lowest bitrates.

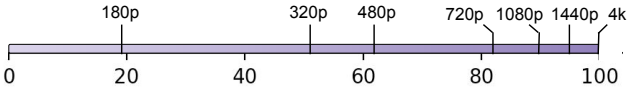


Figure 7: VMAF scores for well known resolutions, averaged across our corpus. A gain of 7.65 corresponds to a bump from 720p to 1080p on a 4k TV at standard viewing distance.

the Mahimahi downlink (outside to inside) capacity; the uplink is fixed at 10 Mbit/s and is never saturated.

8.1.2 Videos Our corpus consists of 19 4K videos between 4 and 5 minutes long, which we label V1 to V19. They span a diversity of genres, including news, action, and animation (Appendix A). Each video has a corresponding *VMAF score*, a perceptual quality metric designed to predict user perceived quality [24]. We use the VMAF score (version 0.3.1) as the PQ function for each chunk.

For a single video, both the chunk sizes and VMAF scores differ from chunk to chunk, even at the same bitrate. Fig. 6 shows the variation in VMAF scores in video V9. Note that while adjacent chunks may have very different qualities, there is also a temporal trend over several chunks. This property makes it important for Minerva to dynamically adjust rates throughout the video.

Since the videos span several genres, the VMAF scores also vary significantly between videos. Fig. 1 visualizes the score of the 19 videos, averaged over all chunks, at the eight available bitrates. Our results are presented in terms of VMAF scores. To contextualize these numbers, Fig. 7 maps popular resolutions to their corresponding scores on our corpus. For example, a bump from 720p to 1080p equates to a gain of 7.65 points. We use this as a benchmark for our evaluation of Minerva, since VMAF scores have a close-to-linear correlation with user-perceived quality [24]. That is, a delta in VMAF accurately predicts a delta in viewer experience, regardless of bitrate or video genre. Consequently, any delta of 7.65 points is comparable to the jump in quality a user would perceive between 720p and 1080p on a 4K TV, uniformly along the bitrate spectrum.

8.1.3 Metrics We use the definition of Quality of Experience defined in Equation 1. The VMAF score $P(c_k)$ ranges from 0 to 100, while our QoE metric uses a rebuffering penalty of $\beta = 25$ and a smoothness penalty of $\gamma = 2.5$. We evaluate Minerva on max-min QoE fairness.

Our baseline is a client running unmodified dash.js over Cubic. Since Minerva is implemented over QUIC, we use the Cubic implementation provided by QUIC, which simulates 2 connections by default. We call this system "Cubic".

8.2 Benchmarking QoE Fairness

We first evaluate Minerva's ability to improve QoE fairness, by playing videos over a fixed emulated PIE-enabled link with capacities ranging from 4 Mbit/s to 16 Mbit/s. We conduct a total of 44 runs, in each one selecting 4 distinct videos uniformly at random from our corpus. We play the same videos over four additional benchmarks:

(1) QUIC Cubic running over a PIE-enabled link.
(2) Minerva over a link with a droptail buffer of 1.5 BDPs.
(3) BBR, running over a droptail link sized to 1.5 BDPs.
(4) The "Fixed-Rate Optimal", computed offline, which is the max-min QoE fairness assuming that each video receives a constant rate over the course of the video. This optimal changes for each video combination, since it depends heavily on the PQ curves of the videos involved.

Fig. 8 shows two representative combinations of videos with these points of comparison. There are four main takeaways.

First, the magnitude of improvement of Minerva over Cubic depends on the link bandwidth and the particular videos used. Videos whose PQ values are far from each other at a particular link rate require a larger bandwidth difference to achieve the same QoE, creating more room for Minerva to surpass Cubic. For example, Fig. 8a and Fig. 8b show average improvements of 12.5 and 3 points, respectively. At high link rates, e.g. 16 Mbit/s, the difference between videos' PQ curves is typically smaller, so Cubic's allocation is closer to optimal and there is less room for Minerva to improve. Conversely, at low link rates, e.g. 4 Mbit/s, or 1 Mbit/s per video on average, the gap between utilities is large. However, VMAF curves are steep at those low rates; in order to improve the worse-off video, QoEs of other videos must drop sharply. As a result, the QoE fairness gain is small. We observe the largest gains on a 10 Mbit/s link, where videos' utilities have a sufficiently large difference with gentler slopes.

Fig. 9 shows the gains Minerva is able to achieve over Cubic on a variety of different video combinations and links. The videos are chosen uniformly at random without replacement from our corpus, and are played over a 4 Mbit/s, 10 Mbit/s, and 16 Mbit/s link. To put the magnitude of Minerva's improvement over Cubic into perspective, consider that the average difference between 720p and 1080p is 7.65 VMAF points, on a scale of 100. For 24% of cases across our corpus (24%, 32%, and 16% on 4 Mbit/s, 10 Mbit/s, and 16 Mbit/s, respectively), the worst-performing video client sees a boost in viewing experience equivalent to or better than a jump in resolution from 720p to 1080p.

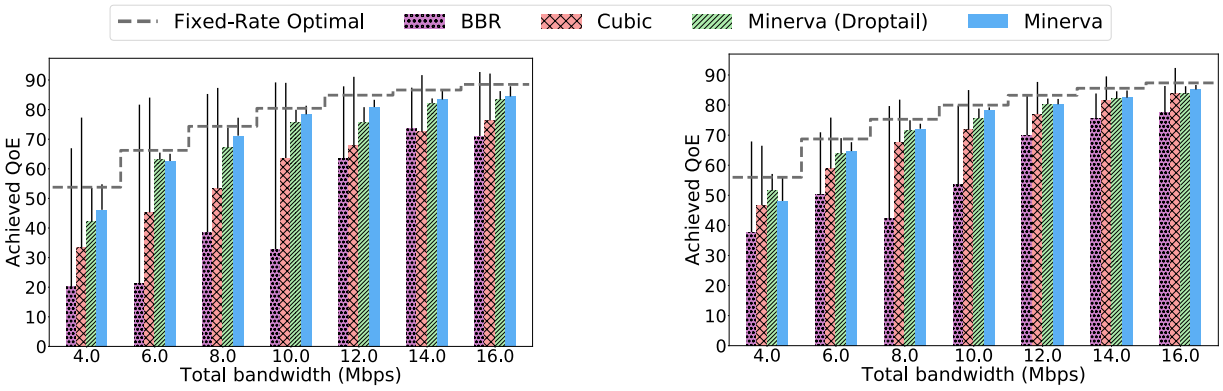
The second takeaway is that Minerva substantially closes the gap between Cubic and the optimal allocation. For videos like Fig. 8a, Cubic's gap to optimal is 17.5 points on average, which Minerva reduces to 4, an improvement of 77%. For videos with more similar PQ curves (Fig. 8b), Cubic's bandwidth split is closer to optimal (6.5 points), but Minerva is still able to bring this down by 53% (3 points).

Third, Minerva's improvements over Cubic do not hinge on using PIE; it performs nearly identically over droptail, with only a marginal reduction in fairness. This reduction is 2.5 points and 0.5 points on average, respectively, over the scenarios in Fig. 8. Cubic relies on independent drops in order to give weighted Cubic flows their expected bandwidth share. Links with droptail queues result in drops that are bursty and correlated, a possible barrier for videos to achieve their desired rate ratios.

Fourth, BBR may achieve better link utilization [7, 11] but it performs poorly from a fairness perspective. Some BBR flows, determined seemingly arbitrarily, grab a larger bandwidth share, starving the others. This observation matches previous findings that BBR does not achieve fairness with itself [17].

8.3 Minerva in a Dynamic Environment

We now evaluate Minerva in a setting where videos sharing the same link start and stop over the course of more than an hour. In



(a) Videos V2, V7, V18, V19

(b) Videos V1, V2, V6, V10

Figure 8: Max-min QoE fairness for protocols over a constant link. The black whiskers extend from the minimum to maximum QoEs.

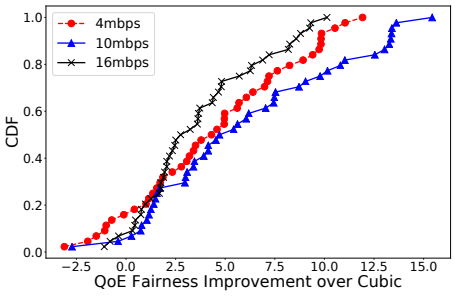
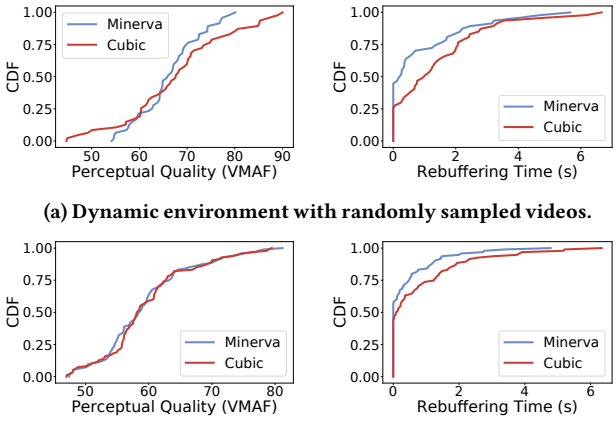


Figure 9: Minerva's improvement in QoE fairness over Cubic over 44 runs, each using 4 distinct videos sampled from our 18-video corpus.



(a) Dynamic environment with randomly sampled videos.

(b) Dynamic environment with identical videos.

Figure 10: Minerva's effect on visual quality and rebuffering time in a dynamic setting, with videos joining and leaving.

contrast to §8.2, in which all videos started simultaneously and played throughout the entire experiment, each video starts at a randomly determined start time and then plays to completion.

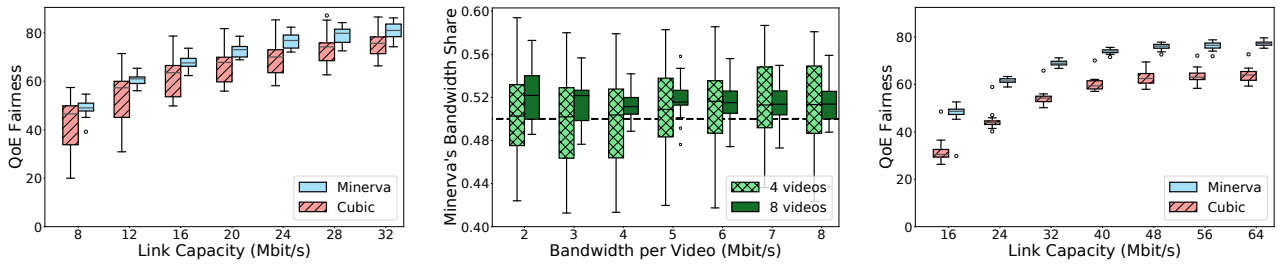
We sample 48 videos uniformly at random from our corpus to play over a link with a constant total capacity of 8 Mbit/s. Video start times are determined by a Poisson process such that, on average, 4 videos share the link at a given time. Appendix E shows the arrival pattern of the videos in this experiment. We run the same configuration of videos using both Minerva and Cubic as the underlying transports.

Since Minerva siphons bandwidth away from videos with a high perceptual quality, one might expect those videos to be at higher risk for rebuffering. This additional rebuffering may not be reflected by max-min QoE fairness, which considers only the video with the worst quality. Therefore, Fig. 10a examines the effect of Minerva on *both* the perceptual quality and rebuffering time, considering the distribution over all videos. In our setup, every video fetches the first chunk at the lowest bitrate and must stall until it finishes downloading. Additionally, Minerva's rate control does not kick in until the second chunk. We therefore focus on *non-startup rebuffering time*, i.e. the amount of time the video spends stalling when downloading any chunk after the first. Unlike startup delay, non-startup rebuffering time materially interrupts the viewing experience and is visually jarring for users.

Minerva increases the minimum QoE by 9.3 VMAF points, while simultaneously reducing rebuffering time across the board: average total rebuffering time (including startup delay) decreases by 17%, while the average non-startup rebuffering time falls by 38%. The drop in rebuffering time is due to Minerva's use of a buffer-aware utility function that captures the negative effects of rebuffering. By understanding when a client is at risk for rebuffering, Minerva can increase that client's weight, improving its bandwidth share and growing its buffer. This allows videos to tap into the *global buffer pool* formed by the other clients sharing the bottleneck link.

Buffer Pooling. The results in Fig. 10a combine improvements due to both *static* attributes, such as the differences in PQ values between videos, and *dynamic* factors like buffer level, which vary across sessions for the same video. To isolate the potential of the global buffer pool, we repeat the same long-running experiment, but with all clients watching the same video. This eliminates the potential for QoE improvements arising from static differences, and focuses on the ability of Minerva to take advantage of dynamic differences in buffer size among clients sharing the same link. Additionally, to place clients in challenging network conditions where rebuffering is more likely, the Poisson process that governs video start time is adjusted so that 8 videos share the link at any given time, on average.

Fig. 10b shows that, by harnessing the buffers of other clients, Minerva is able to significantly reduce time spent rebuffering, but does not hurt the video's perceptual quality in the process. Minerva reduces average visual quality by only 0.4% compared to Cubic, but is able to cut average non-startup rebuffering time by 47% and the number of rebuffering events by 45%. This shows that Minerva can achieve notable gains in viewing experience that do not just stem from perceptual quality differences. In particular, even if information about the videos' perceptual qualities is absent or videos are perceptually similar, Minerva can still provide substantial gains by tapping into the global buffer pool.



(a) 4 videos each. (b) Minerva's aggregate bandwidth share (c) 8 videos each.

Figure 11: Minerva improves QoE fairness when competing with Cubic, each using 4 (a) and 8 (c) videos. Minerva's bandwidth share, as a fraction of total traffic, for these videos is close to equal (b), indicating that it is fair to cross-traffic.

8.4 Fairness with Cross Traffic

Minerva should improve QoE fairness without violating its second design goal: achieving an equal bandwidth split with competing traffic, on average. That is, a collection of N Minerva flows should occupy the same bandwidth as N Cubic flows. To evaluate Minerva on connection-level fairness, we play multiple Minerva videos, again chosen uniformly at random, simultaneously with the same N videos running over Cubic. All videos start at the same time and play for the duration of the experiment, as in §8.2. The reasons for choosing video cross-traffic are twofold. First, even videos running over Cubic are not able to achieve fairness with long-lived Cubic flows, due to the idle periods between chunk requests [45]. Pitting Minerva against other video flows evens the playing field. Second, the QoE fairness achieved by the Cubic videos serves as a convenient point of comparison for Minerva.

Fig. 11a shows that Minerva videos achieve an improvement in median QoE fairness of 5 points over Cubic. 42% of cases result in a 7.65 improvement, corresponding to the same perceptual jump between 720p and 1080p. Minerva's gains in this setting should be viewed in the context of its bandwidth share, i.e. its fraction of the total traffic throughput (Fig. 11b). Minerva maintains a 75th percentile bandwidth share that is within 5% of a perfectly even split with Cubic across the board. However, we note that slightly uneven bandwidth shares may be partly responsible for Minerva's QoE fairness gains exceeding those in the setting without cross-traffic.

Although Minerva's median QoE fairness is higher, Cubic still attains a maximum fairness that exceeds Minerva's. These are due to runs in which all videos have PQ curves that lie above the normalization function. In these particular cases, Minerva occupies less than its fair share of bandwidth, which reduces QoE of all videos. The converse is true when all the videos lie under the normalization curve. These situations, while they exist, constitute only 12% of all cases.

The gap between Minerva and Cubic only improves when more clients are added (Fig. 11c). The median improvement is 14.9 points when Minerva and Cubic run 8 videos each; for perspective, a change from 360p to 480p is an improvement of only 11 points. A full 95% of cases see an improvement larger than the 7.65 point gap between 720p and 1080p. The large improvement is due to this larger set of videos having a higher chance of including two videos with significantly different PQ curves; this creates a large room for improvement over Cubic. A larger number of videos also means that the average PQ curves of the videos more closely matches the normalization function, so Minerva bandwidth share is closer to 50% with lower variance. (Fig. 11b).

8.4.1 Variable Cross Traffic We now shift from video cross-traffic to web cross traffic, still over an emulated link. This

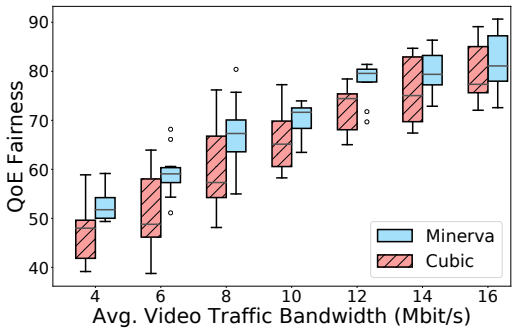


Figure 12: QoE fairness achieved by Minerva and Cubic, each playing 4 videos, over 10 runs on a 20 Mbit/s link with various loads of emulated web cross traffic.

experiment pits Minerva against cross-traffic it is likely to encounter in practice, and tests its performance over link conditions that vary with time. Time varying links stress Minerva's future QoE estimation, which assumes that the client will continue to see the rate they most recently measured for the next several chunks.

We emulate web traffic from 200 servers, drawing flow sizes from an empirical distribution derived from CAIDA data [1]. The cross traffic's average offered load varies between 4 and 16 Mbit/s on a 20 Mbit/s emulated link. We perform 10 runs of Minerva with each cross-traffic load, using 4 distinct videos each time. For comparison, we separately run Cubic over identical loads and videos. Minerva and Cubic achieve similar aggregate bandwidths, so Minerva's gains do not come at the expense of competing traffic; see Appendix F for a sample trace of Minerva's performance.

We observe that Minerva's gains do not suffer when running over a time-variable link. The distribution of Minerva's QoE fairness is more diffuse, due to the workload variability. However, across all tested loads, 37% of video combinations hit our benchmark of 7.65 points. We conclude that Minerva is still able to maintain its QoE fairness improvements and equal-bandwidth guarantees even in variable link conditions.

8.5 A Real Residential Network

The experiments thus far have tested Minerva in a controlled environment using emulated links. To test Minerva in the wild, we run it over an actual residential WiFi link during both peak evening and non-peak hours. The WiFi router supports speeds of 343 Mbit/s [5]. The ISP advertises a rate of 25 Mbit/s, although we see a larger rate in our measurements.

We run two video servers on an Amazon EC2 r5a.4xlarge instance; one is responsible for serving Minerva videos, while the other serves videos over Cubic. We simultaneously launch 8 clients

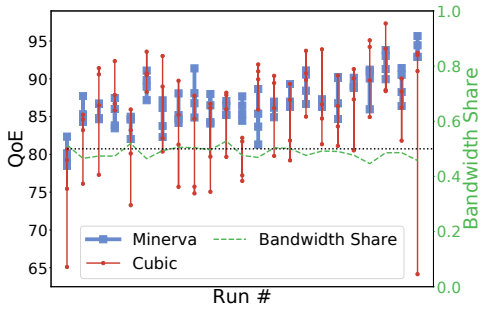


Figure 13: QoEs of Minerva and Cubic videos for 23 runs over a real residential link, each with randomly selected videos. Runs are ordered by increasing total bandwidth.

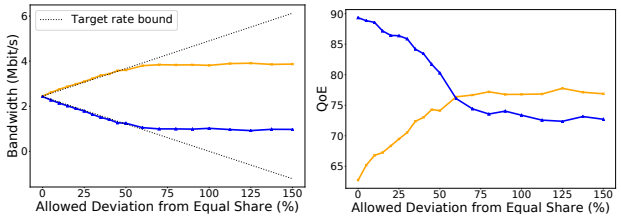


Figure 14: QoE and bandwidth for two Minerva videos, as a function of the rate bounding policy they implement.

(4 Minerva and 4 Cubic) as separate Google Chrome instances for 200 seconds. Fig. 13 compares the QoEs achieved by Minerva flows to those achieved by Cubic over 23 independent runs. We observe that the bandwidth of the link fluctuates over time, so results from different runs are not comparable. Averaged over all runs, Minerva stays close to its equal bandwidth share, while improving the minimum QoE from 79 to 85 points; however, the improvement varies substantially between runs, with 3 of 25 runs (12%) not seeing any improvement over Cubic. Minerva also pushes QoEs of all videos closer together: while the range between Cubic video QoEs is 11 VMAF points, Minerva reduces it to 3.75.

8.6 Generalizing Max-Min Fairness

Max-min QoE fairness may not be suitable for all video providers, since it may require a large deviation from each client's equal-bandwidth share, particularly when videos have substantially different PQ curves; however, Minerva is not tied to this metric. Here, we consider two ways that Minerva generalizes past max-min fairness.

First, Minerva allows policies that optimize max-min fairness *subject to the constraint* that the bandwidth shares of the videos don't exceed a value set by the provider. In particular, providers can implement policies to limit the amount of bandwidth a video grabs above its equal-share allocation. Of course, restricting the client limits the space of rate allocations available to Minerva; Fig. 14 shows the tradeoff as a function of the amount by which the provider chooses to constrain the flows. Minerva videos stay within the target rate bounds, improving QoE fairness as the bounds are relaxed. Ideally, as the bounding policy is relaxed, the QoEs would converge to be equal; however, Minerva's allocation isn't perfectly optimal and still leaves a small gap.

Second, Minerva's weight update rule allows it to optimize for other fairness metrics by choosing the proper utility function for each client. Appendix D demonstrates how Minerva can optimize for proportional fairness, another popular fairness metric.

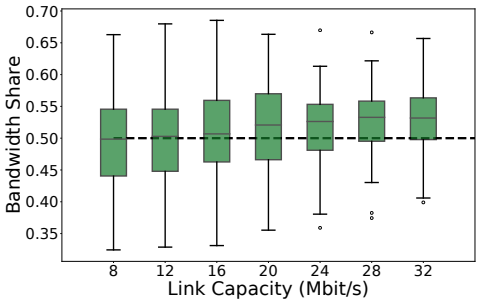


Figure 15: Bandwidth share between two providers using different metrics (PSNR and VMAF) over 20 video combinations. On average, the providers achieve close to a 50% bandwidth split.

8.7 Multiple Providers using Minerva

We demonstrate here that multiple providers can use Minerva independently on the same bottleneck link, without exchanging information or even being aware of other Minerva providers sharing the link.

We run two instances of Minerva against each other, emulating two providers using different measures of video quality: VMAF and PSNR [37]. These metrics use different scales and cannot directly be compared to each other (see Appendix G). Each provider serves 4 videos and uses a normalization computed over the entire corpus with their respective metric. Fig. 15 shows that both providers split bandwidth about equally on average, over 20 randomly sampled video combinations and a variety of link rates. Appendix G contains additional results showing that both providers still achieve QoE fairness improvements despite using different metrics.

This result has two implications. First, the normalization step is critical to the performance of Minerva. Normalization places all providers' weights on the same scale, without them sharing any information. Without it, providers would have to collaborate to decide on a single QoE metric. Second, providers using Minerva compete fairly with *each other*, not just Cubic. This property allows any number of providers to share the same bottleneck link without worrying about the others' presence.

9 Conclusion

Despite the growth of video streaming traffic, there has been relatively little work on deployable solutions to improve QoE fairness between multiple users. We propose Minerva, the first system that achieves QoE fairness in a distributed manner, requiring no changes to underlying network infrastructure. Minerva's formulate-solve-send control flow updates rates for each client independently, such that when they share a bottleneck link, their rates converge to a bandwidth allocation that maximizes QoE fairness, while competing fairly with other traffic on average. We implement Minerva over QUIC and show that, 24% of the time, it can improve the worst viewing experience among a set of videos, by an amount roughly equivalent to a jump from 720p to 1080p. Additionally, in a dynamic environment, Minerva can take effectively pool the buffers of the competing clients to achieve reduction in rebuffering time of up to 47%. Minerva generalizes well to multiple clients, different link speeds, and real residential links, suggesting that it is a deployable solution for video providers seeking to optimize QoE fairness.

Acknowledgements: We thank our shepherd, Te-Yuan Huang, and the SIGCOMM reviewers for their valuable feedback. This work was funded in part by NSF grants CNS-1617702, CNS-1751009, CNS-1563826, a Facebook Faculty Research Award, Sloan Research Fellowship, and sponsors of MIT DSAIL.

References

[1] [n. d.]. Empirical Traffic Generator. https://github.com/datacenter/empirical-traffic-gen. ([n. d.]).

[2] Akamai. 2016. dash.js. https://github.com/Dash-Industry-Forum/dash.js/. (2016).

[3] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 44–58.

[4] Mark Allman, Vern Paxson, Wright Stevens, et al. 2009. TCP Congestion Control. (2009). IETF RFC 5681.

[5] Arris. [n. d.]. TG862G/CT Xfinity Residential Gateway & Router. https://arris.secure.force.com/consumers/ConsumerProductDetail?p=a0ha000000GOZ3yAAH. ([n. d.]).

[6] Abdelhak Bentalab, Ali C. Bergen, Saad Harous, and Roger Zimmermann. 2018. Want to Play DASH? A Game Theoretic Approach for Adaptive Streaming over HTTP. In *Proceedings of the 9th ACM Multimedia Systems Conference (MMSys)*. ACM.

[7] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5, Article 50 (Oct. 2016), 34 pages. https://doi.org/10.1145/3012426.3022184

[8] Junyang Chen, Mostafa Ammar, Marwan Fayed, and Rodrigo Fonseca. 2016. Client-Driven Network-level QoE fairness for Encrypted'DASH-S'. In *Proceedings of the 2016 workshop on QoE-based Analysis and Management of Data Communication Networks*. ACM, 55–60.

[9] Cisco. 2018. Cisco Visual Networking Index: Forecast and Methodology, 2017-2022.

[10] Jon Crowcroft and Philippe Oechslin. 1998. Differentiated End-to-end Internet Services Using a Weighted Proportional Fair Sharing TCP. *SIGCOMM Comput. Commun. Rev.* 28, 3 (July 1998), 53–69. https://doi.org/10.1145/293927.293930

[11] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *NSDI*.

[12] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *NSDI*.

[13] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. 2017. Salsify: Low-Latency Network video through Tighter Integration between a Video Codec and a Transport Protocol. In *NSDI*. USENIX.

[14] Google. [n. d.]. Google Chrome Web Browser. https://www.google.com/chrome/. ([n. d.]).

[15] Greg White. 2015. Active queue management in DOCSIS 3.1 networks. *IEEE Communications Magazine* 53, 3 (March 2015), 126 – 132.

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review* 42, 5 (July 2008), 64–74.

[17] Mario Hock, Roland Bless, and MArtina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *25th International Conference on Network Protocols*. IEEE.

[18] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2015. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 187–198.

[19] IETF. 2006. Cubic for Fast Long-Distance Networks. https://tools.ietf.org/id/draft-ietf-tcpm-cubic-06.html. (2006).

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

[21] Cheng Jin, David X Wei, and Steven H Low. 2006. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. on Networking* 14, 6 (2006), 1246–1259.

[22] Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. 1998. Rate control for communication networks: Shadow price, proportional fairness, and stability. *The Journal of the Operational Research Society* 49, 3 (1998), 237–252.

[23] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *SIGCOMM*.

[24] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward A Practical Perceptual Video Quality Metric. https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652. (2016).

[25] Zhi Li, Xiaoqing Zhu, Josh Gahm, Rong Pan, Hao Hu, Ali C. Begen, and Dave Oran. 2014. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. *IEEE Journal on Selected Areas in Communications*.

[26] Ahmed Mansy, Marwan Fayed, and Mostafa Ammar. 2015. Network-layer fairness for adaptive video streams. In *IFIP Networking Conference (IFIP Networking), 2015*. IEEE, 1–9.

[27] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 197–210.

[28] Jim Martin, Yunhui Fu, Nicholas Wourms, and Terry Shaw. 2013. Characterizing Netflix bandwidth consumption. In *10th Consumer Communications and Networking Conerence*. IEEE.

[29] Anush Krishna Moorthy, Lark Kwon Choi, Alan Conrad Bovik, and Gustavo de Veciana. 2012. Video Quality Assessment on Mobile Devices: Subjective, Behavioral, and OBjective Studies. *IEEE Journal of Selected Topics in Signal Processing* 6, 6 (2012), 652 – 671.

[30] Multichannel News. 2017. Comcast Goes Wide With DOCSIS 3.1 Gigabit Gateway. https://www.multichannel.com/news/comcast-goes-wide-docsis-31-gigabit-gateway-416930. (2017).

[31] Kanthi Nagaraj, Dinesh Bharadia, Hongzi Mao, Sandeep Chinchali, Mohammad Alizadeh, and Sachin Katti. 2016. NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters. In *SIGCOMM*.

[32] Hyunwoo Nam, Bong Ho Kim, Doru Calin, and Henning Schulzrinne. 2013. A mobile video traffic analysis: Badly designed video clients can waste network bandwidth. In *Globecom Workshop*.

[33] Inc. Netflix. 2019. Open Connect. https://openconnect.netflix.com/en/. (2019).

[34] Ravi Netravali, Anirudh Sivaraman, Keith Winstein, Somak Das, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of USENIX ATC*.

[35] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili S. Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *14th International Conference on High Performance Switching and Routing*.

[36] D.C. Pew Research Center, Washington. 2017. A third of Americans live in a household with three or more smartphones. http://www.pewresearch.org/fact-tank/2017/05/25/a-third-of-americans-live-in-a-household-with-three-or-more-smartphones/. (May 2017).

[37] David Salomon. 2007. *Data Compression: The Complete Reference*. Springer.

[38] Sandvine Intelligent Broadband Networks. 2016. Global Internet Phenomena: Latin America and North America. https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf. (2016).

[39] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming over the internet. *IEEE MultiMedia* 18, 4 (2011), 62–67.

[40] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *CoRR* abs/1601.06748.

[41] Jim Summers, Tim Brechy, Derek Eager, and Alex Gutarin. 2016. Characterizing the Workload of a Netflix Streaming Video Server. In *International Symposium on Workload Characterization*. IEEE.

[42] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* 3, 4 (2004), 600–612.

[43] Xiaoqi Yin, Mihovil Bartulović, Vyas Sekar, and Bruno Sinopoli. 2017. On the efficiency and fairness of multiplayer HTTP-based adaptive video streaming. In *American Control Conference (ACC), 2017*. IEEE, 4236–4241.

[44] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, Vol. 45. ACM, 325–338.

[45] Chao Zhou, Chia-Wen Lin, Xinggong Zhang, and Zongming Guo. 2017. TFDASH: A fairness, stability, and efficiency aware rate control approach for multiple clients over DASH. *IEEE Transactions on Circuits and Systems for Video Technology* (2017).

Name	Description
V1	Aerial Footage
V2	Nature
V3	Gaming Livestream
V4	Cooking / Nature
V5	Advertisement (GoPro)
V6	Soccer Match
V7	Action Movie Trailer
V8	Animated Music Video
V9	Animated Short
V10	Tornado Footage
V11	Cat Video
V12	Animated Short
V13	News (Video Blog)
V14	Lecture
V15	Action Movie Clip
V16	Video Game Trailers
V17	Music Video
V18	Advertisement (Apple)
V19	News (Documentary)

Table 1: The 19 videos used in our evaluation corpus.

Appendices are supporting materials that have not been peer reviewed.

A Video Corpus

Table 1 is the list of videos in our corpus with their genres. The labels correspond to those in Fig. 1.

B Proof of Convergence

Let $\mathbb{R}_+$ denote the non-negative real numbers. Consider any function $f:\mathbb{R}_+\to\mathbb{R}_+$, where $\mathbb{R}_+$ denotes the non-negative real numbers.

Definition B.1. We say that a function $f:\mathbb{R}_+\to\mathbb{R}_+$ is (α,β)-subquadratic if for all x,y such that $x\le y$:

$$\left(\frac{y}{x}\right)^{\alpha}\le\frac{f(y)}{f(x)}\le\left(\frac{y}{x}\right)^{2-\beta}$$

Intuitively, the subquadratic condition implies that f is monotonically increasing but does not grow too fast. In particular, over any compact interval, any increasing concave function is $(\alpha,1)$-subquadratic for some $0<\alpha\le 1$.

The following theorem says that, as long as the client is $(\alpha,\beta)-$subquadratic for *any* $\alpha,\beta>0$, their rates converge to optimal in a doubly logarithmic number of iterations. This condition is fairly broad: it includes all concave function as well as some convex ones.

Theorem B.2. *Let $r_{i,t}$ be the rate of client i after t iterations of Minerva's decentralized weight update algorithm (§5.2), let the shared link have constant capacity c, and suppose that each client utility function $U_i(r_i)$ satisfies the following conditions:*

- *$U_i(x)\ge 0$.*
- *There exist $\alpha>0,\beta>0$ such that $U_i(x)$ is (α,β)-subquadratic on the interval $[0,c]$. We take α,β to be the maximal such values.*
- *There exists an optimal allocation of rates $\{r_i^*\}$, with $r_i^*>0$, such that the $U_i(r_i^*)$ are equal.*

Then for all iterations t:

$$\sum_i|\log r_{i,t}-\log r_i^*|<K(1-\min(\alpha,\beta))^t$$

where K is a constant that depends on the initial rates.

Proof. We prove convergence for two clients, with utility functions U_1 and U_2; the result easily extends to more clients. We assume the link capacity is a constant c and that the rates of the two clients, $r_{1,t}$ and $r_{2,t}$, always sum to c. The optimal rates for the two clients are r_1^* and r_2^*, which satisfy $U_1(r_1^*)=U_2(r_2^*)\equiv u^*$.

Without loss of generality, assume $r_{1,t}<r_1^*$, which implies that $r_{2,t}>r_2^*$. We aim to prove convergence of $r_{1,t}\to r_1^*$ and $r_{2,t}\to r_2^*$. Since we are bound by the constraint that $r_{1,t}+r_{2,t}=c$, it is sufficient to prove that $\frac{r_{2,t}}{r_{1,t}}\to\frac{r_2^*}{r_1^*}$ or, equivalently, $\frac{r_1^*}{r_{1,t}}\cdot\frac{r_{2,t}}{r_2^*}\to 1$. Define:

$$X_{1,t}=\frac{r_1^*}{r_{1,t}}\qquad X_{2,t}=\frac{r_{2,t}}{r_2^*}$$

so our goal is to show $X_{1,t}X_{2,t}\to 1$.

In each iteration of the weight update, the clients compute weights $w_i=\frac{r_i}{U_i(r_i)}$, and Minerva's solve step achieves new rates in proportion to these weights:

$$\frac{r_{2,t+1}}{r_{1,t+1}}=\frac{w_2}{w_1}=\frac{u_1}{u_2}\frac{r_{2,t}}{r_{1,t}}\tag{5}$$

Therefore, we have:

$$X_{1,t+1}X_{2,t+1}=\left(\frac{U_1(r_{1,t})}{u^*}\cdot X_{1,t}\right)\left(\frac{u^*}{U_2(r_{2,t})}\cdot X_{2,t}\right)$$

Since the U_i are $(\alpha,\beta)-$subquadratic:

$$\left(\frac{r_{1,t}}{r_1^*}\right)^{2-\beta}\le\frac{U_1(r_{1,t})}{u^*}\le\left(\frac{r_{1,t}}{r_1^*}\right)^{\alpha}$$

and likewise for $\frac{u^*}{U_2(r_{2,t})}$. Note that the direction of the inequality is flipped from the definition because $r_{1,t}<r_1^*$. It follows that:

$$(X_{1,t}X_{2,t})^{\beta-1}\le X_{1,t+1}X_{2,t+1}\le(X_{1,t}X_{2,t})^{1-\alpha}$$

It is possible that $X_{1,t+1},X_{2,t+1}<1$ if $\beta<1$, which means that $r_{1,t+1}>r_1^*$ and $r_{2,t+1}<r_2^*$. In this case:

$$|\log X_{1,t+1}|+|\log X_{2,t+1}|\le(1-\beta)(|\log X_{1,t}|+|\log X_{2,t}|)$$

In the other case, where $\beta>1$ and $X_{1,t+1}>1$:

$$|\log X_{1,t+1}|+|\log X_{2,t+1}|\le(1-\alpha)(|\log X_{1,t}|+|\log X_{2,t}|)$$

We then conclude that:

$$|\log X_{1,t+1}|+|\log X_{2,t+1}|\le(1-\min(\alpha,\beta))|\,(\log X_{1,t}|+|\log X_{2,t}|)$$

Iterating from the initial rates gives that:

$$|\log X_{1,t}|+|\log X_{2,t}|\le(1-\min(\alpha,\beta))^t\,(|\log X_{1,0}|+|\log X_{2,0}|)$$

completing the theorem for 2 clients.

□

C Convergence with a Value Function

Minerva's value function (§5.3) depends heavily on the ABR algorithm used to compute it. As a result, it is not possible to always guarantee that it satisfies the convergence conditions outlined in Appendix B. Here, we consider a sample value function, and show that while it is not exactly convex, it can still be approximated as such.

First consider the following value function $V_h(r,b,e)$, which captures the optimal QoE possible for a video over the next h chunks, given a fixed link rate r, buffer b, and current bitrate e:

$$V_h(r,b,e)=\max_{e'}Q\left(e,\left[\frac{4e'}{r}-b\right]_+,e'\right)+\gamma V_{h-1}\left(r,\left[b-\frac{4e'}{r}\right]_+ +4,e'\right)$$

where Q is the QoE of a single chunk given in Eq. 1, and $0<\gamma\le 1$ is a discount factor and we have assumed a chunk duration of 4 seconds. The expression $[\cdot]_+$ is equivalent to $\max(\cdot,0)$. MPC [44] can be formulated in this way using $\gamma=1$ and $h=5$.

In the non-discounted case, $\gamma=1$, it can be seen that the optimal strategy involves switching between two adjacent bitrates e_i and e_{i+1} such that $e_i\le r\le e_{i+1}$. The client stays at bitrate e_i until it has enough buffer to switch to e_{i+1} for the remainder of the horizon. This incurs a smoothness penalty S only once over all h chunks. The

value function is then:

$$V_h(r) = \frac{aP(e_i) + (h-a)P(e_{i+1})}{h} + \frac{S}{h}$$

for some integer $a \leq h$. As $h \to \infty$, $V_h(r)$ approaches a linear interpolation of P, the perceptual quality. V_h thus approaches a concave function, but for any finite h is not concave. Fig. 4 shows an example undiscounted value functions over a horizon of 5 chunks at different starting buffer levels. The step-like nature of the curves in the figure prevent them from meeting the convergence conditions in Appendix B. However, there are two ways to handle this issue.

First, we can fit the value function with a function known to be meet convergence conditions. For example, the value functions in Fig. 4 can be approximated by exponential functions of the form: $f(r) = A - Be^{-Cr}$, for fitting parameters A,B,C. Though approximating the value function in this manner guarantees convergence, the convergence point may not be the true max-min QoE fair allocation. Alternatively, since the finite-horizon value function largely resembles a concave function, except for local step-like behavior, simply using the function as is may suffice. We find that this is the case for our video corpus: we use the actual function, instead of a fit, in Minerva's implementation, and find that it yields sufficiently strong results.

D Optimizing for Proportional Fairness

Minerva also allows optimizing for QoE fairness with different functional forms from Max-min fairness, such as α–fairness. Here, we consider α– fairness where $\alpha = 1$, known as proportional fairness.

The only change to Minerva comes in the weight-update step. The weight update step only requires each client to have some function of rate $Z_i(r)$; Minerva's decentralized algorithm modifies rates to achieve equality between all the $Z_i(r_i)$. When $Z_i = U_i$, the client's utility function, Minerva achieves max-min fairness. Using a different Z_i would optimize for a different notion of fairness. To illustrate, consider proportional fairness, which maximizes $\sum_i \log(U_i(r_i))$. The optimal rate allocation satisfies:

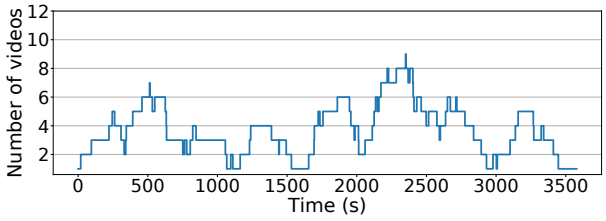
$$\frac{U_i'(r_i)}{U_i(r_i)} = \frac{U_j'(r_j)}{U_j(r_j)} \qquad \forall i,j$$

Setting $Z_i(r_i)$ to be some function of $U_i'(r_i)/U_i(r_i)$ pushes Minerva towards proportional fairness. However, Z_i must still be increasing and concave, so it must be chosen carefully based on the shape of the PQ curves. For example, $Z_i(r_i) = C - U_i'(r_i)/U_i(r_i)$ has the required properties for the utility curves in Fig. 1 and can be substituted for U_i in the weight update step. In particular, the normalization function must be computed using Z_i.

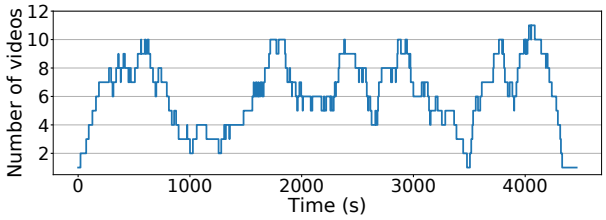
For proportional fairness, $U_i'(r_i)/U_i(r_i)$ should be equal for all i. If $\overline{P}(r)$ is the PQ curve averaged over chunks, then We compute $U_i'(r_i)$ by taking the numerical derivative of the PQ curve averaged over all chunks. On each weight update, we compute the expected QoE q, the *representative rate* $r_r = \overline{P}^{-1}(q)$, and finally $Z_i = C - U_i'(r_r)/q$. The choice of C is arbitrary and does not affect convergence.

E Minerva in a Dynamic Environment

In §8.3, the video start times were determined by a Poisson process such that the average number of videos playing simultaneously matched a given number, either 4 or 8, depending on the experiment. Videos have similar runtimes, between 270 and 300 seconds. Fig. 16a shows the number of videos sharing the link when videos were sampled randomly from the corpus, such that there were 4 playing simultaneously, on average. Fig. 16b shows a process targeting 8 videos sharing the link on average.



(a) 4 videos at a time on average, sampled uniformly from the corpus.



(b) 8 videos at a time on average, all identical (V11).

Figure 16: The number of videos sharing the link over time in the dynamic environment. All videos are between 4 and 5 minutes long.

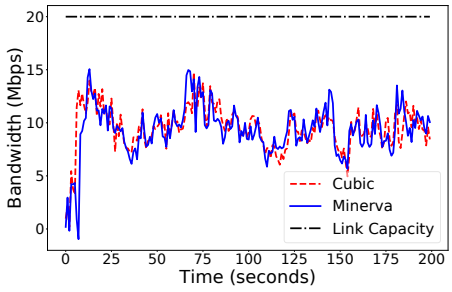


Figure 17: Achieved aggregate bandwidth across all video flows when 4 Minerva or Cubic flows are running on a 20Mbps link along with web cross traffic that consumes 10Mbps on average.

F Cross Traffic Experimental Supplement

We show an example of Minerva behavior when competing with a variable wide-area workload. For reference, we show Cubic competing with the same workload. Fig. 17 highlights the extent of variability in our workload and demonstrates that Minerva tracks Cubic's behavior closely in terms of aggregate bandwidth.

G Minerva with Two Providers

Fig. 15 demonstrates that, on average, two clients using different quality metrics, VMAF and PSNR, and their own respective normalization functions will split bandwidth approximately equally. The PSNR quality curves for our video corpus are shown in Fig. 18. Note that they are noticeably different from the VMAF curves in Fig. 1: the PSNR curves are on a scale of 53, instead of 100, and are generally flatter for bitrates above 1Mbit/s. This impacts the magnitude of gains Minerva achieves, since moving to a higher bitrate does not result in a large improvement in PSNR value. In particular, the difference between 720p and 1080p is only 2.26 on the PSNR scale, averaged over our entire corpus; by contrast, the difference

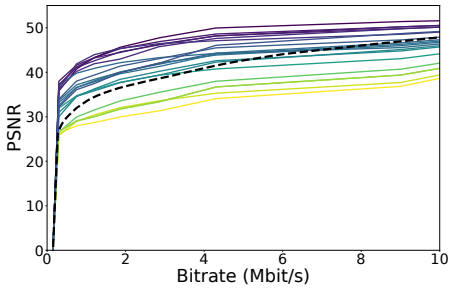
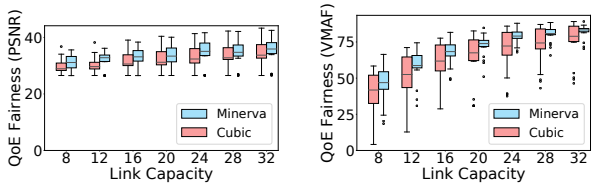


Figure 18: The average PSNR scores for the videos in our corpus, along with the normalization function (dotted).



(a) PSNR **(b) VMAF**

Figure 19: Providers both running Minerva on the same link achieve improvements in QoE fairness over Cubic regardless of which QoE metric they use.

in quality between the same bitrates measured with VMAF is 7.65. Even when accounting for the difference in scale between the two metrics, the marginal improvement in PSNR between adjacent bitrates is much less than the corresponding improvement in VMAF.

We now consider the QoE fairness of clients using Minerva to those using Cubic, in the presence of two providers. As described in §8.7, both providers run four videos each, sampled randomly from our corpus. Both providers run Minerva, with one using VMAF as its quality metric, while the other uses PSNR. As a separate baseline, we run the same videos over Cubic with an ABR module that uses either VMAF or PSNR. We run 20 different video combinations over a range of link rates. Fig. 19 shows that, regardless of which metric is used, both clients see improvements in QoE fairness: the median improvement is 6.1 VMAF points and 2.41 PSNR points for the respective providers. Note that the magnitude of improvements between the two clients are not comparable, since each metric has a different range and a different overall shape. However, the median PSNR improvement is larger than the 2.26-point PSNR difference between 720p and 1080p, suggesting that the improvements in PSNR are visually significant.