A Buffer-Based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service

Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell*, Mark Watson*
Stanford University, Netflix*
{huangty,rjohari,nickm}@stanford.edu, {mtrunnell,watsonm}@netflix.com

ABSTRACT

Existing ABR algorithms face a significant challenge in estimating future capacity: capacity can vary widely over time, a phenomenon commonly observed in commercial services. In this work, we suggest an alternative approach: rather than presuming that capacity estimation is required, it is perhaps better to begin by using *only* the buffer, and then ask *when* capacity estimation is needed. We test the viability of this approach through a series of experiments spanning millions of real users in a commercial service. We start with a simple design which directly chooses the video rate based on the current buffer occupancy. Our own investigation reveals that capacity estimation is unnecessary in steady state; however using simple capacity estimation (based on immediate past throughput) is important during the startup phase, when the buffer itself is growing from empty. This approach allows us to reduce the rebuffer rate by 10–20% compared to Netflix's then-default ABR algorithm, while delivering a similar average video rate, and a higher video rate in steady state.

Categories and Subject Descriptors

C.2.0 [**Computer Systems Organization**]: Computer-Communication Networks—*General*

Keywords

HTTP-based Video Streaming, Video Rate Adaptation Algorithm

1. INTRODUCTION

During the evening peak hours (8pm–1am EDT), well over 50% of US Internet traffic is video streamed from Netflix and YouTube [16, 17]. Unlike traditional video downloads that must complete fully before playback can begin, streaming video starts playing within seconds. Each video is encoded at a number of different rates (typically 235kb/s standard definition to 5Mb/s high definition) and stored on servers as separate files. The video client—running on a home TV, game console, web browser, DVD player, etc.—chooses which video rate to stream by monitoring network conditions and estimating the available network capacity. This process is referred to as *adaptive bit rate selection* or ABR.

SIGCOMM'14, August 17–22, 2014, Chicago, Illinois, USA.

ACM 978-1-4503-2836-4/14/08 ...$15.00.
http://dx.doi.org/10.1145/2619239.2626296.

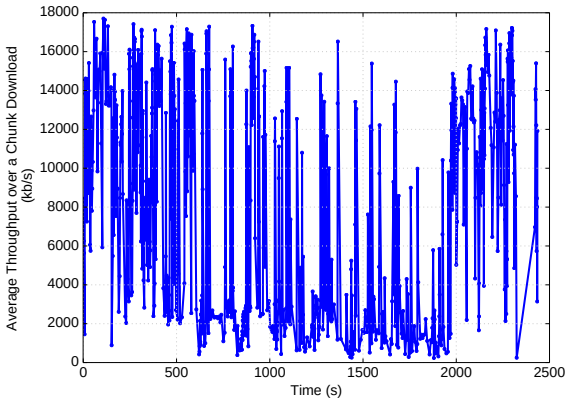


Figure 1: Video streaming clients experience highly variable end-to-end throughput.

ABR algorithms used by such services balance two overarching goals. On one hand, they try to maximize the video quality by picking the highest video rate the network can support. On the other hand, they try to minimize *rebuffering events* which cause the video to halt if the client's playback buffer goes empty.

It is easy for a streaming service to meet either one of the objectives on its own. To maximize video quality, a service could just stream at the maximum video rate $R_{\max}$ all the time. Of course, this would risk extensive rebuffering. On the other hand, to minimize rebuffering, the service could just stream at the minimum video rate $R_{\min}$ all the time—but this extreme would lead to low video quality. The design goal of an ABR algorithm is to *simultaneously* obtain high performance on both metrics in order to give users a good viewing experience [7].

One approach is to pick a video rate by estimating future capacity from past observations. In an environment with constant throughput, past observations are reliable to predict future capacity. However, in an environment with highly variable throughput, although past observations still provide valuable ballpark figures, accurate estimation of future capacity becomes challenging. Figure 1 is a sample trace reported by a Netflix video player, showing how the

measured throughput varies wildly from 17Mb/s to 500kb/s. Each point in the figure represents the average throughput when downloading a video chunk. This variation has a significant impact on customers: approximately 10% of our sessions experience at least this much variation, and 22% of sessions experience at least half as much variation.[1] Variation can be caused by many factors, such as WiFi interference, congestion in the network, congestion in the client (e.g. anti-virus software scanning incoming http traffic), or congestion at an overloaded video server.

In part due to highly variable throughput, current ABR algorithms often augment their capacity estimation with an "adjustment" based on the current level of the playback buffer [5, 20]. Informally, the idea is that this adjustment should make the rate selection more conservative when the buffer is at risk of underrunning, and more aggressive when the buffer is close to full. As we will see in Section 2, designing an optimal adjustment in a highly variable throughput environment is challenging; it is very hard to find an adjustment function that prevents rebuffering without being overly conservative. However, the notion of buffer-based adjustment used in current schemes is quite suggestive: note that the occupancy of the playback buffer is the primary state variable we are trying to manage. This inspires the following question: namely, can we take the design to its logical extreme, and choose the video rate based *only* on the playback buffer occupancy?

In this paper, we consider using *only* the buffer to choose a video rate, and then ask *when* capacity estimation is needed. We observe two separate phases of operation: a *steady-state* phase when the buffer has been built up, and a *startup* phase when the buffer is still growing from empty. Our analysis and experiments show that capacity estimation is not needed during the steady state. We can rely only on the current buffer occupancy to pick a video rate, allowing for a simple function to map current buffer occupancy to video rate. On the other hand, as we will see in Section 6, during the startup phase—just like the slow-start algorithm in TCP—the buffer occupancy carries little or no information about current network conditions. As a result, crude capacity estimation is helpful to quickly ramp up the video rate and drive the algorithm into the steady state.

In this paper, we show—both formally and through a deployment in the commercial Netflix service—that our algorithms can avoid unnecessary rebuffering events and yet achieve a high average video rate. We test this approach in a Netflix browser-based video player, a popular commercial streaming service, and present results from two A/B tests with over half a million real users each, on three continents, over two weekends during May-September 2013. Our experiments allow us to evaluate the viability of the buffer-based design. We find that *this design approach can reduce the rebuffer rate by 10–20% compared to Netflix's then-default ABR algorithm, while improving the steady-state video rate.*

In Section 2, we first dig into the implication of highly variable throughput on ABR algorithm design. This discussion motivates the buffer-based approach. In Section 3, we introduce the broad class of buffer-based algorithms (BBA), and identify the criteria to achieve our design goals in the ideal setting. In Section 4, we design a very simple baseline algorithm to test the viability of this approach in the steady-state. The baseline algorithm reduces the rebuffer rate by a promising 10–20% relative to a production algorithm. Nevertheless, the rebuffer rate is still larger than our empirical lower bound and delivers a lower average video rate than the control algorithm.

We identify two reasons for the lower performance. First, our baseline algorithm does not address variable bit-rate (VBR) video encoding; we adapt our algorithm with a simple fix to handle VBR in Section 5. Second, and more importantly, our baseline algorithm is optimized for steady-state. During the startup phase (the first few minutes of viewing), the buffer is close to empty and contains less information while in a transient phase. Although the performance of the baseline buffer-based algorithm suggests capacity estimation is not necessary in steady state, simple capacity estimation is useful in the startup phase. In Section 6, we validate this hypothesis by implementing techniques to improve video quality in the startup phase by estimating the immediate past throughput. Together, our two improvements maintain the reduction in rebuffer rate by approximately 10–20%, while improving the video rate during steady state, and leaving the average video rate essentially unchanged. Finally, in Section 7, we propose mechanisms to deal with temporary network outages and to minimize rate switching.

2. THE CHALLENGES OF A HIGHLY VARIABLE ENVIRONMENT

In an environment with stable capacity, past observations yield good estimates of future capacity. But if capacity is varying widely, estimating future capacity is much harder. Many techniques have been proposed to leverage the buffer occupancy to work with inaccurate capacity estimates. In this section, we first look into the dynamics of the playback buffer and understand how the buffer occupancy encodes the relation between the selected video rate and the system capacity. We then consider how the buffer occupancy is used to adjust inaccurate capacity estimates: essentially, the algorithm becomes more "aggressive" when the buffer is close to full, and more "conservative" when the buffer is close to empty. While appealing, we find that if capacity is *highly* variable (as we find it to be in practice), it is hard to prevent rebuffering events with only an adjustment to the capacity estimate.

However, the design of buffer-based adjustments is suggestive, and motivates our design. In particular, our design begins by using *only* the buffer occupancy to pick a video rate, and then considers *when* capacity estimation is needed. The pure buffer-based approach is sufficient when the buffer contains enough information about the past capacity trace, i.e., in steady state. On the other hand, simple capacity estimation proves valuable when the buffer contains little information, i.e., when the buffer is still growing from empty a few minutes after the session starts.

2.1 Dynamics of the Playback Buffer

Figure 2 shows the dynamics of the playback buffer in the client. The buffer occupancy is generally tracked in *seconds of video*. Every second, one second of video is removed from the buffer and played to the user. The buffer drains at unit rate (since one second is played back every second of real time). The client requests *chunks* of video from the server, each chunk containing a fixed duration of video (four seconds

[1] We define variation to be the ratio of 75th to 25th percentile throughput; which is 5.6 for this trace.

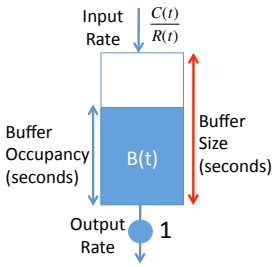


Figure 2: The relationship between system capacity, $C(t)$, and video rate, $R(t)$, in a video playback buffer.

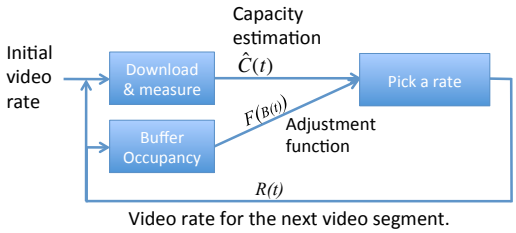


Figure 3: Current practice adjusts the estimation based on the buffer occupancy.

per chunk in our service). The higher the video rate, the larger the chunk (in bytes).

If the ABR algorithm *overestimates* the capacity and picks a video rate, $R(t)$, that is greater than the system capacity, $C(t)$, then new data is put into the buffer at rate $C(t)/R(t) < 1$ and so the buffer decreases. Put another way, if more than one chunk is played before the next chunk arrives, then the buffer is depleted. If the ABR algorithm keeps requesting chunks that are too big for the network to sustain (i.e., the video rate is too high), eventually the buffer will run dry, playback freezes and we see the familiar "Rebuffering..." message on the screen.

2.2 Working with Inaccurate Estimates

Many techniques have been proposed to work with inaccurate estimates, by incorporating information about the playback buffer. Some leverage control theory to adjust the capacity estimation based on the buffer occupancy [5, 20], some smooth the quality degradation according to the buffer occupancy [15], and some randomize chunk scheduling depending on the buffer occupancy to have better samples of the channel [10].

At a high level, we can capture existing approaches using the abstract design flow in Figure 3. The client measures how fast chunks arrive to estimate capacity, $\hat{C}(t)$. The estimate is optionally supplemented with knowledge of the buffer occupancy, which we represent by an *adjustment* factor $F(B(t))$, a function of the playback buffer occupancy. The selected video rate is $R(t) = F(B(t))\hat{C}(t)$; different designs use different adjustment functions $F(\cdot)$.

When the buffer contains many chunks, $R(t)$ can safely deviate from $C(t)$ without triggering a rebuffer. The client can "aggressively" try to maximize the video quality by picking $R(t) = \hat{C}(t)$.

But when the buffer is low, the client should be more "conservative", deliberately underestimating capacity so as to pick a lower video rate and quickly replenish the buffer. In this case, designing the adjustment function is much harder, as the following analysis shows. Consider the case when there is only one chunk in the buffer. The requested chunk (V seconds) must arrive before the current chunk plays, else the buffer will run dry. In other words, we require $VR(t)/C(t) < B(t)$, where $VR(t)$ is the chunk size in bytes. Thus, the selected video rate $R(t)$ needs to satisfy:

$$R(t) < \left(\frac{B(t)}{V}\right) C(t)$$

to prevent rebuffers. Replacing the selected video rate $R(t)$ with $F(B(t))\hat{C}(t)$ in the above inequality, we get the following requirement on $F(B(t))$ to avoid rebuffers:

$$F(B(t)) < \left(\frac{B(t)}{V}\right)\left(\frac{C(t)}{\hat{C}(t)}\right) \text{ for all } t. \quad (1)$$

This tells us we must pick $F(V)$ to be smaller than the *worst case ratio* of $C(t)$ to $\hat{C}(t)$. Unfortunately, $C(t)/\hat{C}(t)$ is tiny if the throughput is varying wildly; and since we have to choose F without knowing the actual capacity that will be observed, it leads to a very conservative algorithm. For example, in Figure 1, the ratio $C(t)/\hat{C}(t)$ can be as small as 0.03 (500 kb/s $< C(t) <$ 17 Mb/s). In other words, for this session, we need to pick $F(V) \leq 0.03$ to prevent rebuffers, and the video rate will be just 3% of the rate we could pick with an accurate estimate. Worse, if $F(.)$ makes us pick a rate lower than the minimum video rate available, the constraint becomes impossible to meet.

In practice, large throughput variation within a session is not uncommon. A random sample of 300,000 Netflix sessions shows that roughly 10% of sessions experience a median throughput *less than half of the 95th percentile throughput.* When designing an ABR algorithm, the service provider needs to choose a $F(\cdot)$ that works well for *all* customers, with both stable and variable throughput.

An example from a Netflix session illustrates the problem. Figure 4 shows an ABR algorithm that is not conservative enough; it keeps requesting video at too high a rate after the capacity has dropped. The client rebuffers and freezes playback for 200 seconds. But notice that the rebuffer is entirely unnecessary because the available capacity $C(t)$ is above $R_{\min}$ for the entire time series. In fact, if the network capacity is always greater than the lowest video rate $R_{\min}$, i.e., $C(t) > R_{\min}, \forall t > 0$, there *never* needs to be a rebuffering event — the algorithm can simply pick $R(t) = R_{\min}$ so that $C(t)/R(t) > 1, \forall t > 0$ and the buffer keeps growing. The main reason the client does not switch is that it overestimates the current capacity, and the adjustment function is not small enough to offset the difference. As a result, despite the fact that capacity is sufficient to sustain $R_{\min}$, the client does not find its way to that video rate in time.

2.3 The Buffer-Based Approach

The discussion above is suggestive. Despite the challenge of finding the right adjustment, using buffer-based adjustments in algorithms is quite appealing, because the playback buffer is the exact state variable an ABR algorithm is trying to control. For example, the easiest way to ensure that the algorithm never unnecessarily rebuffers is to simply re-

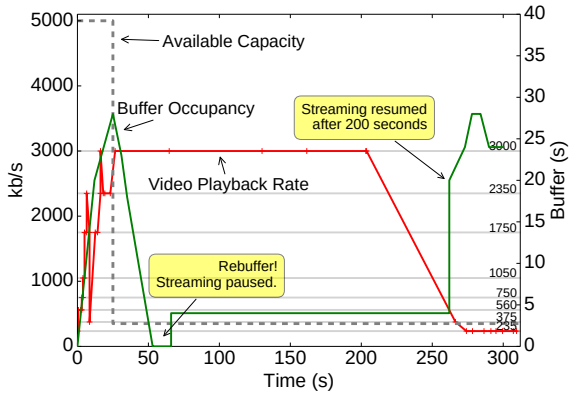


Figure 4: *Being too aggressive*: A video starts streaming at 3Mb/s over a 5Mb/s network. After 25s the available capacity drops to 350 kb/s. Instead of switching down to a lower video rate, e.g., 235kb/s, the client keeps playing at 3Mb/s. As a result, the client rebuffers and does not resume playing video for 200s. Note that the buffer occupancy was not updated during rebufferings.

quest rate $R_{\min}$ when the buffer approaches empty, allowing the buffer to grow as long as $C(t) > R_{\min}$. Note in particular that in the scenario in the preceding section, this approach would have avoided a rebuffering event. On the other hand, as the buffer grows, it is safe to increase $R(t)$ up to the maximum video rate as the buffer approaches full. This motivates our design: our starting point is a simple algorithm design that chooses the video rate based only on the playback buffer.

Inspired by this discussion, we design our algorithms as follows. First, we focus on a pure buffer-based design: we select the video rate directly as a function of the current buffer level. As we find, this approach works well when the buffer adequately encodes information about the past history of capacity. However, when the buffer is still growing from empty (during the first few minutes of a session), it does not adequately encode information about available capacity. In this phase, the pure buffer-based design can be improved by leveraging a capacity estimate.

We call this design the *buffer-based approach*. This design process leads to two separate phases of operation: During the steady-state phase, when the buffer encodes adequate information, we choose the video rate based only on the playback buffer. During the startup phase, when the buffer contains little information, we augment the buffer-based design with capacity estimation. In this way, our design might be thought of as an "inversion" of Figure 3: namely, we begin by using only the playback buffer, and then "adjust" this algorithm using capacity estimation where needed.

3. BUFFER-BASED ALGORITHMS

We say that an ABR algorithm is *buffer-based* if it picks the video rate as a function of the current buffer occupancy, $B(t)$. The design space for this class of algorithms is expressed by the buffer-rate plane in Figure 5. The region between $[0, B_{\max}]$ on the buffer-axis and $[R_{\min}, R_{\max}]$ on the rate-axis defines the feasible region. Any curve $f(B)$ on the plane within the feasible region defines a *rate map*, a

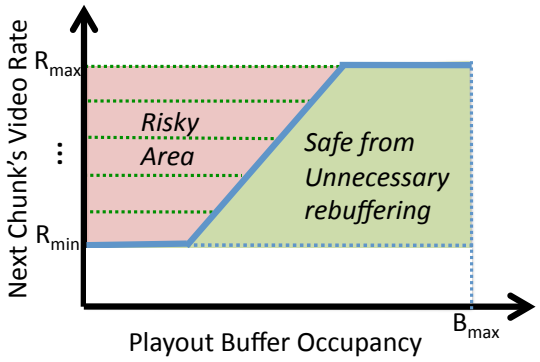


Figure 5: Video rate as a function of buffer occupancy.

function that produces a video rate between $R_{\min}$ and $R_{\max}$ given the current buffer occupancy.

3.1 Theoretical Criteria for Design Goals

From this feasible region, our goal is to find a class of mapping functions that can: (1) avoid unnecessary rebufferings, and (2) maximize average video rate.

To start with, we make the following simplifying assumptions:

1. The chunk size is infinitesimal, so that we can change the video rate continuously.
2. *Any* video rate between $R_{\min}$ and $R_{\max}$ is available.
3. Videos are encoded at a constant bit-rate (CBR).
4. Videos are infinitely long.

We can show that any rate maps that are (1) *continuous* functions of the buffer occupancy B; (2) *strictly increasing* in the region $\{B : R_{\min} < f(B) < R_{\max}\}$; and (3) pinned at both ends, i.e., $f(0) = R_{\min}$ and $f(B_{\max}) = R_{\max}$, will meet the two design goals. In other words, we can achieve our goal by picking any rate map that increases the video rate from lowest to highest as the buffer increases from empty to full. The mapping function in Figure 5 is one such example. We leave the formal proof in our technical report [8], and we summarize the proof here:

No unnecessary rebuffering: As long as $C(t) \geq R_{\min}$ for all t and we adapt $f(B) \to R_{\min}$ as $B \to 0$, we will never unnecessarily rebuffer because the buffer will start to grow before it runs dry.

Average video rate maximization: As long as $f(B)$ is (1) increasing and (2) eventually reaches $R_{\max}$, the average video rate will match the average capacity when $R_{\min} < C(t) < R_{\max}$ for all $t > 0$.

Next, we explore how to remove the assumptions above, then validate the approach with the Netflix deployments in Section 4, 5 and 6.

3.2 Real World Challenges

In practice, the chunk size is finite (V seconds long) and a chunk is only added to the buffer after it is downloaded. To avoid interruption, we always need to have at least one chunk available in the buffer. To handle the finite chunk size, as well as some degree of variation in the system, we shift the rate map to the right and create an extra *reservoir*, noted as r. When the buffer is filling up the reservoir, i.e., $0 \leq B \leq r$, we request video rate $R_{\min}$. Once the reservoir is reached, we then increase the video rate according to $f(B)$.

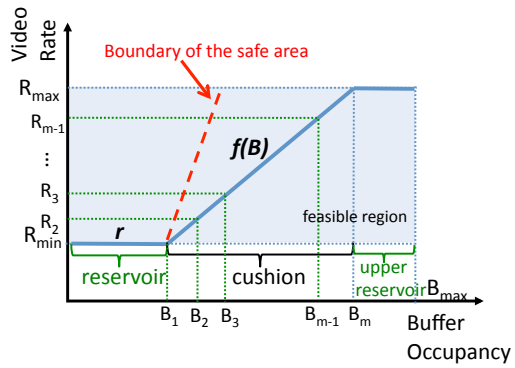


Figure 6: The rate map used in the BBA-0 buffer-based algorithm.

Also because of the finite chunk size, the buffer does not stay at $B_{\max}$ even when $C(t) \geq R_{\max}$; thus, we should allow rate map to reach $R_{\max}$ before $B_{\max}$. We call the buffer between the reservoir and the point where $f(B)$ first reaches $R_{\max}$ the *cushion*, and the buffer after the cushion the *upper reservoir*.

Since many video clients have no control over TCP sockets and they cannot cancel an ongoing video chunk download, we can only pick a new rate when a chunk *finishes* arriving. If the network suddenly slows down while we are in the middle of downloading a chunk, the buffer might run dry before we get the chance to switch to a lower rate. Thus, we need to aim to maintain the buffer level to be above the reservoir r, so that there is enough buffer to absorb the variation caused both by the varying capacity and by the finite chunk size. As a result, $f(B)$ should be designed to ensure a chunk can always be downloaded before the buffer shrinks into the reservoir area. Based on these observations, we say $f(B)$ operates in the *safe* area if it always picks chunks that will finish downloading before the buffer runs below r, when $C(t) \geq R_{\min}$ for all t. In other words, $Vf(B)/R_{\min} \leq (B - r)$. Otherwise, $f(B)$ is in the *risky* area.

Overall, the class of functions that we consider take the piecewise form described in Figure 6. We illustrate there the reservoir, the cushion, and the upper reservoir. We also illustrate the notion of safety described in the previous paragraph: we plot the boundary of the safe area as the red dashed line in the figure. Any $f(B)$ below the boundary will be a safe choice.

In Section 4, we test this concept by deploying a baseline algorithm with fixed-size reservoir and cushion.

4. THE BBA-0 ALGORITHM

To test the buffer-based approach developed in Section 3, we first construct a baseline algorithm with a relatively simple and naive rate map. We start the design with a piecewise function as shown in Figure 6. We then determine the size of reservoir, cushion, and upper reservoir, as well as the shape of the rate map. We implement the algorithm in Netflix's browser-based player, which happens to have a 240 second playback buffer and the convenient property that it downloads the ABR algorithm at the start of the video session.

As discussed in Section 3, the size of reservoir needs to be at least one chunk (4 seconds in our testing environment) to absorb the buffer variation caused by the finite chunk size. However, since the algorithm is tested in a production environment that streams VBR-encoded video, the size of reservoir also needs to be big enough to absorb the buffer variation caused by the VBR encoding. As the first baseline algorithm, we set the size of reservoir to be a large and fixed-size value, 90 seconds. We thought a 90s reservoir is big enough to absorb the variation from VBR, allowing us to focus on testing the approach developed in Section 3.

The size of cushion is defined as the buffer distance between B_1 and B_m, as shown in Figure 6. Since the buffer distance between neighboring rates affects the frequency of rate switches, we maximize the size of cushion while leaving some room for the upper reservoir. As a result, we let the rate map reaches $R_{\max}$ when the buffer is 90% full (216 seconds). In other words, we set the cushion to be 126 seconds (between 90 to 216 seconds) and the upper reservoir to be 24 seconds (between 216 to 240 seconds). To further maximize the distance between each pair of neighboring rates, we use a linear function to increase the rate between $R_{\min}$ and $R_{\max}$. The resulting $f(B)$ is a piecewise linear function, which stays in the safe area defined in Section 3.

Note that a rate map by itself does not fully define the algorithm: the rate map is continuous, while streamed video rates are discrete, $R_{\min}, R_2, R_3 ... R_{m-1}, R_{\max}$. We therefore adapt the rate according to Algorithm 1, following a simple rule: stay at the current video rate as long as the rate suggested by the rate map does not cross the next higher (Rate_+) or lower (Rate_-) discrete video rate. If either "barrier" is hit the rate is switched up or down (respectively) to a new discrete value suggested by the rate map. In this way, the buffer distance between the adjacent video rates provides a natural cushion to absorb rate oscillations, making the video rate a little "sticky". This algorithm, together with the rate map we just defined, constructs our first buffer-based algorithm. We call this algorithm *BBA-0* since it is the simplest of our buffer-based algorithms.

4.1 Experiments

We implemented the BBA-0 algorithm in Netflix's browser-based player. As mentioned, the video player has a 240s playback buffer and downloads the ABR algorithm at the start of the video session. Although this player enjoys a bigger buffer than players on embedded devices, it does not have visibility into, or control of, the network layer. We randomly picked three groups of users from around the world to take part in the experiments between September 6th (Friday) and 9th (Monday), 2013.

Group 1 is our *Control* group and they use Netflix's then-default ABR algorithm.[2] The *Control* algorithm has steadily improved over the past five years to perform well under many conditions. The *Control* algorithm directly follows the design in Figure 3: it picks a video rate primarily based on capacity estimation, with buffer occupancy as a secondary signal. It is representative of how video streaming services work; e.g. Hulu [9] and YouTube [21] are based on capacity estimation. Netflix traffic represents 35% of the US peak Internet traffic and they serve 40 million users world-wide. For these reasons, we believe the Netflix *Control* algorithm is a reasonable algorithm to compare against.

Group 2 always stream at $R_{\min}$, and we call this degenerate algorithm $R_{\min}$ *Always*. Always operating at the lowest

[2]The ABR algorithm in commercial services keeps evolving, and so Netflix's current algorithm is now different.

Algorithm 1: Video Rate Adaptation Algorithm

Input: $\text{Rate}_{\text{prev}}$: The previously used video rate
Buf_{now}: The current buffer occupancy
r: The size of reservoir
cu: The size of cushion
Output: $\text{Rate}_{\text{next}}$: The next video rate

if $\text{Rate}_{\text{prev}} = R_{\max}$ **then**
  $\text{Rate}_{+} = R_{\max}$
else
  $\text{Rate}_{+} = \min\{R_i : R_i > \text{Rate}_{\text{prev}}\}$

if $\text{Rate}_{\text{prev}} = R_{\min}$ **then**
  $\text{Rate}_{-} = R_{\min}$
else
  $\text{Rate}_{-} = \max\{R_i : R_i < \text{Rate}_{\text{prev}}\}$

if $\text{Buf}_{\text{now}} \leq r$ **then**
  $\text{Rate}_{\text{next}} = R_{\min}$
else if $\text{Buf}_{\text{now}} \geq (r + cu)$ **then**
  $\text{Rate}_{\text{next}} = R_{\max}$
else if $f(\text{Buf}_{\text{now}}) \geq \text{Rate}_{+}$ **then**
  $\text{Rate}_{\text{next}} = \max\{R_i : R_i < f(\text{Buf}_{\text{now}})\}$;
else if $f(\text{Buf}_{\text{now}}) \leq \text{Rate}_{-}$ **then**
  $\text{Rate}_{\text{next}} = \min\{R_i : R_i > f(\text{Buf}_{\text{now}})\}$;
else
  $\text{Rate}_{\text{next}} = \text{Rate}_{\text{prev}}$;
return $\text{Rate}_{\text{next}}$;

video rate minimizes the chances of the buffer running dry, giving us a lower bound on the rebuffer rate to compare new algorithms against. For most sessions $R_{\min} = 560kb/s$, but in some cases it is $235kb/s$.[3]

Group 3 uses our new BBA-0 algorithm.

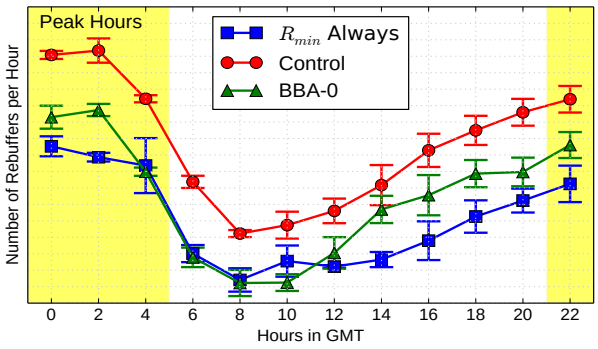
All three user groups are distributed similarly across ISPs, geographic locations, viewing behaviors and devices. The only difference between the three groups of clients is the rate selection algorithm; they share the same code base for other mechanisms, such as prebuffering, CDN selection, and error handling. As a result, all three groups share similar join delay and error rate, allowing us to concentrate on the quality metrics during playback.

Even though testing against a range of other complex algorithms would not be possible in this testing environment, it's unprecedented to be able to report video performance results from a huge commercial service, such as Netflix, and we believe the insight it offers into a real system is invaluable. During our experiments each group of users viewed roughly 120,000 hours of video. To compare their performance, we measure the overall number of rebuffers per playhour and the average delivered video rate in each group.

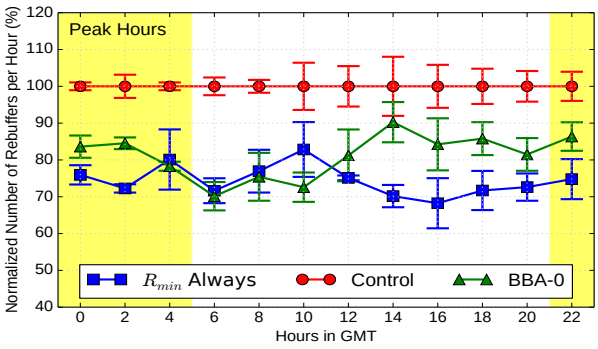
4.2 Results

Rebuffer Rate. Figure 7(a) plots the number of rebuffers per playhour throughout the day. Figure 7(b) simplifies a visual comparison between algorithms by normalizing the average rebuffer rate to the *Control* group in each two-hour

[3]In our service, R_{min} is normally $235kb/s$. However, most customers can sustain 560kb/s, especially in Europe. If a user historically sustained 560kb/s we artificially set $R_{min} = 560kb/s$ to avoid degrading the video experience too far. The mechanism to pick $R_{\min}$ is the same across all three test groups.



(a) Number of rebuffers per playhour during the day.



(b) Normalized number of rebuffers per playhour, normalized to the average rebuffer rate of *Control* in each two hour period.

Figure 7: Number of rebuffers per playhour for the *Control*, $R_{\min}$ *Always*, and BBA-0 algorithms. The error bars represent the variance of rebuffer rates from different days in the same two-hour period.

period. Peak viewing hours for the USA are highlighted in yellow. Error bars represent the variance of rebuffer rates from different days in the same two-hour period. The $R_{\min}$ *Always* algorithm provides an empirical lower bound on the rebuffer rate. Note that because the users in the three groups are different and their environments are not exactly the same, $R_{\min}$ *Always* only approximates the lower bound for the other groups. The first thing to notice from the figure is that $R_{\min}$ *Always* and BBA-0 always have a lower rebuffer rate than the *Control* algorithm. The difference between the *Control* algorithm and the $R_{\min}$ *Always* algorithm suggests that 20–30% of the rebuffers might be caused by poor choice of video rate.

During the middle-of-night period in the USA just after peak viewing (6am–12pm GMT), BBA-0 matches the $R_{\min}$ *Always* lower bound very closely. At 10am GMT, even though BBA-0 has a lower average rebuffer rate than $R_{\min}$ *Always*, the difference is not statistically significant.[4] These two algorithms perform equally during this off-peak period, because the viewing rate is relatively low, overall Internet usage is low, and the network capacity for individual sessions does not change much. The rebuffer rate during these

[4]The hypothesis of BBA-0 and $R_{\min}$ *Always* share the same distribution is not rejected at the 95% confidence level (p-value = 0.25).

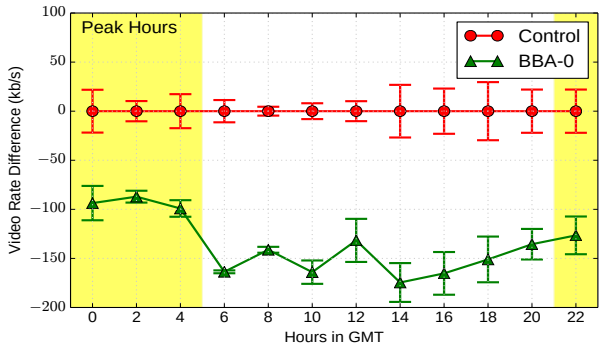


Figure 8: Comparison of video rate between *Control* and BBA-0. The error bars represent the variance of video rates from different days in the same two-hour period. The Y-axis shows the difference in the delivered video rate between *Control* and BBA-0.

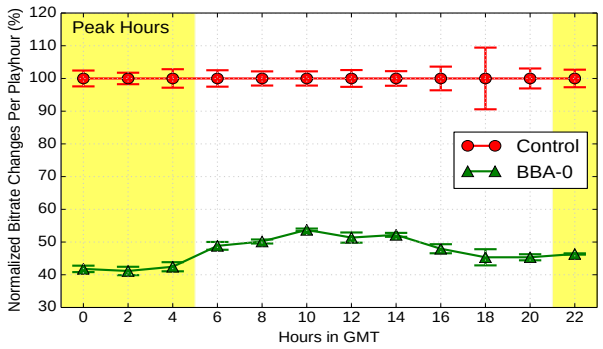


Figure 9: Average video switching rate per two hour window for the *Control* and *BBA-0* algorithms. The numbers are normalized to the average switching rate of the *Control* for each window.

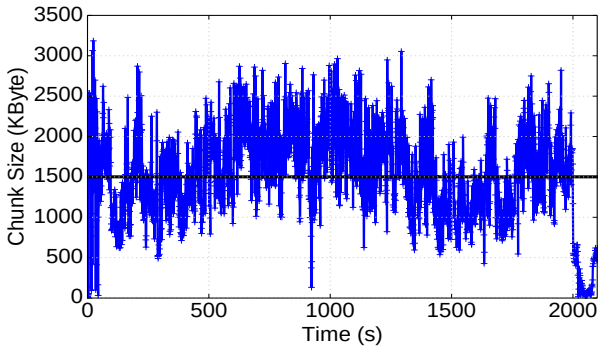


Figure 10: The size of 4-second chunks of a video encoded at an average rate of 3Mb/s. Note the average chunk size is 1.5MB (4s times 3Mb/s).

hours is dominated by random local events, such as WiFi interference, instead of congested networks.

During peak hours, the performance with BBA-0 is significantly worse than with the $R_{\min}$ *Always* algorithm. Nevertheless, the BBA-0 algorithm consistently has a 10–30% lower rebuffer rate than the *Control* algorithm. This performance difference is encouraging given the extremely simple nature of the BBA-0 algorithm. Still, we hope to do better. In Section 5 and 6, we will develop techniques to improve the rebuffer rate of buffer-based algorithms.

Video Rate. Figure 8 shows the difference in the delivered video rate between *Control* and BBA-0. The daily average bitrate for the *Control* algorithm for each ISP can be found in the Netflix ISP Speed Index [18]. Since $R_{\min}$ *Always* always streams at $R_{\min}$ (except when rebuffering), its delivered video rate is a flat line and is excluded from the figure. The BBA-0 algorithm is roughly 100kb/s worse than the *Control* algorithm during peak hours, and 175kb/s worse during off-peak hours. There are two main reasons for the degradation in video quality. First, our BBA-0 algorithm uses a large and fixed-size reservoir to handle VBR, while the size of reservoir should be adjusted to be just big enough to absorb the variation introduced by VBR. Second, and more significantly, while the reservoir is filling up during the startup period, our BBA-0 algorithm always requests video at rate $R_{\min}$. Given that we picked a 90s reservoir, it downloads 90 seconds worth of video at rate $R_{\min}$, which is a non-negligible fraction of the average session length. We will address both issues in Section 5 and 6.

Video Switching Rate. Since our BBA-0 algorithm picks the video rate based on the buffer level, we can expect the rate to fluctuate as the buffer occupancy changes. However, Algorithm 1 uses the distance between adjacent video rates to naturally cushion, and absorb, rate oscillations. Figure 9 compares BBA-0 with the *Control* algorithm. Note the numbers are normalized to the average switching rate of the Control group for each two-hour period. The BBA-0 algorithm reduces the switching rate by roughly 60% during peak hours, and by roughly 50% during off-peak hours.

In summary, BBA-0 confirms that we can reduce the rebuffer rate by focusing on buffer occupancy. The results also show that the buffer-based approach is able to reduce the video switching rate. However, BBA-0 performs worse on video rate compared to the *Control* algorithm. In the next section, we will develop techniques to improve both rebuffer rate and video rate by considering the VBR encoding scheme.

5. HANDLING VARIABLE BITRATE (VBR)

In Section 4, the BBA-0 algorithm attempts to handle VBR by setting the reservoir size to a large and somewhat arbitrary value. Although we are able to get a significant reduction in rebuffering compared to the *Control*, there is still room to improve when comparing to the empirical lower bound. In addition, the average video rate achieved by the BBA-0 algorithm is significantly lower than the *Control* algorithm. In this section, we will discuss techniques to improve both rebuffer rate and video rate by taking the encoding scheme into consideration. A key advance is to *design* the reservoir based on the instantaneous encoding bitrate of the stream being delivered.

In practice, most of the video streaming services encode their videos in variable bitrate (VBR). VBR encodes static scenes with fewer bits and active scenes with more bits, while maintaining a consistent quality throughout the video. VBR encodings allow more flexibility and can use bits more efficiently. When a video is encoded in VBR at a nominal video rate, the nominal rate represents the *average* video rate, and

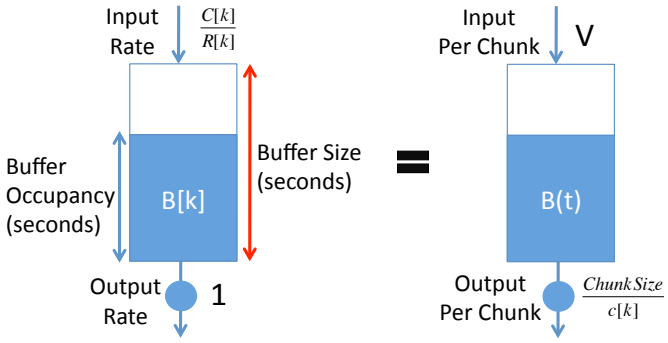


Figure 11: Two equivalent models of the streaming playback buffer.

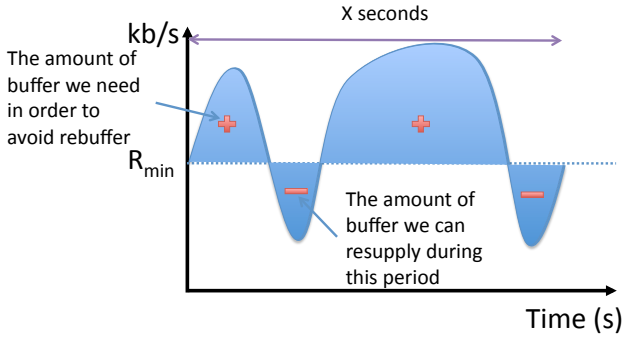


Figure 12: Reservoir calculation: We calculate the size of the reservoir from the chunk size variation.

the instantaneous video rate varies around the average value. As a result, the chunk size will not be uniformly identical in a stream of a given rate. Figure 10 shows the size of 4-second chunks over time from a production video (*Black Hawk Down*) encoded at 3 Mb/s. The black line represents the average chunk size. As we can see from the figure, the variation on chunk size can be significant within a single video rate.

Given the variation on chunk size, we need to take the size of each chunk into consideration and re-consider the buffer dynamics under VBR. Because we can only select video rates on a chunk-by-chunk basis, it is useful to consider the buffer dynamics when observed *at the time points when a chunk finishes*, as shown in Figure 11. Let $r[k]$ be the video rate selected for the k-th chunk and $c[k]$ be the average system capacity during the download of the k-th chunk. For the k-th chunk from the stream of nominal video rate r, we denote the chunk size as $Chunk[r][k]$. Since each chunk still contains V seconds of video, the buffer now drains $Chunk[r][k]/c[k]$ seconds while it fills with V seconds of video.

5.1 Reservoir Calculation

Since the instantaneous video rate can be much higher than the nominal rate in VBR, we could still encounter a rebuffer event even when the capacity $c[k]$ is exactly equal to $R_{\min}$, unless we have enough buffer to absorb the buffer oscillation caused by the variable chunk size. Thus, the size of reservoir should be big enough to ensure the client can continue playing at $R_{\min}$ when $c[k] = R_{\min}$.

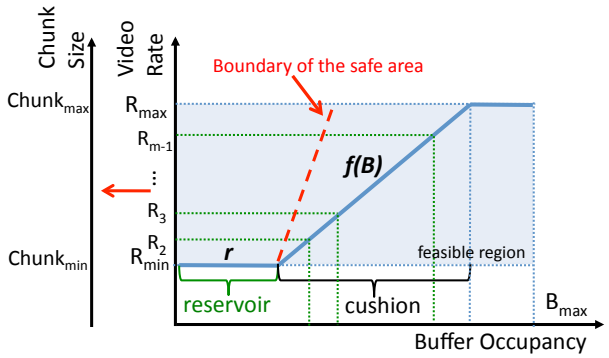


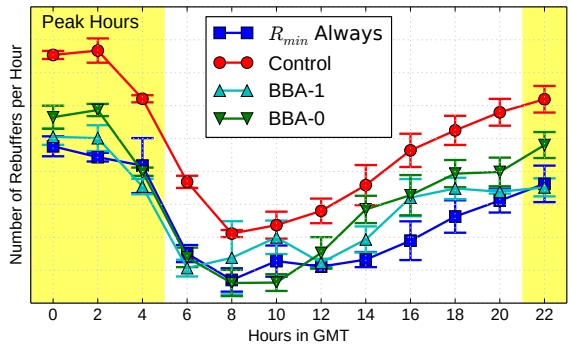
Figure 13: Handling VBR with chunk maps. To consider variable chunk size, we generalize the concept of rate maps to chunk maps by transforming the Y-axis from video rates to chunk sizes.

Assuming $c[k] = R_{\min}$, when the chunk size is larger than the average, $VR_{\min}$, the video client will consume more video in the buffer than the input. On the other hand, when the chunk size is lower than the average, the buffer is consumed more slowly than the input and the buffer occupancy will increase. Thus, by summing up the amount of buffer the client will consume minus the amount it can resupply during the next X seconds, we can figure out the amount of reservoir we need. We dynamically adjust the reservoir based on this prospective calculation over the lifetime of the stream. X should be set at least as the size of the playout buffer, since users expect the service to continue for that period even when bandwidth drops. Figure 12 summarizes how the calculation is done. In the implementation, we set X as twice of the buffer size, i.e., 480 seconds. The calculated reservoir size depends highly on the specific video and the playing segment. For example, when playing static scenes such as opening credits, since they are encoded with very few bits, the calculated reservoir size is negative; when playing active scenes that are encoded with much more bits, the calculated reservoir size can be even larger than half the buffer size (120 seconds). As a practical matter, we bound the size of reservoir to be between 8 seconds to 140 seconds.

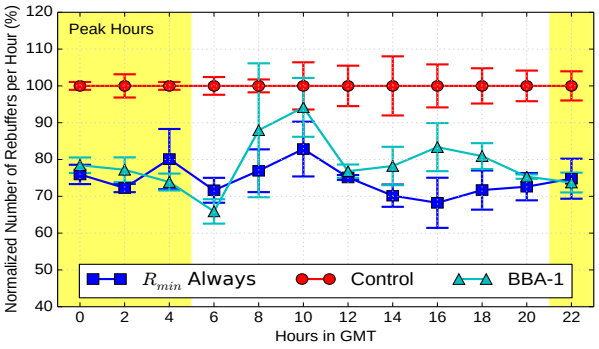
5.2 Chunk Map

Since the buffer dynamics now depend on the chunk size of the upcoming video segments instead of the video rate, it makes more sense to map the buffer occupancy to the chunk size directly. In other words, we can generalize the design space and change it from the buffer-rate plane to the buffer-chunk plane as shown in Figure 13. Each curve in the figure now defines a *chunk map*, which represents the maximally allowable chunk size according to the buffer occupancy. In the figure, the feasible region is now defined between $[0, B_{\max}]$ on the buffer-axis and $[Chunk_{\min}, Chunk_{\max}]$ on the chunk-axis, where $Chunk_{\min}$ and $Chunk_{\max}$ represent the average chunk size in $R_{\min}$ and $R_{\max}$, respectively.

We can now generalize Algorithm 1 to use the chunk map: the algorithm stays at the current video rate as long as the chunk size suggested by the map does not pass the size of the next upcoming chunk at the next highest available video rate (Rate_+) or the next lowest available video rate (Rate_-). If either of these "barriers" are passed, the rate is switched up or down, respectively. Note that by using the chunk map,



(a) Number of rebuffers per playhour throughout the day.



(b) Normalized number of rebuffers per playhour. Each percentage is normalized to the average rebuffer rate of the *Control* algorithm in a two-hour period.

Figure 14: The BBA-1 algorithm achieves close-to-optimal rebuffer rate, especially during the peak hours.

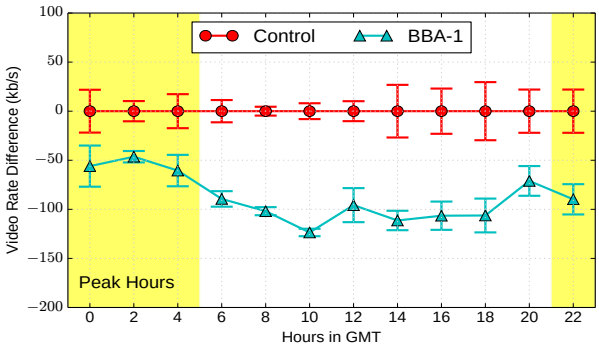


Figure 15: The BBA-1 algorithm improved video rate by 40–70 kb/s compare to BBA-0, but still 50–120 kb/s away from the *Control*.

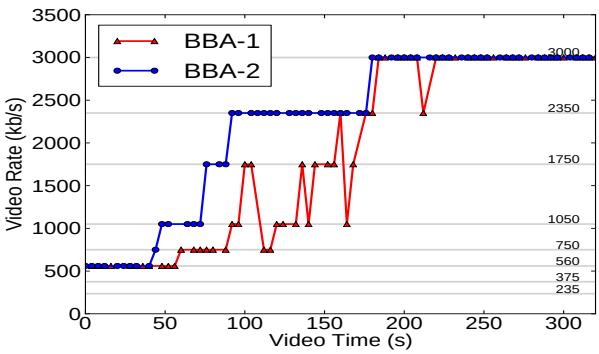


Figure 16: Typical time series of video rates for BBA-1 (red) and BBA-2 (blue). BBA-1 follows the chunk map and ramps slowly. BBA-2 ramps faster and reaches the steady-state rate sooner.

we no longer have a fixed mapping between buffer levels and video rates. This could result in a higher frequency of video rate switches. We will explore techniques to address this issue in Section 7.

5.3 Results

We use the same setup as in Section 4. We select the same number of users in each group to use our VBR-enabled buffer-based algorithm, which dynamically calculates the reservoir size and uses a chunk map. We will refer to the algorithm as *BBA-1* in the following, as it is our second iteration of the buffer-based algorithm. This experiment was conducted along with the experiment in Section 4 between September 6th (Friday) and 9th (Monday), 2013.

Figure 14(a) shows the rebuffer rate in terms of number of rebuffers per playhour, while Figure 14(b) normalizes to the average rebuffer rate of the *Control* in each two-hour period. We can see from the figure that the BBA-1 algorithm comes close to the optimal line and performs better than the BBA-0 algorithm. BBA-1 has a lower average rebuffer rate than R_{min} *Always* during 4–6am GMT, but the difference is not statistically significant.[5] The improvement over the *Control* algorithm is especially clear during peak hours, where the BBA-1 algorithm provides a 20–28% improvement in the rebuffer rate.

Figure 15 shows the difference in the average video rate between the *Control*, BBA-0, and BBA-1 algorithms. As shown in Figure 15, the BBA-1 algorithm also improves the video rate compared to BBA-0 by 40–70kb/s on average, although it is still 50–120kb/s away from the *Control* algorithm. This discrepancy in video rate comes from the startup period, when the buffer is still filling up. If we compare the average video rate of the first 60 seconds between the BBA-1 algorithm and the *Control* algorithm, the BBA-1 algorithm achieves 700kb/s less than the *Control*. Before the client builds up its buffer to the size of the reservoir, the BBA-1 algorithm will always request for $R_{\min}$, as it is the only *safe* rate given the buffer occupancy. In the next section, we will further improve the video rate by entering into the *risky* area and develop techniques to minimize the risk.

6. THE STARTUP PHASE

As discussed in the previous section, most of the differences in video rate between BBA-1 and the *Control* algorithm can be accounted for by the startup phase, i.e., after

[5]The hypothesis of BBA-1 and $R_{\min}$ *Always* share the same distribution is not rejected at the 95% confidence level (p-value = 0.74).

starting a new video or seeking to a new point.[6] During the startup phase, the playback buffer starts out empty and carries no useful information to help us choose a video rate. BBA-1 follows the usual chunk map, starting out with a low video rate since the buffer level is low. It gradually increases the rate as the buffer fills, as shown by the red line in Figure 16. BBA-1 is too conservative during startup. The network can sustain a much higher video rate, but the algorithm is just not aware of it yet.

In this section, we test the following hypothesis. During the startup, we can improve the video rate by entering into the risky area; in the steady state, we can improve both video rate and rebuffer rate by using a chunk map. Our next algorithm, BBA-2, tries to be more aggressive during the startup phase, by incorporating a simple capacity estimation into the startup behavior. When possible, BBA-2 ramps up quickly and fills the buffer with a much higher rate than what the map suggests.

This two phases of operation can be found in many network protocols, such as the slow-start and congestion avoidance phases in TCP. For TCP, when a connection starts, the congestion control algorithm knows nothing about network conditions from the sending window, and the window is quickly opened to use available capacity until packet losses are induced. Similar to TCP, ABR algorithms get little or no information from the playback buffer at the beginning of a session. However, while ABR algorithms also ramp up the video rate quickly, unlike TCP, they need to do it in a controlled manner to prevent unnecessary rebuffers.

From Figure 11, we know that the change of the buffer, $\Delta B = V - (ChunkSize/c[k])$, captures the difference between the instantaneous video rate and system capacity. Now, assuming the current video rate is R_i, to safely step up a rate, $c[k]$ needs to be at least R_{i+1} to avoid rebuffers. In other words, we require $\Delta B \geq V - (ChunkSize/R_{i+1})$. Further, since videos are encoded in VBR, the instantaneous video rate can be much higher than the nominal rate. Let the max-to-average ratio in a VBR stream be e, so that eR_{i+1} represents the maximum instantaneous video rate in R_{i+1}. When the player first starts up, since there is no buffer to absorb the variation, $c[k]$ needs to be at least larger than eR_{i+1} in order to safely step up a rate. In other words, when considering VBR and the buffer is empty, ΔB needs to be larger than $V - (ChunkSize/(eR_{i+1}))$ for the algorithm to safely step up from R_i to R_{i+1}. According to Figure 10, the max-to-average ratio e is around 2 in our system. Since $e = 2$, $R_i/R_{i+1} \sim 2$, and a chunk size can be smaller than half the average chunk size ($ChunkSize \leq 0.5VR_i$), ΔB needs to be larger than $0.875V$s to safely step up a rate when the buffer is empty in our system.

Based on the preceding observation, BBA-2 works as follows. At time $t = 0$, since the buffer is empty, BBA-2 only picks the next highest video rate, if the ΔB increases by more than $0.875V$s. Since $\Delta B = V - ChunkSize/c[k]$, $\Delta B > 0.875V$ also means that the chunk is downloaded *eight times* faster than it is played. As the buffer grows, we use the accumulated buffer to absorb the chunk size variation and we let BBA-2 increase the video rate faster. Whereas at the start, BBA-2 only increases the video rate if the chunk downloads *eight times* faster than it is played, by the time

[6]Note that the startup phase does not refer to the join delay. The startup phase refers to the first few minutes *after* the video has started.

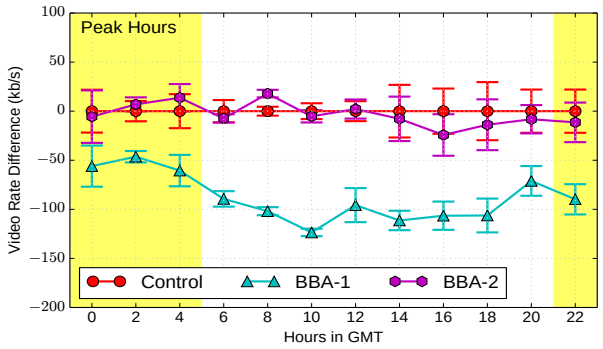


Figure 17: BBA-2 achieved a similar video rates to the *Control* algorithm overall.

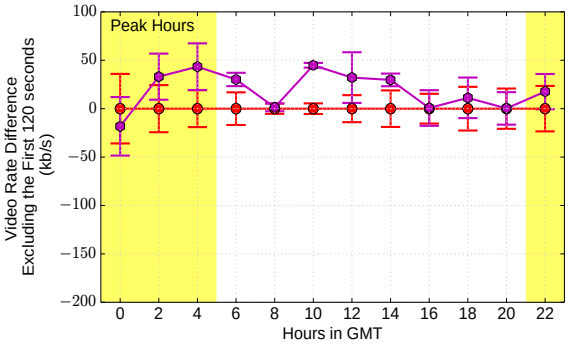


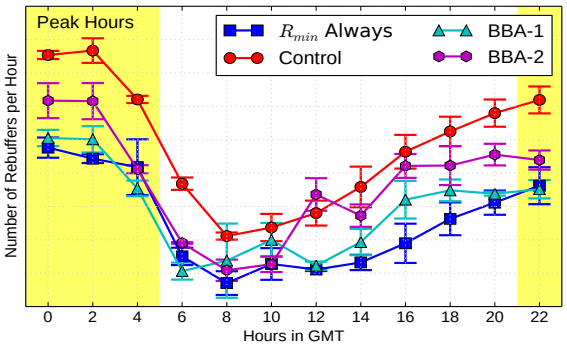
Figure 18: BBA-2 achieved better video rate at the steady state. The steady state is approximated as the period after the first two minutes in each session.

it fills the cushion, BBA-2 is prepared to step up the video rate if the chunk downloads *twice* as fast as it is played. The threshold decreases linearly, from the first chunk until the cushion is full. The blue line in Figure 16 shows BBA-2 ramping up faster. BBA-2 continues to use this startup algorithm until (1) the buffer is decreasing, or (2) the chunk map suggests a higher rate. Afterwards, we use the $f(B)$ defined in the BBA-1 algorithm to pick a rate.

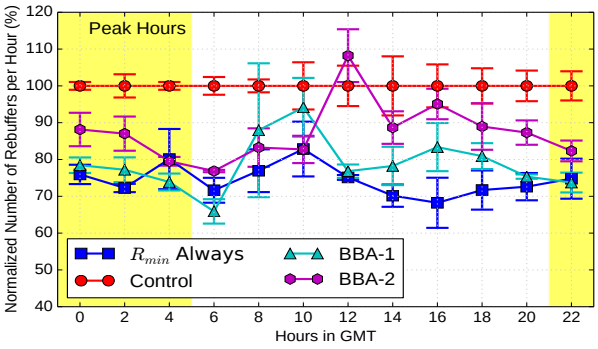
Note that BBA-2 is using ΔB during startup, which encodes a simple capacity estimate: the throughput of the last chunk. This design helps make the algorithm more aggressive at a point when the buffer has not yet accumulated enough information to accurately determine the video rate to use. Nevertheless, note that our use of capacity estimation is restrained. We only look at the throughput of the last chunk, and crucially, once the buffer is built up and the chunk map starts to suggest a higher rate, BBA-2 becomes buffer-based—it picks a rate from the chunk map, instead of using ΔB. In this way, BBA-2 enables us to enjoy the improved steady-state performance of the buffer-based approach, without sacrificing overall bitrate due to a slow startup ramp.

6.1 Results

We ran our experiments during the same time period and with the same pool of users as the previously described experiments, which all occurred between September 6th (Friday) and 9th (Monday), 2013. Figure 17 shows the differ-



(a) Number of rebuffers per playhour throughout the day.



(b) Normalized number of rebuffers per playhour, the number is normalized to the average rebuffer rate of the *Control* in each two hour period.

Figure 19: BBA-2 has a slightly higher rebuffer rate compared to BBA-1, but still achieved 10–20% improvement compared to the *Control* algorithm during peak hours.

ence in the average video rate between *Control*, BBA-1, and BBA-2. From the figures, we see that BBA-2 does indeed increase the video rate. With a faster startup-phase ramp, the video rate with BBA-2 is almost indistinguishable from the *Control* algorithm. This supports our hypothesis that the lower video rates seen by BBA-0 and BBA-1 were due to their conservative rate selection during startup. Furthermore, if we exclude the first two minutes as an approximation of the steady state, the average video rate of BBA-2 is mostly higher than *Control*, as shown in Figure 18. This observation verifies our discussion in Section 3: The buffer-based approach is able to better utilize network capacity and achieve higher average video rate in the steady state.

Figure 19 shows absolute and normalized rebuffers. BBA-2 slightly increases the rebuffer rate. BBA-2 operates in the *risky* zone of Figure 13 and therefore will inevitably rebuffer more often than BBA-1, which only operates in the *safe* area. Nevertheless, the improvements are significant relative to *Control*: BBA-2 maintains a *10–20% improvement in rebuffer rate* compared to the *Control* algorithm during peak hours.

So far, we have successfully relaxed the four idealized assumptions made in Section 3. In BBA-0, we handle the finite chunk size and discrete available video rates through a piecewise mapping function. In BBA-1, we handle the VBR encoding through a variable reservoir size and a chunk map. In BBA-2, we further handle the finite video length by dividing each session into two phases. BBA-2 still follows the buffer-based approach in the steady state, and it uses a simple capacity estimation to ramp up the video rate during the startup. The results demonstrate that by focusing on the buffer, we can reduce the rebuffer rate without compromising the video rate. In fact, the buffer-based approach improves the video rate in the steady state.

In the following section, we will further discuss how to extend the buffer-based approach to tackle other practical concerns.

7. OTHER PRACTICAL CONCERNS

In this section, we extend the buffer-based approach and develop techniques to address two other practical concerns: temporary network outage and frequent video switches.

7.1 Handling Temporary Network Outage

We have shown that buffer-based algorithms never need to rebuffer if the network capacity is always higher than $R_{\min}$. In this section we explore what happens if the network capacity falls *below* $R_{\min}$, for example during a complete network outage. Temporary network outages of 20–30s are not uncommon; e.g., when a DSL modem retrains or a WiFi network suffers interference. To make buffer-based algorithms resilient to brief network outages, we can reserve part of the buffer by shifting the chunk map curve further to the right. The buffer will now converge to a higher occupancy than before, providing some protection against temporary network outage. We call this extra portion of buffer the *outage protection*.

How should we allocate buffers to outage protection? One way is to gradually increase the size of outage protection after each chunk is downloaded. In the implementation of BBA-1, we accumulate outage protection by 400ms for each chunk downloaded when the buffer is increasing and still less than 75% full. In the implementation of BBA-2, we only accumulate outage protection after the algorithm exits the startup phase and is using the chunk map algorithm. A typical amount of outage protection is 20–40 seconds at steady state and is bounded at 80 seconds. The downside of this approach is that the chunk map keeps moving, and can cause video rates to oscillate.

In the following, we describe an alternative way to protect against temporary network outage, while reducing changes to the chunk map, by combining it with the dynamic reservoir calculation.

7.2 Smoothing Video Switch Rate

In Section 5, we showed that we can improve the video rate by using a chunk map and dynamic reservoir calculation. However, this choice makes the video rate change frequently, as shown in Figure 20. Note that it is debatable as to whether video switching rate really matters to the viewer's quality of experience. For example, if a service offers closely spaced video rates, the viewer might not notice a switch. Nevertheless, in the following we will explore mechanisms to reduce the switching rate and introduce a modified algorithm, *BBA-Others*, to address this issue. We will see that by smoothing the changes, we can at least match the switching rate of the *Control* algorithm.

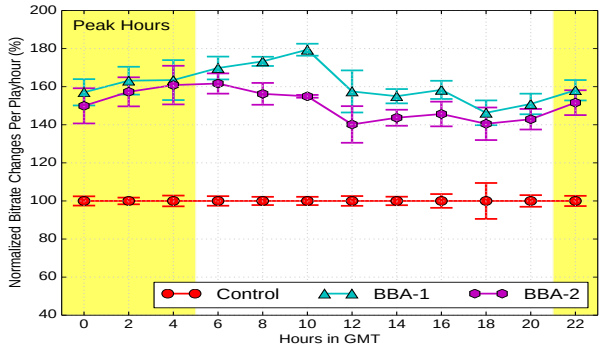


Figure 20: After switching from using a rate map to using a chunk map, the video switching rate of BBA-1 and BBA-2 is much higher than the *Control* algorithm.

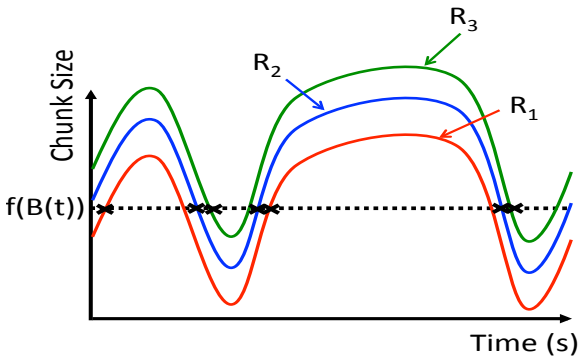


Figure 21: A reason using chunk map increases video switching rate. When using a chunk map, even if the buffer level and the mapping function remains constant, the variation of chunk sizes in VBR streams can make a buffer-based algorithm switch between rates. The lines in the figure represent the chunk size over time from three video rates, R_1, R_2, and R_3. The crosses represent the points where the mapping function will suggest a rate change.

There are two main reasons our buffer-based algorithms increase the frequency of video-rate switches. First, when we use the chunk map, there is no longer a fixed mapping function between buffer levels and video rates. Instead, buffer levels are mapped to chunk sizes, and the nominal rate might change every time we request a new chunk. Even if the buffer level remains constant, the chunk map will cause BBA-1 to frequently switch rates, since the chunk size in VBR encoding varies over time, as illustrated in Figure 21. We can reduce the chance of switching to a new rate—and then switching quickly back again—by looking ahead to future chunks. When encountering a small chunk followed by some big chunks, even if the chunk map tells us to step up a rate, our new algorithm *BBA-Others* will not do so to avoid a likely step down in the near future. The further this modified algorithm looks ahead, the more it can smooth out rate changes. If, in the extreme, we look ahead to the end of the movie, it is the same as using a rate map instead of a chunk map. In the implementation of BBA-Others, we look ahead the same number of chunks as what we have in the buffer.

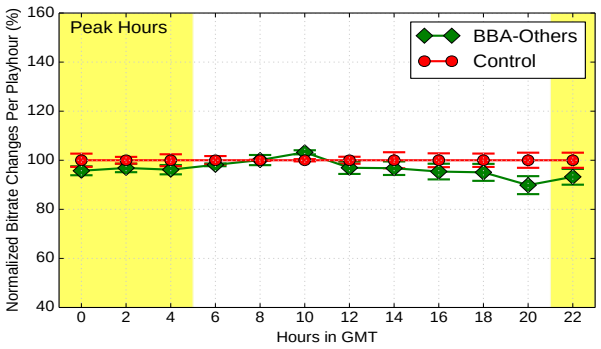


Figure 22: BBA-Others smoothes the frequency of changes to the video rate, making it similar to the *Control* algorithm.

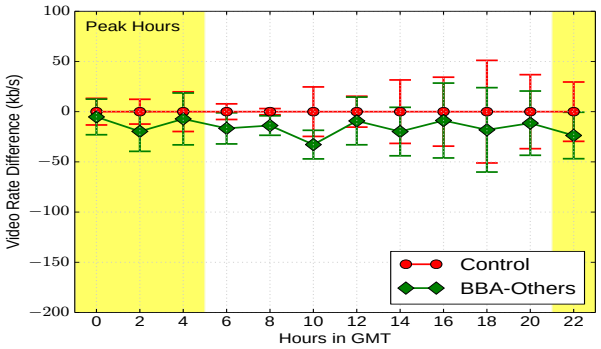


Figure 23: BBA-Others achieves a similar video rate during the peak hours but reduces the video rate by 20–30kb/s during the off-peak.

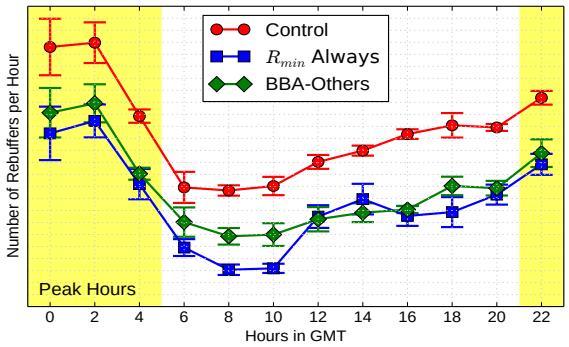
When the buffer is empty, we pick a rate by only looking at the next chunk; when the buffer is full, we look ahead for the next 60 chunks.[7] Note that BBA-Others only smooths out *increases* in video rate. It does not smooth decreases so as to avoid increasing the likelihood of rebuffering.

To explain the second reason, we look at Figure 12. The size of the reservoir is calculated from the chunk size variation in the next 480 seconds. As a result, the reservoir will shrink and expand depending on the size of upcoming chunks. If large chunks are coming up, the chunk map will be right-shifted, and if small chunks are coming up, the chunk map will be left-shifted. Even if the buffer level remains constant, a shifted chunk map might cause the algorithm to pick a new video rate. On top of this, as described in Section 7.1, a gradual increase in outage protection will *also* gradually right-shift the chunk map. Hence, we reduce the number of changes by only allowing the chunk map to shift to the right, never to the left, i.e., the reservoir expands but never shrinks. Since the reservoir cannot be shrinked, the reservoir grows faster than it needs to, letting us use the excess for outage protection.

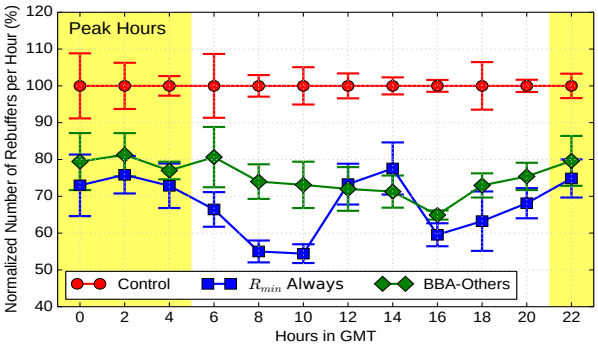
7.3 Results

As before, we randomly pick three groups of real users for our experiment. One third are in the *Control* group, one

[7]Our buffer size is 240 seconds and each chunk is 4 seconds.



(a) Number of rebuffers per playhour throughout the day.



(b) Normalized number of rebuffers per playhour, the number is normalized to the average rebuffer rate of the *Control* in each two hour period.

Figure 24: BBA-Others reduces rebuffer rate by 20–30% compared to the *Control* algorithm.

third always stream at $R_{\min}$, giving us an approximation of the lower bound on rebuffer rate, and one third run the BBA-Others algorithm, which smooths the switching rate by looking ahead and by only allowing the chunk map to be right-shifted. The experiment was conducted between September 20th (Friday) and 22nd (Sunday), 2013.

Figure 22 shows that the video rate changes much less often with BBA-Others than with BBA-1 or BBA-2 (Figure 20). In fact, BBA-Others is almost indistinguishable from *Control*—sometimes higher, sometimes lower.[8] Figure 23 shows the video rate for BBA-Others. Since we does not allow the chunk map to be left-shifted, BBA-Others switches up more conservatively than BBA-2. Although the video rate is almost the same as *Control*, we trade about 20kb/s of video rate compared to BBA-2 in Figure 17.[9] As other buffer-based algorithms, BBA-Others improves the rebuffer rate, since we do not change the frequency of switches to a lower rate. As shown in Figure 24, BBA-Others improves the rebuffer rate by 20–30% compares to the *Control* algorithm.

[8]The numbers are normalized to the average switching rate in *Control* for each two-hour window.

[9]This is only an approximation, since the experiments in Figure 23 and 17 ran in two different weekends in September, 2013.

8. RELATED WORK

Understanding the Impact of Inaccurate Estimates. Prior works have shown that sudden changes in available network capacity confuse existing ABR algorithms, causing the algorithms to either overestimate or underestimate the available network capacity [1, 2, 6, 10, 12].

The overestimation leads to unnecessary rebuffers [2, 6]. In this paper, we quantify how often unnecessary rebuffers happen in a production system and show that 20–30% of rebuffers are unnecessary. Based on this observation, we then propose the buffer-based approach to reduce unnecessary rebuffers.

The underestimation not only fills the buffer with video chunks of lower quality, but also leads to the ON-OFF traffic pattern in video traffic: when the playback buffer is full, the client pauses the download until there is space. In the presence of competing TCP flows, the ON-OFF pattern can trigger a bad interaction between TCP and the ABR algorithm, causing a further underestimate of capacity and a downward spiral in video quality [9]. When competing with other video players, overlapping ON-OFF periods can confuse capacity estimation, leading to oscillating quality and unfair link share among players [1, 10, 12].

In our work, since we request only R_{max} when the buffer approaches full, the ON-OFF traffic pattern appears only when the available capacity is higher than R_{max}. When competing with a long-lived TCP flow, our algorithm continues to request R_{max} when the ON-OFF pattern occurs, avoiding the downward spiral. When competing with other video players, if the buffer is full, all players have reached R_{max}, and so the algorithm is fair.

Buffer-aware ABR Algorithms. Others have proposed using buffer level to adjust capacity estimation. Tian et al. [20] uses a buffer and a PID controller to compute the adjustment function applied to capacity estimates, balancing responsiveness and smoothness. Elastic [5] first measures the network capacity through a harmonic filter, then drives the buffer to a set-point through a controller. These prior works reveal that buffer occupancy provides important information for selecting a video rate. In this paper, we observe that buffer occupancy is in fact the primary state variable that an ABR algorithm should control. This motivates a design that directly chooses the video rate according to the current buffer occupancy, and uses simple capacity estimation only when the buffer itself is growing from empty.

Quality Metrics and User Engagement. User engagement and quality of experience (QoE) are known to depend on rebuffering rate and video rate [7, 11, 14], as well as the delay before playing and how often the video rate changes [7, 19]. Modeling user engagement is complex and on-going [4, 14]. In this work, we focus on the tradeoff between rebuffer events and video bitrate (with some consideration for switching rate). The buffer-based approach can serve as a foundation when considering other metrics.

Improving QoE through other system designs. Client-side ABR algorithms try to make the best decision based on local observations. Their distributed nature yields system scalability, and arguably each client has the best position to observe local events. However, the decisions of these algorithms are reactive and optimize only the performance of a single client. Thus, a centralized control plane is proposed to optimize the global performance through aggregating measurements [13]. The potential benefits from CDN augmenta-

tion mechanisms, such as CDN federation and peer-assisted CDN-P2P hybrid model, are also investigated [3]. Our work is complementary to these efforts and will benefit from them.

9. CONCLUSION

Existing ABR algorithms face a significant challenge in environments where the capacity is rapidly varying (as is observed in practice). In response, ABR algorithms often adjust the capacity estimate based on the buffer occupancy, becoming more conservative (resp., aggressive) as the buffer falls (resp., grows). Motivated by the observation that accurate estimation is challenging when capacity is highly variable, we take this design to an extreme: we directly choose the video rate based on the current buffer occupancy and only use estimation when necessary. Our own investigation reveals that capacity estimation is unnecessary in steady state; however using (simple) capacity estimation (based on immediate past throughput) is important during the startup phase, when the buffer occupancy is growing from empty. We test the viability of this approach through a deployment in Netflix, and the results show that our algorithm can achieve a significant performance improvement.

More generally, our work suggests an alternative roadmap for the development of ABR algorithms: rather than presuming that capacity estimation is required, it is perhaps better to begin by using *only* the buffer, and then ask *when* capacity estimation is needed. Similar to the observations we make in this paper, we might expect that in any setting where the startup phase is a significant fraction of the overall video playback, estimation may be valuable (e.g., for short videos). However, in all such cases, the burden of proof is on the algorithm designer to ensure the additional complexity is necessary.

Acknowledgment

We are grateful to our shepherd Krishna Gummadi and the anonymous reviewers for their valuable comments and feedback, which greatly helped improve the final version. The authors would also like to thank Yiannis Yiakoumis, Greg Wallace-Freedman, Kevin Morris, Wei Wei, Siqi Chen, Daniel Ellis, Alex Gutarin and many other colleagues in both Stanford and Netflix for helpful discussions that shaped the paper. This work was supported by Google U.S./Canada PhD Student Fellowship and the National Science Foundation under grants CNS-0832820, CNS-0904609, and CNS-1040593.

10. REFERENCES

[1] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. Begen. What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth? In *ACM NOSSDAV*, June 2012.

[2] S. Akhshabi, C. Dovrolis, and A. Begen. An Experimental Evaluation of Rate Adaptation Algorithms in Adaptive Streaming over HTTP. In *ACM MMSys*, 2011.

[3] A. Balachandran, V. Sekar, A. Akella, and S. Seshan. Analyzing the Potential Benefits of CDN Augmentation Strategies for Internet Video Workloads. In *ACM IMC*, October 2013.

[4] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a Predictive Model of Quality of Experience for Internet Video. In *ACM SIGCOMM*, August 2013.

[5] L. D. Cicco, V. Caldaralo, V. Palmisano, and S. Mascolo. ELASTIC: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *IEEE Packet Video Workshop*, December 2013.

[6] L. D. Cicco and S. Mascolo. An Experimental Investigation of the Akamai Adaptive Video Streaming. In *USAB*, November 2010.

[7] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *ACM SIGCOMM*, August 2011.

[8] T.-Y. Huang. *A Buffer-Based Approach to Video Rate Adaptation*. PhD thesis, CS Department, Stanford University, June 2014.

[9] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *ACM IMC*, November 2012.

[10] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *ACM CoNEXT*, 2012.

[11] S. S. Krishnan and R. K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. In *ACM IMC*, November 2012.

[12] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran. Probe and adapt: Rate adaptation for http video streaming at scale. In *http://arxiv.org/pdf/1305.0510*.

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

[14] Y. Liu, S. Dey, D. Gillies, F. Ulupinar, and M. Luby. User Experience Modeling for DASH Video. In *IEEE Packet Video Workshop*, December 2013.

[15] R. Mok, X. Luo, E. Chan, and R. Chang. QDASH: a QoE-aware DASH system. In *ACM MMSys*, 2012.

[16] Sandvine: Global Internet Phenomena Report 2012 Q2. `http://tinyurl.com/nyqyarq`.

[17] Sandvine: Global Internet Phenomena Report 2013 H2. `http://tinyurl.com/nt5k5qw`.

[18] Netflix ISP Speed Index. `http://ispspeedindex.netflix.com/`.

[19] H. Sundaram, W.-C. Feng, and N. Sebe. Flicker Effects in Adaptive Video Streaming to Handheld Devices. In *ACM MM*, November 2011.

[20] G. Tian and Y. Liu. Towards Agile and Smooth Video Adaptation in Dynamic HTTP Streaming. In *ACM CoNEXT*, December 2012.

[21] Private conversation with YouTube ABR team.