Downton Abbey Without the Hiccups: Buffer-Based Rate Adaptation for HTTP Video Streaming

Te-Yuan Huang Ramesh Johari Nick McKeown
Stanford University
{huangty,ramesh.johari, nickm}@stanford.edu

ABSTRACT

Recent work has shown how hard it is to pick a video streaming rate. Video service providers use heuristics to estimate the network capacity leading to unnecessary rebuffering events and suboptimal video quality. This paper argues that we should do away with estimating network capacity, and instead directly observe and control the playback buffer. We present a class of rate selection algorithms that allow us to optimize the delivered video quality while provably never unnecessarily rebuffering. Our algorithms work with discrete video rates, video chunking and for both CBR and VBR video codecs.

Categories and Subject Descriptors

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

General Terms

Algorithms, Design

Keywords

HTTP-based Video Streaming, Video Rate Adaptation Algorithm

1. INTRODUCTION

Today, video streaming is a huge and growing fraction of Internet traffic, with Netflix and YouTube accounting for over 50% of the peak download traffic in the US [7]. In order to provide a high-quality user experience, most commercial video streaming services use dynamic rate selection algorithms to adapt video quality based on the available capacity their clients perceive. The majority of these streaming services run over HTTP.

How do HTTP streaming services pick a video rate? First, the video client estimates how fast the server can deliver video (i.e. the available capacity) by measuring the time-weighted average arrival rate of data at the HTTP layer. Next, the client picks a video rate to match the estimated capacity. If it chooses a video rate that is *too high*, the viewer will experience *rebuffering events* (i.e., playback will halt because the playback buffer goes empty); if it picks a video rate that is *too low*, the viewer will experience suboptimal video quality.

Since capacity is estimated using a weighted average of recent throughput, the estimate is typically not the same as the true current available capacity. This mismatch leads to a known problem in today's video streaming algorithms: they can be both *too conservative* and *too aggressive*. Figure 1(a) shows an example in which the video is streamed too slowly and the stream does not obtain its fair share [5] (too conservative). On the other hand, Figure 1(b) shows an example in which the video keeps playing at an inappropriately high rate after the available capacity has dropped (too aggressive).[1] As a result, the client rebuffers unnecessarily and is not able to resume playing until 200 seconds later. Note that the rebuffering events were entirely avoidable, since the available capacity is still high enough to support the lowest video rate.

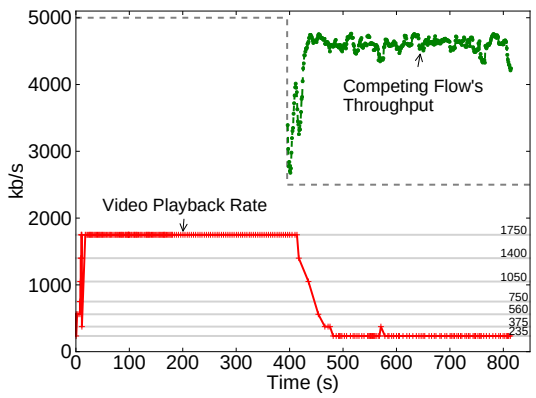
At first glance it seems video rate selection algorithms are forced into a tradeoff. Requesting a higher video rate might lead to being overly aggressive and unnecessary rebuffering. On the other hand, requesting a lower video rate might under utilize the available capacity and lead to unnecessarily low video quality.

In this paper we show that this is a *false choice*: *neither* of the situations should ever happen! We present a class of video rate selection algorithms that: (1) never unnecessarily rebuffer; and (2) are free to pick the highest possible video rate. Our algorithms achieve both objectives simultaneously by *choosing a video rate based only on the current buffer occupancy, and avoid estimating bandwidth at all.* To the best of our knowledge, we are the first to remove bandwidth estimation from video rate selection algorithms and base the algorithms exclusively on playout buffer occupancy.

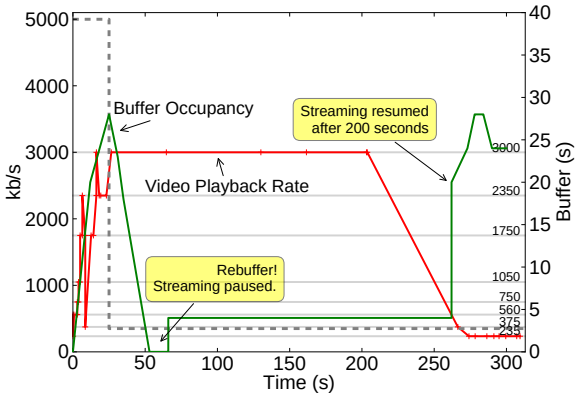
Why not estimate bandwidth? Capacity estimation depends on many factors, such as the size of each video chunk, the dynamics of TCP's congestion control algorithm, competing flows and the load on the server. Today's video streaming services introduce many heuristics to control the error without understanding the implications. For example, some heuristics can adversely interact with the underlying

FhMN'13, August 16, 2013, Hong Kong, China.

[1]Both figures are based on real measurements from Service A in [5], a very large video service provider. We do not identify the service provider as we have found the problem to be common across many commercial video service offerings.



(a) *Being too conservative*: A video starts streaming at 1.75Mb/s over a 5Mb/s network. After 395 seconds, a second flow starts (from the same server). The video should be able to stream at 1.75Mb/s (given its fair share of 2.5Mb/s), but instead drops down to 235kb/s.



(b) *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.

Figure 1: A paradox in video streaming algorithms that use capacity estimation (Service A in [5]).

TCP logic as shown in [1, 2, 5]. As we see above, picking a video rate based on capacity estimation can lead to unnecessary rebuffering and sub-optimal video quality.

If we are not estimating capacity, how should our algorithm pick a video rate? Let's take a step back and look at the purpose of the video rate selection algorithm. Ultimately, the algorithm is trying to *control the playback buffer*. It is trying to prevent unnecessary buffer underruns (i.e. rebuffering events) and overruns (i.e. streaming below the highest possible quality level [5]).

If it is the playback buffer we are controlling, then why not measure and control its occupancy directly? The current buffer occupancy tells us how far we are from an underrun or overrun, and its rate of change captures the mismatch between the system capacity and the requested video rate. The buffer occupancy reflects the end-to-end system capacity, including current load conditions of the network, the CDN, and the video client. In brief, the buffer occupancy contains a lot of information and is the variable we are directly trying to control.

Inspired by these observations, we propose a class of algorithms that select the video rate based *solely* on the playback buffer level. Our algorithms do no capacity estimation at all; rather, they rely on the buffer dynamics to implicitly capture the available capacity. In this paper we show that our algorithms (1) will never unnecessarily rebuffer, and (2) achieve an average video rate equal to available capacity in steady state.

The basic intuition behind our results – obvious with hindsight – is that if the network capacity never falls below the minimum video rate, the algorithm should *never* rebuffer. This observation leads to a simple but interesting consequence: we can separate the task of maximizing video quality from the task of avoiding rebuffering. First we dispense with unnecessary rebuffering, allowing us to control the video rate independently.

Of course, in practice, a streaming algorithm must take care of many other considerations: it must converge quickly to the right video rate, but it must not change the video rate too often (since users find it distracting) [3]; it must work with discrete video rates, and both constant and variable bit-rate coding. We will show how our algorithms allow all of these factors to be taken into consideration.

The remainder of the paper is organized as follows. Section 2 formally defines a model for HTTP video streaming and our problem statement. Section 3 defines the class of algorithms we consider. Section 4 uses an idealized setting to show that the algorithms we propose can prevent unnecessary rebuffering while maximizing video quality. We then relax our assumptions and discuss variable bit rate encoding in Section 5. In Section 6, we discuss other performance criteria, including convergence time and rate oscillation. We then show some preliminary results in Section 7. Finally, we conclude this work in Section 8.

2. AN HTTP STREAMING MODEL

To set the stage, we start with a formal model of HTTP-based video streaming.

Video rates and video chunks. For a typical commercial HTTP-based video streaming service, videos are pre-encoded into a discrete set of video rates (e.g. 235kb/s, 500kb/s, 1.5Mb/s and so on) and stored as separate files by the CDNs. When a customer streams a video, the video client downloads the video piece by piece by issuing HTTP byte-range requests. The individual pieces are called *chunks*. A client can only change its selected video rate on a chunk-by-chunk basis (since this is the granularity of requests). Regardless of the encoded video rate, each video chunk contains the same length of video in seconds of playback.

In our formal model, we assume the video client can choose from a set of m discrete video rates, $\{R_1, \ldots, R_m\}$, where $R_1 < R_2 < \ldots < R_m$. We also refer to R_1 as $R_{\min}$ (the minimum video rate), and R_m as $R_{\max}$ (the maximum video rate).

The streaming buffer. In the model we consider, the streaming buffer in the video client is typically measured in

seconds of playback time.[2] At any time the buffer may contain chunks with many different video rates, and the output bitrate of the buffer will depend on the video rate of the chunk currently being played. As a result, there is no direct mapping between buffer occupancy in bytes and buffer occupancy in seconds. By measuring the buffer in the time domain, the client keeps a record of how many "video seconds" worth of playback video currently resides in the buffer, without having to track the video rate associated with each video chunk. Since the viewer ultimately cares about playback, streaming clients commonly measure the buffer in seconds rather than bytes. Previous literature on HTTP video streaming also uses the time-based buffer model [5, 8]. We let $B_{\max}$ denote the buffer capacity in seconds.

Buffer dynamics. We index time by $t \geq 0$, and let $C(t)$ denote the system capacity at time t. Here, the system capacity represents the overall end-to-end capability of the system, including the capacity of the CDN servers, the video client and the available bandwidth of the network in between. We let $B(t)$ be the streaming buffer occupancy at time t (measured in seconds). Finally, we let $R(t)$ denote the video rate selected at time t. (Note that if the buffer is full, then no chunk can be downloaded at time t; thus we adopt the convention that if $B(t) = B_{\max}$ then $C(t) = 0$.)

Observe that the buffer drains at unit rate (since one second is played back every second of real time), and fills at rate $C(t)/R(t)$. Thus the buffer dynamics obey the following simple differential equation:

$$\frac{dB(t)}{dt} = \begin{cases} [C(t)/R(t) - 1]^+, & \text{if } B(t) = 0; \\ C(t)/R(t) - 1, & \text{if } 0 < B(t) < B_{\max}; \\ [C(t)/R(t) - 1]^-, & \text{if } B(t) = B_{\max}. \end{cases} \quad (1)$$

(Here the notation $[x]^+$ means the positive part of x, $\max\{x, 0\}$; and $[x]^-$ means the negative part of x, $\min\{x, 0\}$.)

Given the constraint that 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*. Formally, let t_k be the completion time of the k-th chunk; by convention, let $t_0 = 0$. Let $r[k]$ be the video rate selected for the k-th chunk, so that $R(t) = r[k]$ for $t_{k-1} < t \leq t_k$. Similarly, let $c[k]$ be the average download capacity for the k-th chunk, so that:

$$c[k] = \frac{\int_{t_{k-1}}^{t_k} C(t)\, dt}{t_k - t_{k-1}}.$$

If each chunk contains V seconds of video, the k-th chunk is $Vr[k]$ bytes long. This assumes a constant bit rate (CBR) stream; we extend our results to variable bit rate video (VBR) in Section 5. Thus we have $t_k = t_{k-1} + Vr[k]/c[k]$ (since this is when the k-th chunk completes downloading). On the other hand, between t_{k-1} and t_k, the buffer fills with V seconds of video. Therefore:

$$B(t_k) = \left[B(t_{k-1}) + V - \frac{Vr[k]}{c[k]} \right]^+. \quad (2)$$

(Note that since $C(t) = 0$ whenever $B(t) = B_{\max}$, the buffer must be less than or equal to $B_{\max}$ when a chunk completes.)

We summarize the two equivalent models of buffer dynamics in Figure 2.

Problem statement. We can now define the two objectives discussed in the introduction.

[2]This is typical of streaming clients, e.g., Silverlight.

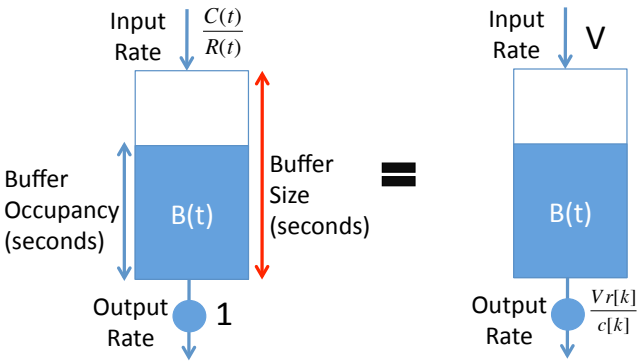


Figure 2: Two models of the streaming buffer.

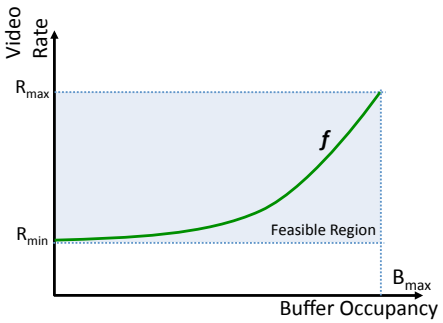


Figure 3: The design space of rate maps.

No unnecessary rebuffering. An *unnecessary* rebuffering event occurs when the buffer underruns, despite the fact that sufficient capacity was available. Formally, we require the following property: *If* $C(t) > R_{\min}$ *for all* $t \geq 0$, *then* $B(t) > 0$ *for all* $t \geq 0$.

Average video rate optimization. The playback quality (as perceived by the viewer) is measured by averaging the video rate over chunks; thus the long-run average video rate is $\bar{R} = \lim_{K\to\infty} \frac{1}{K} \sum_{k=0}^{K} r[k]$. Our goal is to maximize $\bar{R}$. At the same time, this rate must be less than or equal to the long-run average capacity, $\bar{C} \leq \lim_{T\to\infty} \frac{1}{T} \int_0^T C(t)\, dt$. Note that even though our goal is to maximize capacity utilization, the underlying TCP ensures the algorithm stays a "good citizen" and only gets its fair share of available capacity.

3. RATE MAPS AND BUFFER-BASED ALGORITHMS

In this section we define our class of algorithms. Since we select a video rate based solely on the buffer occupancy, the design space for our algorithms can be expressed as the buffer-rate plane shown in Figure 3. The shaded 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*, that maps a current buffer occupancy to a video rate between $R_{\min}$ and $R_{\max}$. We focus on rate maps that are *continuous* functions of the buffer occupancy B, and that are *strictly increasing* on the region $\{B : R_{\min} < f(B) < R_{\max}\}$.

Note that a rate map by itself does not define an algorithm. Since the rate map is continuous, it may not directly correspond to an available discrete video rate. And con-

tinuously changing the rate might cause the video rate to oscillate. Instead, we desire an implementation where the video rate is a little "sticky".

We therefore use the rate adaptation algorithm described in Algorithm 1. The algorithm follows a simple principle: it stays at the current video rate as long as the rate suggested by the rate map does not pass either the next higher available video rate (Rate_+) or the next lower available video rate (Rate_-). If either of these "barriers" are 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.

Algorithm 1: Video Rate Adaptation Algorithm

Input: $\text{Rate}_{\text{prev}}$: The previously used video rate
Buf_{now}: The current buffer occupancy
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 $f(\text{Buf}_{\text{now}}) \geq \text{Rate}_+$ **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 $\text{Rate}_{\text{next}}$;

4. AN IDEALIZED SETTING

Given the algorithm described in Section 3, we now find rate mappings that achieve our two objectives: (1) no unnecessary rebuffers and (2) average video rate maximization. In this section we use an idealized model to gain some intuition, before relaxing our assumptions.

We make the following two simplifying assumptions:

1. *Any* video rate between $R_{\min}$ and $R_{\max}$ is available.
2. The chunk size is infinitesimal, so that we can change the video rate continuously.

With these two assumptions the algorithm from Section 3 becomes quite simple: at every time t, we *instantaneously* map the buffer level $B(t)$ to the video rate $f(B(t))$. The resulting buffer dynamics are:

$$\frac{dB(t)}{dt} = \begin{cases} [C(t)/f(0) - 1]^+, & \text{if } B(t) = 0; \\ C(t)/f(B(t)) - 1, & \text{if } 0 < B(t) < B_{\max}; \\ [C(t)/f(B_{\max}) - 1]^-, & \text{if } B(t) = B_{\max}. \end{cases} \tag{3}$$

In what follows, we consider rate maps f (cf. Section 3) that are pinned at both ends: $f(0) = R_{\min}$ and $f(B_{\max}) = R_{\max}$. In other words, the rate map moves from the lowest to highest video rate as the buffer moves from empty to full. As we will see, any such rate map automatically give us the desired properties (in this idealized setting).

No unnecessary rebuffering. Since $f(B) \to R_{\min}$ as $B \to 0$, the derivative of $B(t)$ will become positive before the buffer hits zero. Thus we have the following result.

Theorem 1 (No unnecessary rebuffers)**.** *As long as $C(t) > R_{\min}$ for all t, $B(t) > 0$ for all $t > 0$ as well.*

Average video rate maximization. Now suppose that $C(t) = C$ for all t, where $R_{\min} < C < R_{\max}$. Informally, we can expect the buffer level to eventually converge to a value B^* where $f(B^*) = C$—in other words, the video rate selected will exactly match the capacity. This is captured in the following theorem (and proved in [6]).

Theorem 2 (Average video rate maximization)**.** *Suppose that $R_{\min} < C < R_{\max}$. Then starting from any initial buffer level, $\lim_{t\to\infty} B(t) = B^*$, where B^* is the unique solution to $f(B^*) = C$.*

Note that if $C > R_{\max}$, it is clear from (3) that the buffer will fill up and remain full; and thus the video rate will remain at $R_{\max}$, which is the best we can hope for. (In practice, we would observe "on-off" behavior in the video stream due to finite chunk sizes.)

The preceding results are obtained in a highly idealized setting. In the next two sections we see how our algorithms behave in a more realistic context, and show that essentially the same insights continue to hold.

5. A MORE REALISTIC SETTING

In this section, we remove the two simplifying assumptions. First, we select the video rate from the finite feasible set $\{R_1, \ldots, R_m\}$; second, we assume the chunk size is finite (V seconds long), and we can only change the video rate when requesting a new chunk.

Note that with a discrete set of video rates, the buffer level $B(t)$ no longer directly maps to a video rate $f(B(t))$, since $f(B(t))$ might not be an available rate. Thus Algorithm 1 defined in Section 3 has slightly more complex dynamics, making the analysis more challenging.

Nevertheless, we now show how to choose rate maps so that our algorithm still achieves our two objectives. We also discuss extensions of our results to a setting with variable bit rates (VBR).

No unnecessary rebuffering. Because video is requested and delivered in V second chunks, the buffer might go empty while a chunk is still downloading, giving us no opportunity to react (since we can only pick a new rate when a chunk has finished). To prevent this possibility, we build a "reservoir" $r > 0$ into the rate map—a lower threshold in the playback buffer, such that $f(B) = R_{\min}$ for all $B \leq r$. Figure 4 shows the rate map and a reservoir on the buffer-rate plane.

To choose r we use the chunk-based buffer dynamic in equation (2). Because the buffer can drop by at most $\frac{V\,r[k]}{c[k]}$ during a chunk download, we can show that if the reservoir is $r \geq V(R_{\max}/R_{\min})$, the client will not unnecessarily rebuffer.[3]

Optimizing average video rate. Ideally, if the capacity is constant ($C = C(t)$ for all t, and $R_{\min} < C < R_{\max}$) we want the average video rate to match the capacity, i.e., $\bar{R} = C$,

[3]Note that this condition is quite conservative if $R_{\max}/R_{\min}$ is large; in general, with knowledge of the video rates we can reduce the reservoir. See [6] for details.

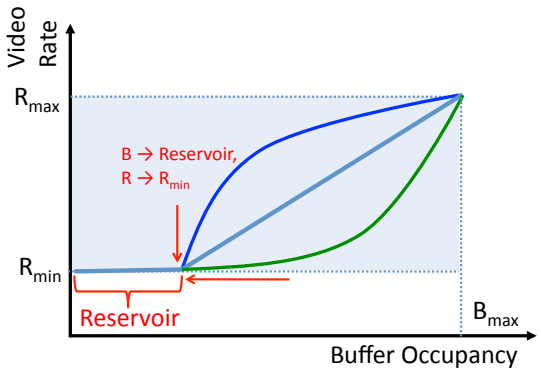


Figure 4: Rate Maps with *Reservoir*.

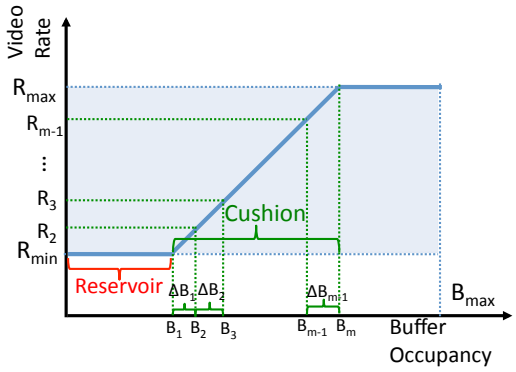


Figure 5: Both convergence time and number of video rate switches depends on the buffer distance between rates.

even with video delivered in chunks, and even when the capacity does not equal one of the discrete video rates. Under Algorithm 1, the buffer level might not be able to converge to B^*, because $f(B^*)$ might not map to an available video rate. The buffer will swing between two occupancies B_i and B_j, where $f(B_i)$ and $f(B_j)$ maps to discrete rates above and below C (R_i and R_j respectively, i.e. $R_i < C < R_j$).

Suppose we again use "pinned" rate maps such that $f(B) = R_{\min}$ for all B such that $0 \leq B \leq r$ (the reservoir), and $f(B_{\max}) = R_{\max}$. For such rate maps, informally, the rate will "hover" around the capacity C in steady state. Formally, we prove that the long-run average video rate, $\bar{R}$, equals the capacity C (the proof is in [6]).

Variable Bit Rate (VBR) Coding. So far we have only considered constant bit-rate (CBR) video, i.e., for a given video rate, each chunk contains the same number of bytes. In practice, video streaming services use variable bit-rate (VBR) encoding depending on how dynamic a scene is. The chunk size varies over time and, typically, the average chunk size is bounded over a fixed time period (e.g. 10 seconds). Because we know the size of each chunk, we can normalize the buffer input with the ratio of a chunk size to the average chunk size. With this scaling, the buffer dynamics are essentially the same as for CBR (proofed in Theorem 3), and hence we can obtain the same results as before.

Note that we only need to have one additional variable to keep track of the normalized buffer occupany. This variable is updated whenever a chunk is finished download and is reset to the actual buffer occupancy whenever the video rate is changed. The reset is necessary to prevent the growing difference between normalized buffer occupancy and the actual one. When staying at the same video rate, the difference between two buffer occupancy will be offseted by the nature of VBR encoding. More details can be found in [6].

Theorem 3. *Given a VBR stream, if we normalize the V seconds contains in each chunk to $V \times \frac{ChunkSize}{AvgChunkSize}$, the change rate of the normalized buffer occupancy on a VBR stream will be the same as the change rate of the actual buffer occupancy on a CBR stream.*

Proof. In the case of CBR, the buffer occupancy will be changed at the rate of $\frac{C(t)}{R(t)} - 1$, as shown in Figure 2.

Suppose the size of the downloaded chunk is $VR_i + \Delta$, where VR_i is the averange chunk size for chunks with video rate R_i and Δ is the deviation of the current chunk size from the average chunk size. Note Δ can be either negative or positive. The scaling normalizes the V seconds contains in a chunk by the ratio of its size and the average chunk size in the selected video rate. Since $R(t) = R_i$, the normalized video length of a chunk, i.e., the input of the normalized buffer after downloading the chunk, would be:

$$V \times \frac{VR_i + \Delta}{VR_i} = \frac{VR_i + \Delta}{R_i} \tag{4}$$

Since the capacity is $C(t)$, it requires $\frac{VR_i+\Delta}{C(t)}$ seconds for the chunk to be downloaded, which means the streaming buffer depleted $\frac{VR_i+\Delta}{C(t)}$ seconds during the download. From Equation 4, we know the input is normalized to $\frac{VR_i+\Delta}{R_i}$. Thus, the change of the normalized buffer occupancy would be the input minus the output: $\frac{VR_i+\Delta}{R_i} - \frac{VR_i+\Delta}{C(t)}$. The change rate of the normalized buffer occupancy would be:

$$\frac{\frac{VR_i+\Delta}{R_i} - \frac{VR_i+\Delta}{C(t)}}{\frac{VR_i+\Delta}{C(t)}} = \frac{C(t)}{R_i} - 1 \tag{5}$$

Equation 5 shows that the change rate is the same as the change rate of the actual buffer occupancy when downloading a CBR chunk. Thus, the normalized buffer occupancy can effectively offset the buffer occupancy variation caused by VBR chunk size variation. As a result, if we adapt the video rate based the normalized buffer occupancy, instead of the actual one, we can maintain all the properties mentioned in the previous sections. □

6. TRANSIENT BEHAVIOR

In practice, a rate selection algorithm needs to control the transient behavior, including the convergence time and how often the video rate switches. We now show how to control transient behavior, and show that these two properties are tradeoffs.

Controlling convergence time. If the capacity is constant ($C(t) = C$), then our target buffer level is B^*, where $f(B^*) = C$. The convergence time is how long it takes us to reach target B^* from our current buffer level. Let Δ_k be the *buffer distance* between rates R_k and R_{k+1} according to the rate map; i.e., $\Delta_k = f^{-1}(R_{k+1}) - f^{-1}(R_k)$. It follows that the smaller the buffer distances between video rates (i.e, the "steeper" the rate map), the shorter the convergence time. By picking the buffer distance between video rates in the

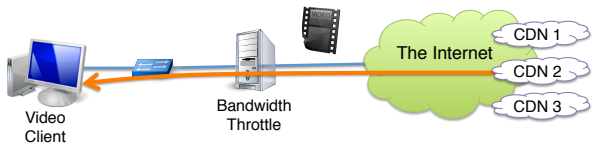


Figure 6: The Experiment Setup.

rate map, we can control the convergence time. (See [6] for details.)

Limiting how often we switch video rates. Although smaller buffer distances between rates mean a faster convergence time, it also means lower resistance to capacity variation and therefore increases how often we switch video rates. For example, if the capacity fluctuates between $R_i + \Delta R$ and $R_i - \Delta R$ every T seconds, the video rate will only stay at the correct average value, R_i, if the buffer level does not cross the "barriers" between adjacent rates.

This makes clear the tradeoff between faster convergence time (i.e. small buffer distances) and less frequent video rate changes (i.e. larger buffer distances). By expressing the tradeoff, the algorithm designer can explore the design space and optimize the rate map for the operating environment.

7. PRELIMINARY RESULTS

To compare the performance of the proposed buffer-based algorithm with the current algorithm used in Service A, we have our prototype video client, with our buffer-based algorithm, streaming real video from Service A, using the same methodology described in [5]. The router buffer size is 120kbit, enough to sustain 100% throughput for connections with a 4-20ms RTT. Our client uses the rate map shown in Figure 5, with a reservoir of 45 seconds and a cushion of 15 seconds.

We use the same bandwidth setting as in Figure 1(b): the bandwidth starts out at 5Mb/s and then is throttled to 350kb/s after 25 seconds. The result is shown in Figure 7. The solid lines show the buffer occupancy and playback rate if we use our basic algorithm, and assume chunks are CBR encoded. The dashed lines account for the fact that the video is really VBR, and so our algorithm operates on the normalized buffer occupancy as discussed in Section 5. In both cases, the video rate converges to the highest video rate quickly, since the buffer distance between the reservoir and the target rate is relatively small. If we account for VBR, the buffer converges even faster because this video happens to start with larger-than-average chunk sizes.

When we throttle the bandwidth to 350kb/s, for the solid line, the video rate steps down to 1750kb/s as the buffer level is depleted, then all the way down to 235kb/s when the buffer level dips into the reservoir. For the dashed line, it takes more than 40 seconds to finish downloading the last 3000kb/s chunk. As a result, it drops directly to the lowest rate, since the buffer has already dipped into the reservoir. Neither case experiences rebuffering events. In contrast, the current algorithm used in Service A would keep requesting for an inappropriately high video rate after the bandwidth has dropped and resulted in buffer underrun, as shown in Figure 1(b).

These preliminary results demonstrate our prototype buffer-based video selection algorithm with real streaming video. More extensive trials are still needed to fully test the approach.

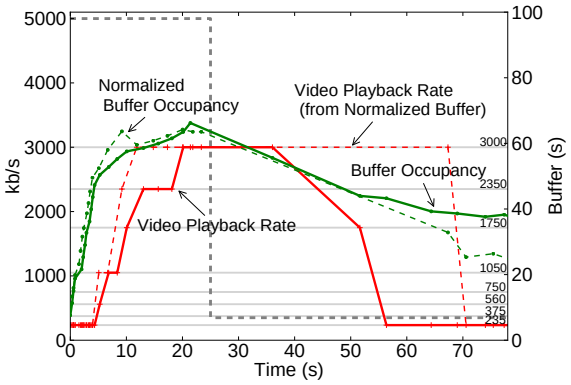


Figure 7: Preliminary Results.

8. CONCLUSION

To the best of our knowledge, we are the first to propose a video rate adaptation algorithm based *solely* on the playback buffer. Most of the existing works depends the bandwidth estimation to pick a video rate [5]. Some prior works used the buffer occupancy as a feedback signal, but the main algorithm was still based on capacity estimation [4, 8]. The immediate benefit of our approach is to allow designers to focus on optimizing the viewer's video quality—maximize the average delivered rate, converge quickly, avoid switching rates too often—without worrying about rebuffering events. In essence, we are benefiting from the well-known "separation of concerns" that naturally arises from layering: By letting TCP find the fair share of the network capacity for us, we can limit our scope to effective management of the playback buffer.

9. REFERENCES

[1] S. Akhshabi et al. An Experimental Evaluation of Rate Adaptation Algorithms in Adaptive Streaming over HTTP. In *ACM MMSys*, 2011.

[2] S. Akhshabi et al. What Happens When HTTP Adaptive Streaming Players Compete for Bandwidth? In *ACM NOSSDAV*, June 2012.

[3] A. Balachandran et al. A Quest for an Internet Video Quality-of-Experience Metric. In *ACM HotNets-XI*, 2012.

[4] L. De Cicco et al. Feedback Control for Adaptive Live Video Streaming. In *ACM MMSys*, 2011.

[5] T.-Y. Huang et al. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *ACM IMC*, November 2012.

[6] T.-Y. Huang et al. Buffer-based rate adaptation for http video streaming. Technical report, 2013. http://goo.gl/mD1uM.

[7] Sandvine: Global Internet Phenomena Report. http://www.sandvine.com/news/pr_detail.asp?ID=312.

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