Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale

Zhi Li, Xiaoqing Zhu, *Member, IEEE*, Joshua Gahm, Rong Pan, Hao Hu, *Student Member, IEEE*, Ali C. Begen, *Senior Member, IEEE*, and David Oran

Abstract—Today, the technology for video streaming over the Internet is converging towards a paradigm named HTTP-based adaptive streaming (HAS), which brings two new features. First, by using HTTP/TCP, it leverages network-friendly TCP to achieve both firewall/NAT traversal and bandwidth sharing. Second, by pre-encoding and storing the video in a number of discrete rate levels, it introduces video bitrate adaptivity in a scalable way so that the video encoding is excluded from the closed-loop adaptation. A conventional wisdom in HAS design is that since the TCP throughput observed by a client would indicate the available network bandwidth, it could be used as a reliable reference for video bitrate selection.

We argue that this is no longer true when HAS becomes a substantial fraction of the total network traffic. We show that when multiple HAS clients compete at a network bottleneck, the discrete nature of the video bitrates results in difficulty for a client to correctly perceive its fair-share bandwidth. Through analysis and test bed experiments, we demonstrate that this fundamental limitation leads to video bitrate oscillation and other undesirable behaviors that negatively impact the video viewing experience. We therefore argue that it is necessary to design at the application layer using a "probe and adapt" principle for video bitrate adaptation (where "probe" refers to trial increment of the data rate, instead of sending auxiliary piggybacking traffic), which is akin, but also orthogonal to the transport-layer TCP congestion control. We present PANDA – a client-side rate adaptation algorithm for HAS – as a practical embodiment of this principle. Our test bed results show that compared to conventional algorithms, PANDA is able to reduce the instability of video bitrate selection by over 75% without increasing the risk of buffer underrun.

Index Terms—CDN, DASH, HAS, HTTP Adaptive Streaming, TCP, Video.

I. INTRODUCTION

OVER the past few years, we have witnessed a major technology convergence for Internet video streaming towards a new paradigm named HTTP-based adaptive streaming (HAS). Since its inception in 2007 by Move Networks [1], HAS has been quickly adopted by major vendors and service providers. Today, HAS is employed for over-the-top video delivery by many major media content providers. A recent report by Cisco [7] predicts that video will constitute more than 90% of the total Internet traffic by 2014. Therefore, HAS may become a predominant form of Internet traffic in just a few years.
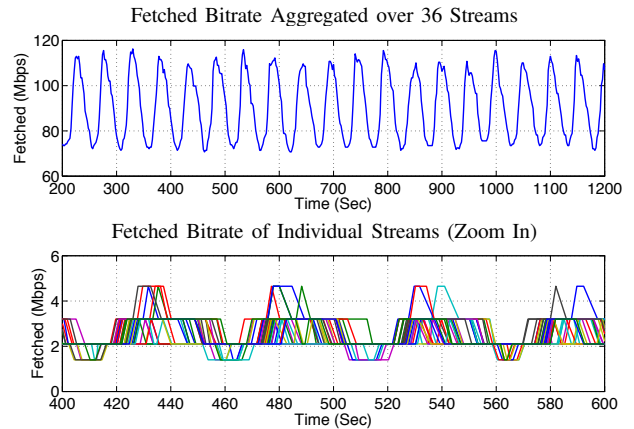
Fig. 1. Oscillation of video bitrate when 36 Microsoft Smooth clients compete at a 100-Mbps link. For more detailed experimental setup, refer to §VII-B.

In contrast to conventional RTP/UDP-based video streaming, HAS uses HTTP/TCP – the protocol stack traditionally used for Web traffic. In HAS, a video stream is chopped into short segments of a few seconds each. Each segment is pre-encoded and stored at a server in a number of versions, each with a distinct video bitrate, resolution and/or quality. After obtaining a manifest file with necessary information, a client downloads the segments sequentially using plain HTTP GETs, estimates the network conditions, and selects the video bitrate of the next segment on-the-fly. A conventional wisdom assumes that since the bandwidth sharing of HAS is dictated by TCP, the problem of video bitrate selection can be resolved straightforwardly. A simple rule of thumb is to approximately match the video bitrate to the observed TCP throughput.

A. Emerging Issues

A major trend in HAS use cases is its large-scale deployment in managed networks by service providers, which typically results in aggregating multiple HAS streams in the aggregation/core network. For example, an important scenario is that within a single household or a neighborhood, several HAS flows belonging to one DOCSIS[1] bonding group compete for bandwidth. Similarly, in the unmanaged wide-area Internet, as HAS is growing to become a substantial fraction of the total traffic, it will become more and more common to have multiple HAS streams compete for available bandwidth at a bottleneck link.

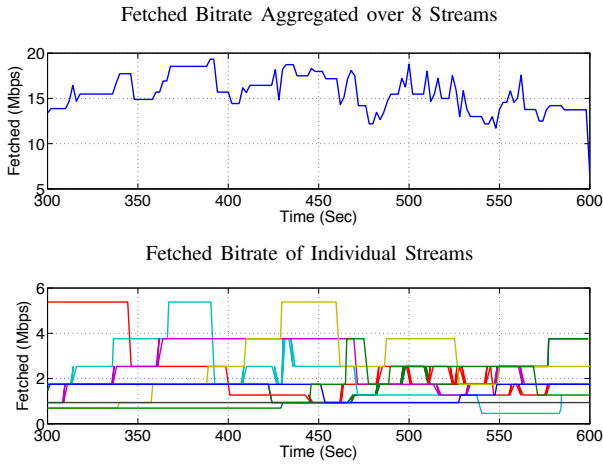[1] Data over cable service interface specification.

Fig. 2. Video bitrate fluctuates when 8 Apple HLS clients compete at a 24-Mbps link. Accompanied with the fluctuation are frequent video playout stalls, which are not shown in the plot. For more detailed experimental setup, refer to §VII-B.
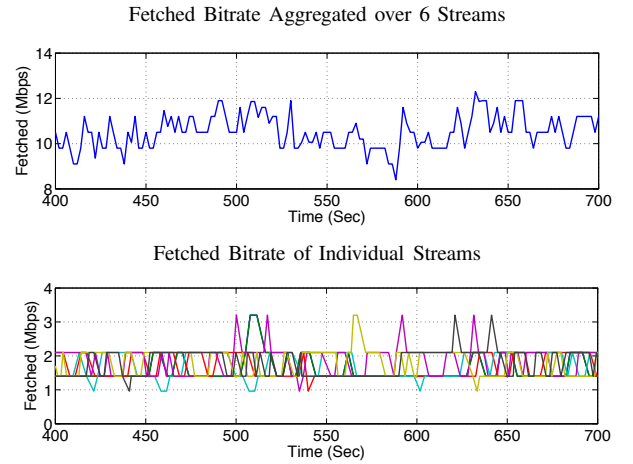

Fig. 3. Video bitrate fluctuates when 6 Adobe HDS clients compete at a 10-Mbps link. Accompanied with the fluctuation is frequent video playout stalls, which are not shown in the plot. For more detailed experimental setup, refer to §VII-B.

While a simple rate adaptation algorithm might work fairly well for the case where a single HAS stream operates alone or shares bandwidth with non-HAS traffic, recent studies [14], [3] have reported undesirable behaviors when multiple HAS streams compete for bandwidth at a bottleneck link. For example, while studies have suggested that significant video quality variation over time is undesirable for a viewer's quality of experience [20], in [14] the authors reported unstable video bitrate selection and unfair bandwidth sharing among three Microsoft Smooth Streaming clients (or Smooth clients for short) sharing a 3-Mbps link. In our own test bed experiments (see Fig. 1), we observed significant and regular video bitrate oscillation when multiple Smooth clients share a bottleneck link. We also found that oscillation behavior persists under a wide range of parameter settings, including the number of players, link bandwidth, start time of clients, heterogeneous RTTs, random early detection (RED) queueing parameters, the use of weight fair queueing (WFQ), the presence of moderate web-like cross traffic, etc. When extending our tests to other types of clients, we found somewhat different behaviors – neither the Apple HLS clients nor the Adobe HDS clients show oscillation on the aggregate fetched rate as regular as the Smooth clients, yet individual clients exhibit video bitrate fluctuations and frequent video playout stalls (see Fig. 2 and Fig. 3, respectively).

Our study shows that these rate oscillation and instability behaviors are not incidental – they are *symptoms* of a much more fundamental limitation of the conventional HAS rate adaptation algorithms, in which *the TCP downloading throughput observed by a client is directly taken as its fair share of the network bandwidth*. This fundamental problem would also impact a HAS client's ability to avoid buffer underrun when the bandwidth suddenly drops, which results in video playout stalls. In brief, the problem derives from the discrete nature of HAS video bitrates and the resulting mismatch between the fetched bitrate and the network bandwidth. This leads to link undersubscription and on-off downloading patterns. The off-intervals then become a source of ambiguity for a client to correctly perceive its fair share of the network
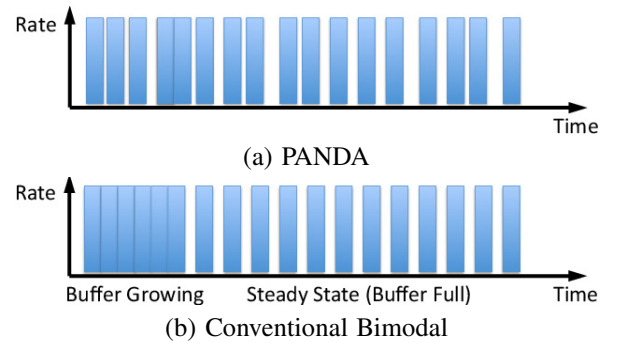

Fig. 4. Illustration of PANDA's fine-granular request intervals vs. a conventional algorithm's bimodal request intervals.

bandwidth, thus preventing the client from making accurate rate adaptation decisions[2].

B. Overview of Solution

To overcome this fundamental limitation, we envision a solution based on a "probe and adapt" principle. In this approach, the TCP downloading throughput is taken as an input *only when* it is an accurate indicator of the fair-share bandwidth. This usually happens when the network is oversubscribed (or congested) and the off-intervals are absent. In the presence of off-intervals, the algorithm constantly *probes*[3] the network bandwidth by incrementing its sending rate, and prepares to back off once it experiences congestion. This new mechanism shares the same spirit with TCP's congestion control, but it operates independently at the application layer and at a per-segment rather than a per-RTT time scale. We present PANDA (Probe AND Adapt) – a client-side rate adaptation algorithm – as a specific implementation of this principle.

[2]In [3], Akhshabi et al. have reached similar conclusions. But they identify the off-intervals instead of the TCP throughput-based measurement as the root cause. Their sequel work [4] attempts to tackle the problem from a very different angle using traffic shaping. In §VIII, we provide a justification for our design choice.

[3]By probing, we refer to small trial increment of data rate, instead of sending auxiliary piggybacking traffic.

TABLE I
NOTATIONS USED IN THIS PAPER

Notation	Explanation
w	Probing additive increase bitrate
κ	Probing convergence rate
α	Smoothing convergence rate
β	Client buffer convergence rate
Δ	Quantization margin
ϵ	Multiplicative safety margin
τ	Video segment duration (in video time)
B	Client buffer duration (in video time)
$B_{\min}; B_{\max}$	Minimum/maximum client buffer duration
T	Actual inter-request time
$\hat{T}$	Target inter-request time
$\tilde{T}$	Segment download duration
x	Actual average data rate
$\hat{x}$	Target average data rate (or bandwidth share)
$\hat{y}$	Smoothed version of $\hat{x}$
$\tilde{x}$	TCP throughput measured, $\tilde{x} := \frac{r \cdot \tau}{T}$
$\mathcal{R}$	Set of video bitrates $\mathcal{R} := \{R_1, ..., R_L\}$
r	Video bitrate available from $\mathcal{R}$
$S(\cdot)$	Rate smoothing function
$Q(\cdot)$	Video bitrate quantization function

Probing constitutes fine-tuning the requested network data rate, with continuous variation over a range. By nature, the available video bitrates in HAS can only be discrete. A main challenge in our design is to create a continuous decision space out of the discrete video bitrate. To this end, we propose to *fine-tune the intervals between consecutive segment download requests* such that the *average data rate* sent over the network is a continuous variable (see Fig. 4 for an illustrative comparison with the conventional scheme). Consequently, instead of directly tuning the video bitrate, we probe the bandwidth based on the average data rate, which in turn determines the selected video bitrate and the fine-granularity inter-request time.

There are various benefits associated with the probe-and-adapt approach. First, it avoids the pitfall of inaccurate bandwidth estimation. Having a robust bandwidth measurement to begin with gives the subsequent operations improved discriminative power (for example, strong smoothing of the bandwidth measurement is no longer required, leading to better responsiveness). Second, with constant probing via incrementing the rate, the network bandwidth can be more efficiently utilized. Third, it ensures that the bandwidth sharing converges towards fair share (i.e., the same or adjacent video bitrate) among competing clients. Lastly, an innate feature of the probe-and-adapt approach is *asymmetry of rate shifting –* PANDA is equipped with conservative bitrate level upshift but more responsive downshift. Responsive downshift facilitates fast recovery from sudden bandwidth drops, and thus can effectively mitigate the danger of playout stalls caused by buffer underrun.

C. Paper Organization

In the rest of the paper, we first give a brief overview of the related work (§II). After formalizing the problem (§III), we first introduce a method to characterize the conventional rate adaptation algorithms (§IV), based on which we analyze the root cause of its problems (§V). We then introduce our probe-and-adapt approach (§VI) to directly address the root cause, and present the PANDA rate adaptation algorithm as a

concrete implementation of this idea. We provide comprehensive performance evaluations (§VII). We conclude the paper with final remarks and discussion of future work (§VIII).

II. RELATED WORK

AIMD Principle: The design of the probing mechanism in PANDA shares similarity with Jacobson's additive-increase-multiplicative-decrease (AIMD) principle for TCP congestion control [12]. Kelly's framework on network rate control [15] provides a theoretical justification for the AIMD principle, and proves its stability in the general network setup.

HAS Measurement Studies: Various research efforts have focused on understanding the behavior of several commercially deployed HAS systems. One such example is [2], where the authors characterize and evaluate HTTP streaming players such as Microsoft Smooth Streaming, Netflix, and Adobe OSMF via experiments in controlled settings. The first measurement study to consider HAS streaming in the multi-client scenarios is [3]. The authors identify the root cause of the player's rate oscillation/fluctuation problem as the existence of on-off patterns in HAS traffic. In [11], the authors measure behavior of commercial video streaming services, i.e., Hulu, Netflix, and Vudu, when competing with other long-lived TCP flows. The results reveal that inaccurate estimations can trigger a feedback loop leading to undesirably low-quality video. In [8], the authors conduct a measurement study on Akamai's HAS service. The study shows that an asymmetric rate shift mechanism has been implemented, where the downshift resembles a multiplicative decrease function.

Existing HAS Designs: To improve the performance of adaptive HTTP streaming, several rate adaptation algorithms [17], [21], [22], [19], [18] have been proposed, which, in general, fit into the four-step model discussed in Section III-B. In [13], a sophisticated Markov Decision Process (MDP) is employed to compute a set of optimal client strategies in order to maximize viewing quality. The MDP requires the knowledge of network conditions and video content statistics, which may not be readily available. Control-theoretical approaches, including use of a PID controller, are also considered by several works [6], [21], [22]. A PID controller with appropriate parameter choice can improve streaming performance. Server-side mechanisms are also advocated by some works [10], [4]. Two designs have been considered to address the multi-client issues: in [4], a rate-shaping approach aiming to eliminate the off-intervals, and in [14], a client rate adaptation algorithm design implementing a combination of randomization, stateful rate selection and harmonic mean based averaging.

III. PROBLEM MODEL

In this section, we formalize the problem by first describing a representative HAS server-client interaction process. We then outline a four-step model for an HAS rate adaptation algorithm. This will allow us to compare the proposed PANDA algorithm with its conventional counterpart. Table I lists the main notations used in this paper.
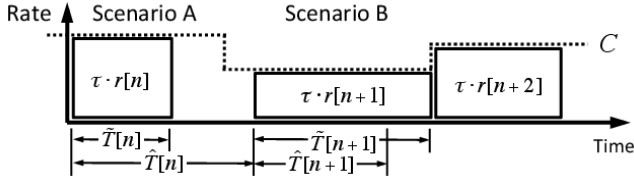
Fig. 5. The HAS segment downloading process.

A. Process of HAS Server-Client Interaction

Consider that a video stream is chopped into segments of τ seconds each. Each segment has been pre-encoded at L video bitrates, all stored at a server. Denote by $\mathcal{R} := \{R_1, ..., R_L\}$ the set of available video bitrates, with $0 < R_\ell < R_m$ for $\ell < m$.

For each client, the streaming process is divided into sequential segment downloading steps $n = 1, 2, ...$. The process we consider here generalizes the process used by conventional HAS clients by further incorporating variable durations between consecutive segment requests. Refer to Fig. 5. At the beginning of each download step n, a rate adaptation algorithm:

- Selects the video bitrate of the next segment to be downloaded, $r[n] \in \mathcal{R}$;
- Specifies how much time to give for the current download, until the next download request (i.e., the inter-request time), $\hat{T}[n]$.

The client then initiates an HTTP GET request to the server for the segment of sequence number n and video bitrate $r[n]$, and the downloading starts immediately. Let $\tilde{T}[n]$ be the *download duration* – the time required to complete the download. Assuming that no pipelining of downloading is involved, the next download step starts after time

$$T[n] = \max(\hat{T}[n], \tilde{T}[n]), \qquad (1)$$

where $T[n]$ is the *actual inter-request time*. That is, if the download duration $\tilde{T}[n]$ is shorter than the target delay $\hat{T}[n]$, the client waits time $\hat{T}[n] - \tilde{T}[n]$ (i.e., the off-interval) before starting the next downloading step (Scenario A); otherwise, the client starts the next download step immediately after the current download is completed (Scenario B).

Typically, a rate adaptation algorithm also measures its *TCP throughput* $\tilde{x}$ during the segment downloading, via:

$$\tilde{x}[n] := \frac{r[n] \cdot \tau}{\tilde{T}[n]}. \qquad (2)$$

The downloaded segments are stored in the client buffer. After playout starts, the buffer is consumed by the video player at a natural rate of one video second per real second on average. Let $B[n]$ be the buffer duration (measured in video time) at the end of step n. Then the buffer dynamics can be characterized by:

$$B[n] = \max\left(0, B[n-1] + \tau - T[n]\right). \qquad (3)$$

B. Four-Step Model

We present a four-step model for an HAS rate adaptation algorithm, generic enough to encompass both the conventional

---

Algorithm 1 Conventional

At the beginning of each downloading step n:

1) Estimate the bandwidth share $\hat{x}[n]$ by equating it to the measured TCP throughput:

$$\hat{x}[n] = \tilde{x}[n-1]. \qquad (4)$$

2) Smooth out $\hat{x}[n]$ to produce filtered version $\hat{y}[n]$ by

$$\hat{y}[n] = S(\{\hat{x}[m] : m \le n\}). \qquad (5)$$

3) Quantize $\hat{y}[n]$ to the discrete video bitrate $r[n] \in \mathcal{R}$ by

$$r[n] = Q\left(\hat{y}[n]; ...\right). \qquad (6)$$

4) Schedule the next download request depending on the buffer fullness:

$$\hat{T}[n] = \begin{cases} 0, & B[n-1] < B_{\max}, \\ \tau, & \text{otherwise.} \end{cases} \qquad (7)$$

---

algorithms (e.g., [17], [21], [22], [19], [18]) and the proposed PANDA algorithm. In this model, a rate adaptation algorithm proceeds in the following four steps.

- *Estimating*. The algorithm starts by estimating the network bandwidth $\hat{x}[n]$ that can legitimately be used.
- *Smoothing*. $\hat{x}[n]$ is then noise-filtered to yield the smoothed version $\hat{y}[n]$, with the aim of removing outliers.
- *Quantizing*. The continuous $\hat{y}[n]$ is then mapped to the discrete video bitrate $r[n] \in \mathcal{R}$, possibly with the help of side information such as client buffer size, etc.
- *Scheduling*. The algorithm selects the target interval until the next download request, $\hat{T}[n]$.

IV. CONVENTIONAL APPROACH

Using the four-step model above, in this section we introduce a scheme to characterize a conventional rate adaptation algorithm, which will serve as a benchmark.

To the best of our knowledge, almost all of today's commercial HAS players[4] implement the *measuring* and *scheduling* parts of the rate adaptation algorithm in a similar way, though they may differ in their implementation of the smoothing and quantizing parts of the algorithm. Our claim is based on a number of experimental studies of commercial HAS players [2], [11], [14]. The scheme described in Algorithm 1 characterizes their essential ideas.

First, the algorithm equates the currently available bandwidth share $\hat{x}[n]$ to the past TCP throughput $\tilde{x}[n-1]$ (as in (2)) observed during the on-interval $\tilde{T}[n-1]$. As the bandwidth is inferred reactively based on the previous downloads, we refer to this as *reactive bandwidth estimation*.

The algorithm then obtains a filtered version $\hat{y}[n]$ using a smoothing function $S(\cdot)$ that takes as input the measurement history $\{\hat{x}[m] : m \le n\}$, as described in (5). Various filtering methods are possible, such as sliding-window moving average, exponential weighted moving average (EWMA) or harmonic mean [14].

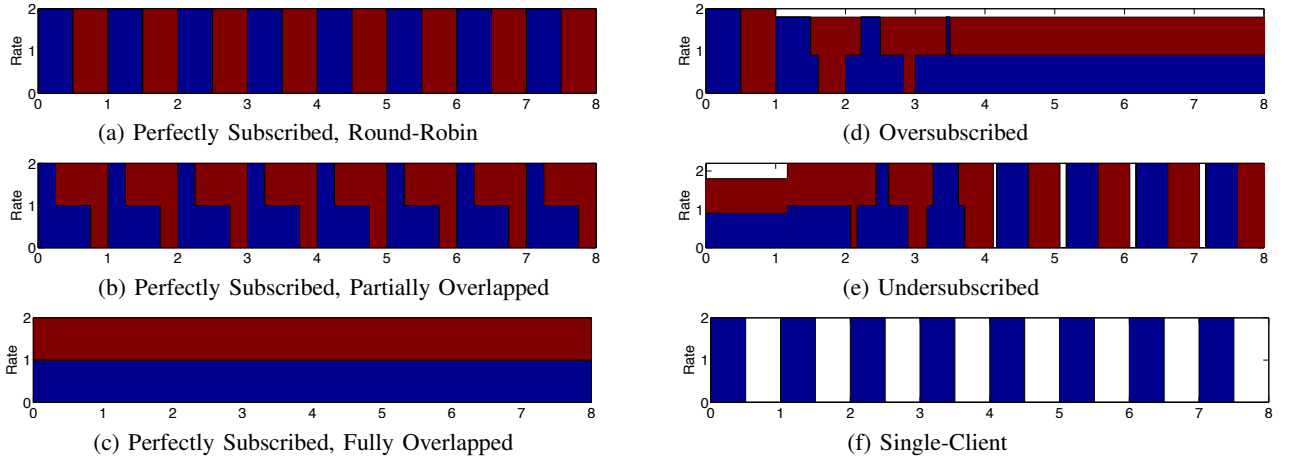[4]In this paper, the terms "HAS player" and "HAS client" are used interchangeably.

Fig. 6. Illustration of various bandwidth sharing scenarios. In (a), (b) and (c), the link is perfectly subscribed. In (d), the bandwidth sharing starts with round-robin mode but then link becomes oversubscribed. In (e), the bandwidth sharing starts with fully overlapped mode when the link is oversubscribed. Starting from the second round, the link becomes undersubscribed. In (f), a single client is downloading, and the downloading on-off pattern exactly matches that of the blue segments in (a).

The next step maps the continuous $\hat{y}[n]$ to a discrete video bitrate $r[n] \in \mathcal{R}$ using a quantization function $Q(\cdot)$. In general, $Q(\cdot)$ can also incorporate side information, including the past fetched bitrates $\{r[m] : m < n\}$ and the buffer history $\{B[m] : m < n\}$.

Lastly, the algorithm determines the target inter-request time $\hat{T}[n]$. In (7), $\hat{T}[n]$ is a mechanical function of the buffer duration $B[n-1]$. If $B[n-1]$ is less than a pre-defined maximum buffer $B_{\max}$, $\hat{T}[n]$ is set to 0, and by (1), the next segment downloading starts right after the current download is finished; otherwise, the inter-request time is set to the video segment duration τ, to stop the buffer from further growing. This creates two distinct modes of segment downloading – the *buffer growing* mode and the *steady-state* mode, as shown in Fig. 4(b). We refer to this as the *bimodal download scheduling*.

V. ANALYSIS OF THE CONVENTIONAL APPROACH

In this section, we take a deep dive into the conventional rate adaptation algorithms and study their limitations.

A. Bandwidth Cliff Effect

As we have seen in the previous section, conventional rate adaptation algorithms use reactive bandwidth estimation (4) that equates the estimated bandwidth share to the TCP throughput observed during the on-intervals. In the presence of competing HAS clients, however, the TCP throughput does not always faithfully represent the fair-share bandwidth. In this section, we extend the analysis first presented in [3].

First, we illustrate with simple examples. Fig. 6 (a) - (e) show the various scenarios of how a link can be shared by two HAS clients in steady-state mode. We consider three different scenarios: perfect link subscription, link oversubscription and link undersubscription. We assume ideal TCP behavior, i.e., perfectly equal sharing of the available bandwidth when the transfers overlap.

Perfect Subscription: In perfect link subscription, the total amount of traffic requested by the two clients perfectly fills the link. (a), (b) and (c) illustrate three different modes of bandwidth sharing, depending on the starting time of downloads

relative to each other. Essentially, under perfect subscription, there are unlimited number of bandwidth sharing modes.

Oversubscription: In (d), the two clients start with round-robin mode and perfect subscription. Starting from the second round of downloading, the bandwidth is reduced and the link becomes oversubscribed, i.e., each client requests segments larger than its current fair-share portion of the bandwidth. This will result in unfinished downloads at the end of each downloading round. Then, the unfinished segment will start overlapping with segments of the next round. This repeats and the downloading will become more and more overlapped, until all the clients enter the fully overlapped mode.

Undersubscription: In (e), initially the bandwidth sharing is in fully overlapped mode, and the link is oversubscribed. Starting from the second round, the bandwidth increases and the link becomes undersubscribed. Then the clients start filling up each other's off-intervals, until a link utilization gap emerges. The bandwidth sharing will eventually converge to a mode which is determined by the download start times.

In any case, the measured TCP throughput (as in (2)) faithfully represents the fair-share bandwidth *only when* the bandwidth sharing is in the fully overlapped mode; in all other cases the TCP throughput overestimates the fair-share bandwidth. Thus, most of the time, the bandwidth estimate is accurate when the link is oversubscribed. Bandwidth overestimation occurs when the link is undersubscribed or perfectly subscribed. In general, when the number of competing clients is n, the bandwidth overestimation ranges from one to n times the fair-share bandwidth.

The preceding simple examples assume idealized TCP behavior which abstracts away the complexity of TCP congestion control dynamics. Our assumption is justifiable based on the fact that TCP and HAS operate at very different time scales – TCP dynamics occur in milli-seconds whereas HAS bandwidth sharing converges in seconds. It is easy to verify that similar behavior above occurs with real TCP connections. To see this, we conducted a simple test bed experiment as follows. We implemented a "thin client" to mimic an HAS client in the steady-state mode. Each thin client repeatedly
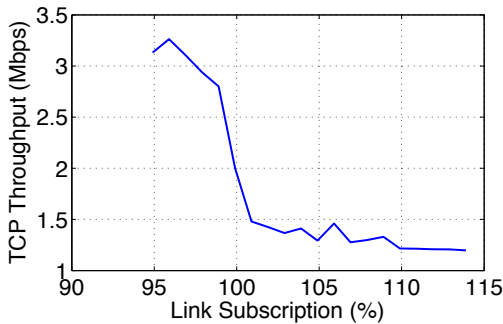
Fig. 7. Bandwidth cliff effect: measured TCP throughput vs. link subscription rate for 100 thin clients sharing a 100-Mbps link. Each thin client repeatedly downloads a segment every $\tau = 2$ seconds.
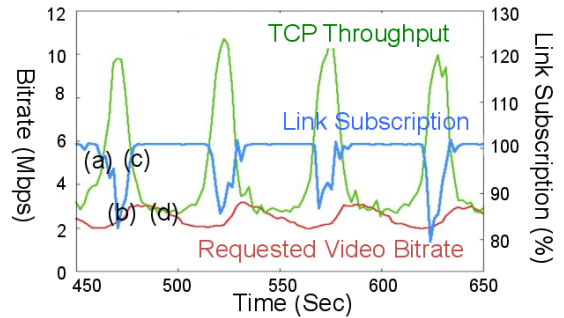

Fig. 8. Illustration of vicious cycle of video bitrate oscillation. This plot is obtained with 36 Smooth clients sharing a 100-Mbps link. For experimental setup, refer to §VII-B.

downloads a segment every 2 seconds. We run 100 instances of the thin client sharing a bottleneck link of 100 Mbps, each with a starting time randomly selected from a uniform distribution between 0 and 2 seconds. Fig. 7 plots the measured average TCP throughput as a function of the link subscription rate. We observe that when the link subscription is below 100%, the measured throughput is about 3x the fair-share bandwidth of ~1 Mbps. When the link subscription is above 100%, the measured throughput successfully predicts the fair-share bandwidth quite accurately. We refer to this sudden transition from overestimation to fairly accurate estimation of the bandwidth share at 100% subscription as the *bandwidth cliff effect*.

We summarize our findings as follows:

- Link oversubscription (even a slight one) converges to fully overlapped bandwidth sharing and accurate bandwidth estimation.
- Link undersubscription (even a slight one) converges to a bandwidth sharing pattern determined by the download start times and bandwidth overestimation.
- In perfect link subscription, there exist unlimited bandwidth sharing modes, almost always leading to bandwidth overestimation.

B. Video Bitrate Oscillation

With an understanding of the bandwidth cliff effect, we are now in a good position to explain the bitrate oscillation observed in the Smooth clients, as shown in Fig. 1.

Fig. 8 illustrates this process. When the client buffer reaches the maximum level B_{max}, by (7), off-intervals start to emerge. The link becomes undersubscribed, leading to bandwidth overestimation (a). This triggers the upshift of requested video bitrate (b). As the available bandwidth cannot keep up with the video bitrate, the buffer falls below B_{max}. By (7), the client falls back to the buffer growing mode and the off-intervals disappear, in which case the link again becomes oversubscribed and the measured throughput starts to converge to the fair-share bandwidth (c). Lastly, due to the quantization effect, the requested video bitrate falls below the fair-share bandwidth (d), and the client buffer starts growing again, completing one oscillation cycle.

An intriguing question is, why neither the HLS nor the HDS clients exhibit similar regular oscillation as the Smooth clients? Our experience shows that when a strong smoothing filter is applied to the measured TCP throughput (as in the Smooth case), the resulting clients tend to demonstrate synchronized behavior; when the smoothing filter is weak (as in HLS and HDS), the resulting clients tend to be more volatile, producing more unpredictable bitrate shifts. Nevertheless, it is the same underlying effect, i.e., the bandwidth cliff, that drives the video bitrate to shift up and down.

C. Fundamental Limitation

The bandwidth overestimation phenomenon reveals a more general and fundamental limitation of the class of conventional reactive bandwidth estimation approaches discussed so far. As video bitrates are chosen solely based on measured TCP throughput from past segment downloads during the on-intervals, such decisions completely ignore the network conditions during the off-intervals. This leads to an *ambiguity of client knowledge* of available network bandwidth during the off-intervals, which, in turn, hampers the adaptation process.

To illustrate this point, consider two alternative scenarios as depicted in Figs. 6 (f) and (a). In (f), the client downloading the blue video segments occupies the link alone; in (a), it shares the same link with a competing client downloading the red video segments. Note that the on/off-intervals for all the blue video segments follow exactly the same pattern in both scenarios. Consequently, the client observes exactly the same TCP throughput measurement over time. If the client would obtain a complete picture of the network, it would know to upshift its video bitrate in (f) but retain its current bitrate in (a). In practice, however, an individual client cannot distinguish between these two scenarios, hence, is bound to the same behavior in both.

Note that as long as the off-intervals persist, such *ambiguity in client knowledge* is inherent to the bandwidth measurement step in a network with competing streams. It cannot be resolved or remedied by improved filtering, quantization, or scheduling steps performed later in the client adaptation algorithm. Moreover, the bandwidth cliff effect, as discussed in Section V-A, suggests that the bandwidth overestimation problem does not improve with more clients, and that it can introduce large errors even with slight link undersubscription.

Instead, the client needs to take a more proactive approach in adapting the video bitrate — whenever it is known that the client knowledge is impaired, it must *avoid* using such knowledge in bandwidth estimation. A way to distinguish the

case when the knowledge is impaired from when it is not, is to *probe* the network subscription by small increment of its data sending rate. We describe one algorithm that follows such an alternative approach in the next section.

VI. PROBE-AND-ADAPT APPROACH

In this section, we introduce our proposed probe-and-adapt approach to directly address the root cause of the conventional algorithms' problems. We begin the discussion by laying out the design goals that a rate adaptation algorithm aims to achieve. We then describe the PANDA algorithm as an embodiment of the probe-and-adapt approach, and provide its functional verification using experimental traces.

A. Design Goals

Designing an HAS rate adaptation algorithm involves trade-offs among a number of competing goals. It is not legitimate to optimize one goal (e.g., stability) without considering its tradeoff factors. From an end-user's perspective, an HAS rate adaptation algorithm should be designed to meet these criteria:

- *Avoiding buffer underrun.* Once the playout starts, buffer underrun (i.e., complete depletion of buffer) leads to a playout stall. Empirical study [9] has shown that buffer underrun may have the most severe impact on a user's viewing experience. To avoid it, some minimal buffer level must be maintained at all times[5], and the adaptation algorithm must be highly responsive to network bandwidth drops.
- *High quality smoothness.* In the simplest setting without considering visual perceptual models, high video quality smoothness translates into avoiding both frequent and significant video bitrate shifts among available video bitrate levels [14], [20].
- *High average quality.* High average video quality dictates that a client should fetch high-bitrate segments as much as possible. Given a fixed network bandwidth, this translates into high network utilization.
- *Fairness.* In the simplest setting, fairness translates into equal network bandwidth sharing among competing clients.

Note that this list above is non-exhaustive. Other criteria, such as low playout startup latency, are also important factors impacting user's viewing experience.

B. PANDA Algorithm

In this section, we discuss the PANDA algorithm. PANDA does *not* require more measurements than a conventional scheme – all it needs are the TCP throughput as calculated in (2) and the client buffer size. Furthermore, one only needs to make modifications to two steps out of the four-step model as discussed in Section III-B. Thus, it is fairly straightforward to implement PANDA based upon modifying an existing scheme.

[5]Note that, however, the buffer level must also have an upper bound, for a few different reasons. In live streaming, the end-to-end latency from the real-time event to the event being displayed on user's screen must be reasonably short. In video-on-demand, the maximum buffered video must be limited to avoid wasted network usage in case of an early termination of playback and to limit memory usage.

Algorithm 2 PANDA

At the beginning of each downloading step n:

1) Estimate the bandwidth share $\hat{x}[n]$ by

$$\frac{\hat{x}[n]-\hat{x}[n-1]}{T[n-1]} = \kappa \cdot (w - \max(0, \hat{x}[n-1] - \tilde{x}[n-1] + w)). \tag{8}$$

2) Smooth out $\hat{x}[n]$ to produce filtered version $\hat{y}[n]$ by

$$\hat{y}[n] = S(\{\hat{x}[m] : m \le n\}). \tag{9}$$

3) Quantize $\hat{y}[n]$ to the discrete video bitrate $r[n] \in \mathcal{R}$ by

$$r[n] = Q(\hat{y}[n]; ...). \tag{10}$$

4) Schedule the next download request via

$$\hat{T}[n] = \frac{r[n] \cdot \tau}{\hat{y}[n]} + \beta \cdot (B[n-1] - B_{\min}). \tag{11}$$

Compared to the reactive bandwidth estimation used by a conventional rate adaptation algorithm, PANDA uses a more proactive probing mechanism. By probing, PANDA determines a target average data rate $\hat{x}$. This average data rate is subsequently used to determine the video bitrate r to be fetched, and the interval $\hat{T}$ until the next segment download request.

The PANDA algorithm is described in Algorithm 2, and a block diagram interpretation of the algorithm is shown in Fig. 9. Compared to the conventional algorithm in Algorithm 1, we only need to make modifications to the *estimating* and *scheduling* steps – we replace (4) with (8) for estimating the bandwidth share, and (7) with (11) for scheduling the next download request. We now focus on elaborating each of these two modifications.

In the estimating step, (8) is designed to directly address the root cause that leads to the video bitrate oscillation/fluctuation phenomenon. Based on the insights obtained from §V-A, when the link becomes undersubscribed, the direct TCP throughput estimate $\tilde{x}$ becomes inaccurate in predicting the fair-share bandwidth, and thus should be avoided. Instead, the client continuously increments the target average data rate $\hat{x}$ by $\kappa \cdot w$ per unit time as a probe of the available capacity. Here κ is the probing convergence rate and w is the additive increase rate. The algorithm keeps on monitoring the TCP throughput $\tilde{x}$ according to (2) once per segment downloading cycle, and compares it against the target average data rate $\hat{x}$. Note that the term w is present in $\hat{x} - \tilde{x} + w$ such that the target data rate $\hat{x}$ matches the TCP throughput $\tilde{x}$ at equilibrium (see the Appendix for the equilibrium analysis). If $\tilde{x} > \hat{x} + w$, $\tilde{x}$ would not be informative, since in this case the link may still be undersubscribed and $\tilde{x}$ may overestimate the fair-share bandwidth. Thus, its impact is suppressed by the $\max(0, \cdot)$ function. But if $\tilde{x} < \hat{x} + w$, then TCP throughput cannot keep up with the target average data rate indicates that congestion has occurred. This is when the target data rate $\hat{x}$ should back off. The reduction imposed on $\hat{x}$ is made proportional to $\hat{x} - \tilde{x} + w$. Intuitively, the lower the measured TCP throughput $\tilde{x}$, the more reduction that needs to be imposed on $\hat{x}$. This design makes our rate adaptation algorithm very agile to bandwidth changes.
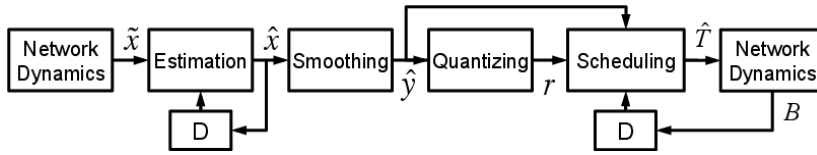
Fig. 9. Block diagram for PANDA (Algorithm 2). Module D represents delay of one adaptation step.

PANDA's probing mechanism shares similarities with TCP's congestion control [12], and has an additive-increase-multiplicative-decrease (AIMD) interpretation: $\kappa \cdot w$ is the additive increase term, and $-\kappa \cdot \max(0, \hat{x}[n-1] - \tilde{x}[n-1] + w)$ can be interpreted as the multiplicative decrease term. The main difference is that in TCP, congestion is indicated by packet losses (TCP Reno) or increased round-trip time (delay-based TCP), whereas in (8), congestion is indicated by the reduction of measured TCP throughput. This AIMD property ensures that PANDA is able to efficiently utilize the network bandwidth, and in the presence of multiple clients, the bandwidth for each client eventually converges to fair-share status[6].

In the scheduling step, (11) aims to determine the target inter-request time $\hat{T}[n]$. By right, $\hat{T}[n]$ should be selected such that the smoothed target average data rate $\hat{y}[n]$ is equal to $\frac{r[n] \cdot \tau}{\hat{T}[n]}$. But additionally, the selection of $\hat{T}[n]$ should also drive the buffer $B[n]$ towards a minimum reference level $B_{\min} > 0$, so the second term is added to the right hand side of (11), where $\beta > 0$ controls the convergence rate.

One distinctive feature of the PANDA algorithm is its hybrid closed-loop/open-loop design. Refer to Fig. 9. In this system, (8) forms a closed loop by itself that determines the target average data rate $\hat{x}$. (11) forms a closed loop by itself that determines the target inter-request time $\hat{T}$. Overall, the estimating, smoothing, quantizing and scheduling steps together form an open loop (ignoring the network dynamics' feedback). The main motivation behind this design is to reduce the bitrate shifts associated with quantization. Since quantization is excluded from the closed loop of $\hat{x}$, it allows $\hat{x}$ to settle in a steady state. Since $r[n]$ is a deterministic function of $\hat{x}[n]$, it can also settle in a steady state.

In the Appendix, we present an equilibrium and stability analysis of PANDA. We summarize the main results as follows. Our equilibrium analysis shows that at steady state, the system variables settle at

$$\hat{x}_o = \tilde{x}_o \tag{12}$$
$$= \hat{y}_o$$
$$r_o = Q(\hat{x}_o; ...)$$
$$B_o = \left(1 - \frac{r_o}{\hat{y}_o}\right) \cdot \frac{\tau}{\beta} + B_{\min}, \tag{13}$$

where the subscript o denotes value of variables at equilibrium, assuming $\hat{y}$ is an unbiased estimator of $\hat{x}$. Our stability analysis shows that for the system to converge towards the steady state, it is necessary to have:

$$\kappa < \frac{2}{\tau} \tag{14}$$
$$\Delta \geq 0, \tag{15}$$

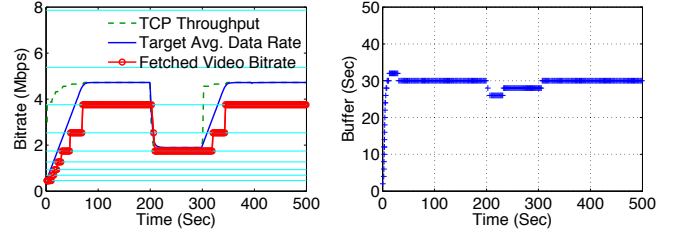[6]Assuming the underlying TCP is fair (e.g., equal RTTs).

Fig. 10. A PANDA client adapts its video bitrate under a bandwidth-varying link. The bandwidth is initially at 5 Mbps, drops to 2 Mbps at 200 seconds and rises back to 5 Mbps at 300 seconds.

where Δ is a parameter associated with the quantizer $Q(\cdot)$, referred to as the *quantization margin*, i.e., the selected discrete rate r must satisfy

$$r[n] \leq \hat{y}[n] - \Delta. \tag{16}$$

C. Functional Verification

We verify the behavior of PANDA using experimental traces. For detailed experiment setup (including the selection of function $S(\cdot)$ and $Q(\cdot)$), refer to §VII-B.

First, we evaluate how a single PANDA client adjusts its video bitrate as the the available bandwidth varies over time. In Fig. 10, we plot the TCP throughput $\tilde{x}$, the target average data rate $\hat{x}$, the fetched video bitrate r and the client buffer B for a duration of 500 seconds, where the bandwidth drops from 5 to 2 Mbps at 200 seconds, and rises back to 5 Mbps at 300 seconds. Initially, the target average data rate $\hat{x}$ ramps up gradually over time; the fetched video bitrate r also ramps up correspondingly. After the initial ramp-up stage, $\hat{x}$ settles in a steady state. It can be observed that at steady state, $\hat{x}$ matches $\tilde{x}$, which is consistent with (12). Similarly, the buffer B also settles in a steady state, and after plugging in all the parameters, one can verify that the steady state of buffer (13) also holds. At 200 seconds, when the bandwidth suddenly drops, the fetched video bitrate quickly drops to the desirable level. With this quick response, the buffer hardly drops. This property makes PANDA favorable for live streaming applications. When the bandwidth rises back to 5 Mbps at 300 seconds, the fetched video bitrate gradually ramps up to the original level.

Note that, in practical implementation, we can further add startup logic to improve PANDA's rate ramp-up speed at the initial stage. Since off-intervals would not need to appear before the buffer reaches the minimum reference level, a conventional rate adaptation algorithm based on TCP throughput measurement would suffice.

The more intriguing question is whether PANDA could effectively mitigate the bitrate fluctuation observed in the conventional HAS players. We conduct an experiment with the same setup as the experiment shown in Fig. 1, except that
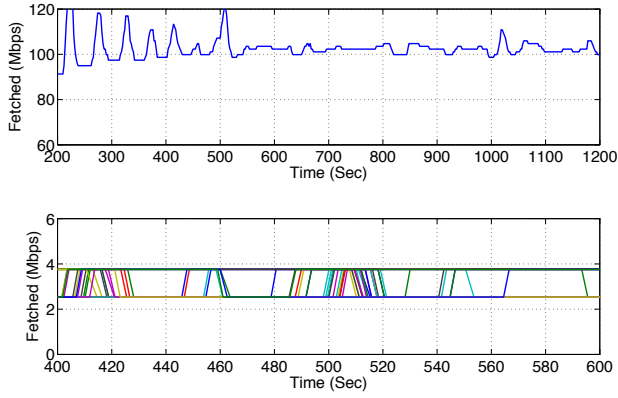
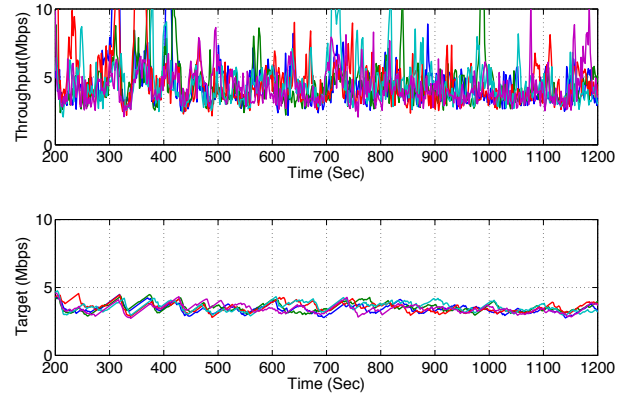Fig. 11. 36 PANDA clients compete at a 100-Mbps link in steady state.

Fig. 12. The traces of the TCP throughput and the unfiltered target average data rate of 36 PANDA clients compete at a 100-Mbps link in steady state. The traces of the first five clients are plotted.
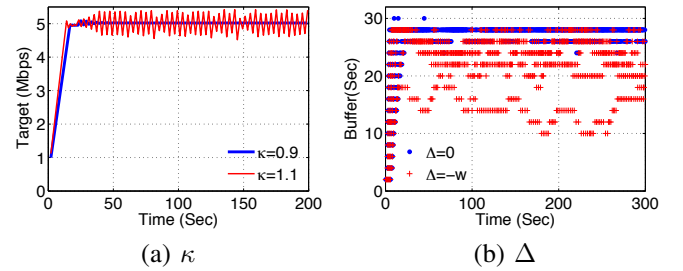

(a) κ (b) Δ

Fig. 13. Experimental verification of PANDA's stability criteria. In (a), one PANDA client streams over a link of 5 Mbps. In (b), 10 PANDA clients compete over a 10 Mbps link.

the PANDA player and the Smooth player use slightly different video bitrate levels (due to different packaging methods). The resulting fetched bitrates in aggregate and for each client are shown in Fig. 11. From the plot of the aggregate fetched bitrate, except for the initial fluctuation, the aggregate bitrate closely tracks the available bandwidth of 100 Mbps. Zooming in to the individual streams' fetched bitrates, the fetched bitrates are confined within two adjacent bitrate levels and the number of shifts is much smaller than the Smooth client's case. This affirms that PANDA is able to achieve better stability than the Smooth's rate adaptation algorithm. In §VII, we perform a comprehensive performance evaluation on each adaptation algorithm.

To help the reader develop a better intuition on why PANDA performs better than a conventional algorithm, in Fig. 12 we plot the trace of the measured TCP throughput and the target average data rate for the same experiment as in Fig. 11. Note that the fair-share bandwidth for each client is about 2.8 Mbps. From the plot, the TCP throughput not only grossly overestimates the fair-share bandwidth, it also has a large variation. If used directly, this degree of noisiness gives the subsequent operations a very hard job to extract useful information. For example, one may apply strong filtering to smooth out the bandwidth measurement, but this would seriously affect the responsiveness of the client. When the network bandwidth suddenly drops, the client would not be able to respond quickly enough to reduce its video bitrate, leading to catastrophic buffer underrun. Moreover, the bias is both large and difficult to predict, making any correction to the mean problematic. In comparison, the target average data rate estimated by the probing mechanism is neither biased nor have a large variation, enabling the subsequent operations to easily pick a video bitrate without sacrificing responsiveness.

In Fig. 13, we verify the stability criteria (14) and (15) of PANDA. With $\tau = 2$, the system is stable if $\kappa < 1$. This is demonstrated by Fig. 13 (a), where we show the traces of the target average rate $\hat{x}$ for two κ values 0.9 and 1.1. In Fig. 13 (b), we show that when $\Delta = -w$, the buffer cannot converge towards the reference level of 30 seconds.

VII. PERFORMANCE EVALUATION

In this section, we conduct a set of test bed experiments to evaluate the performance of PANDA against other rate adaptation algorithms.

A. Evaluation Metrics

In §VI-A, we discussed four criteria that are most important for a user's watching experience – i) ability to avoid buffer underruns, ii) quality smoothness, iii) average quality, and iv) fairness. In this paper, we use *buffer undershoot* as the metric for Criterion i), described as follows.

- *Buffer undershoot*: We measure how much the buffer goes down after a bandwidth drop as a indicator of an algorithm's ability to avoid buffer underruns. The less the buffer undershoot, the less likely the buffer will underrun. Let B_o be a reference buffer level (30 seconds for all players in this paper), and $B_{i,t}$ the buffer level for player i at time t. The buffer undershoot for player i at time t is defined as $\frac{\max(0, B_o - B_{i,t})}{B_o}$. The buffer undershoot for player i within a time interval $\mathcal{T}$ (right after a bandwidth drop), is defined as the 90th-percentile value of the distribution of buffer undershoot samples collected during $\mathcal{T}$.

We inherit the metrics defined in [14] – *instability*, *inefficiency* and *unfairness*, as the metrics for Criteria ii), iii) and iv), respectively. We only make a slight modification to the definition of inefficiency. Let $r_{i,t}$ be the video bitrate fetched by player i at time t.

- *Instability*: The instability for player i at time t is $\frac{\sum_{d=0}^{k-1} |r_{i,t-d} - r_{i,t-d-1}| \cdot w(d)}{\sum_{d=0}^{k-1} r_{i,t-d} \cdot w(d)}$, where $w(d) = k - d$ is a weight function that puts more weight on more recent samples. k is selected as 20 seconds.

- *Inefficiency*: Let C be the available bandwidth. [14] defines inefficiency as $\frac{|\sum_i r_{i,t} - C|}{C}$ for player i at time t.

TABLE II
PARAMETERS USED IN EXPERIMENTS

Algorithm	Parameter	Default	Values
PANDA	κ	0.14	0.04,0.07,0.14,0.28,0.42,0.56
	w	0.3	
	α	0.2	0.05,0.1,0.2,0.3,0.4,0.5
	β	0.2	
	ϵ	0.15	0.5,0.4,0.3,0.2,0.1,0
	$B_{\min}$	26	
Conventional	α	0.2	0.01,0.04,0.07,0.1,0.15,0.2
	ϵ	0.15	
	$B_{\max}$	30	
FESTIVE	Window	20	20,15,10,6,3,1
	$targetbuf$	30	

Fig. 14. The network topology configured in the test bed. Local indicates that the bitrate is effectively unbounded and the link delay is 0 ms.

But sometimes the sum of fetched bitrate $\sum_i r_{i,t}$ can be greater than C. To avoid unnecessary penalty in this case, we revise the inefficiency metric to $\frac{\max\left(0, C - \sum_i r_{i,t}\right)}{C}$ for player i at time t.

- *Unfairness*: Let $JainFair_t$ be the Jain fairness index calculated on the rates $r_{i,t}$ at time t over all players. The unfairness at t is defined as $\sqrt{1 - JainFair_t}$.

B. Experimental Setup

HAS Player Configuration: The benchmark players that we use to compare against PANDA are:

- Microsoft Smooth player [5], a commercially available proprietary player. The Smooth players are of version 1.0.837.34 using Silverlight runtime version 4.0.50826. To our best knowledge, the Smooth player as well as the Apple HLS and the Adobe HDS players all use the same TCP throughput measurement mechanism, so we picked the Smooth player as a representative.
- FESTIVE player, which we implemented based on the details specified in [14]. The FESTIVE algorithm is the first known client-side rate adaptation algorithm designed to specifically address the multi-client scenario.
- A player implementing the conventional algorithm (Algorithm 1), which differs from PANDA only in the estimating and scheduling steps.

For fairness, we ensure that PANDA and the conventional player use the same smoothing and quantizing functions. For smoothing, we implemented a *EWMA smoother* of the form: $\frac{\hat{y}[n] - \hat{y}[n-1]}{T[n-1]} = -\alpha \cdot (\hat{y}[n-1] - \hat{x}[n])$, where $\alpha > 0$ is the convergence rate of $\hat{y}[n]$ towards $\hat{x}[n]$. For quantization, we implemented a *dead-zone quantizer* $r[n] = Q(\hat{y}[n], r[n-1])$, defined as follows: Let the upshift threshold be defined as $r_{up} := \max_{r \in \mathcal{R}} r$ subject to $r \leq \hat{y}[n] - \Delta_{up}$, and the downshift threshold as $r_{down} := \max_{r \in \mathcal{R}} r$ subject to $r \leq \hat{y}[n] - \Delta_{down}$, where Δ_{up} and Δ_{down} are the upshift and downshift safety margin respectively, with $\Delta_{up} \geq \Delta_{down} \geq 0$. The dead-zone quantizer updates $r[n]$ as

$$r[n] = \begin{cases} r_{up}, & r[n-1] < r_{up}, \\ r[n-1], & r_{up} \leq r[n-1] \leq r_{down}, \\ r_{down}, & \text{otherwise.} \end{cases} \quad (17)$$

The "dead zone" $[r_{up}, r_{down}]$ created by setting $\Delta_{up} > \Delta_{down}$ mitigates frequent bitrate hopping between two adjacent levels, thus stabilizing the video quality (i.e. hysteresis control). For the conventional player and PANDA, set $\Delta_{up} = \epsilon \cdot \hat{y}$ and
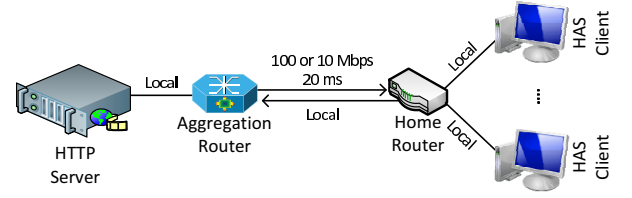
$\Delta_{down} = 0$, where $0 \leq \epsilon < 1$ is the multiplicative safety margin.

Table II lists the default parameters used by each player, as well as their varying values. For fairness, all players attempt to maintain a steady-state buffer of 30 seconds. For PANDA, $B_{\min}$ is selected to be 26 seconds such that the resulting steady-state buffer is 30 seconds (by (13)).

Server Configuration: The HTTP server runs Apache on Red Hat 6.2 (kernel version 2.6.32-220). The Smooth player interacts with an Microsoft IIS server by default, but we also perform experiments of Smooth player interacting with an Apache server on Ubuntu 10.04 (kernel version 2.6.32.21).

Network Configuration: As service provider deployment over a managed network is our primary case of interest, our experiments are configured to highly match the important scenario where a number of HAS flows compete for bandwidth within a DOCSIS bonding group. The test bed is configured as in Fig. 14. The queueing policy used at the aggregation router-home router bottleneck link is the following. For a link bandwidth of 10 Mbps or below, we use random early detection (RED) with $(min\_thr, max\_thr, p) = (30, 90, 0.25)$; if the link bandwidth is 100 Mbps, we use RED with $(min\_thr, max\_thr, p) = (300, 900, 1)$. The video content is chopped into segments of $\tau = 2$ seconds, pre-encoded with $L = 10$ bitrates: 459, 693, 937, 1270, 1745, 2536, 3758, 5379, 7861 and 11321 Kbps. For the Smooth player, the data rates after packaging are slightly different.

C. Performance Tradeoffs

It would not be legitimate to discuss a single metric without minding its impact on other metrics. In this section, we examine the performance tradeoffs among the four metrics of interest. We designed an experimental process under which we can measure all four metrics in a single run. For each run, five players (of the same type) compete at a bottleneck link. The link bandwidth stays at 10 Mbps from 0 seconds to 400 seconds, drops to 2.5 Mbps at 400 seconds and stays there until 500 seconds. We record the instability, inefficiency and unfairness averaged over 0 to 400 seconds over all players, and the buffer undershoot over 400 to 500 seconds averaged over all players. Fig. 15 shows the tradeoff between stability and each of the other criteria – buffer undershoot, inefficiency and unfairness – for each of the types of player. Each data point is obtained via averaging over 20 runs, and each data point represents a different value for one of the parameters of the corresponding algorithm, as indicated in the Values column of Table II.

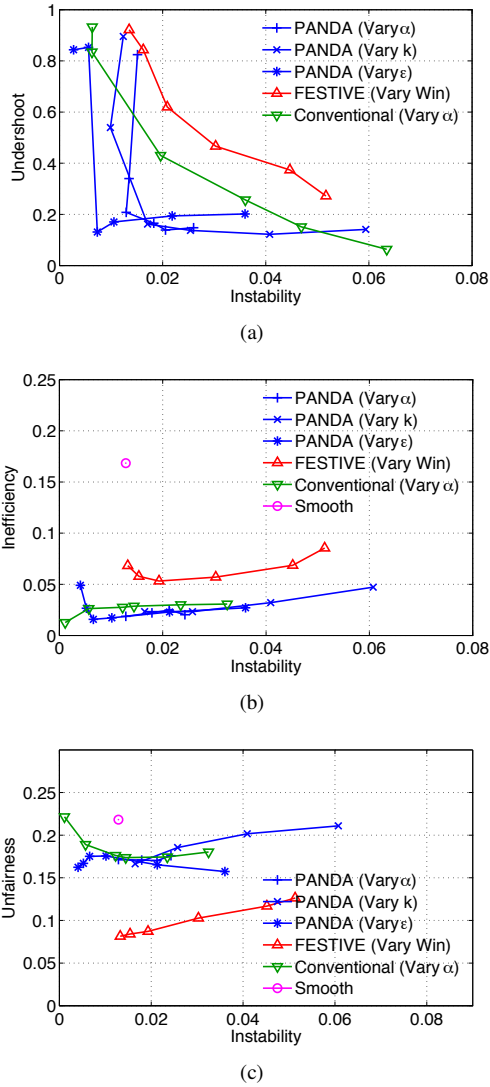For the PANDA player, the three parameters that affect instability the most are: the probing convergence rate κ, the

Fig. 15. The impact of varying instability on buffer undershoot, inefficiency and unfairness for PANDA and other benchmark players.

Fig. 16. Instability, inefficiency and unfairness as the number of clients increases. The link bandwidth is fixed at 10 Mbps.

smoothing convergence rate α and the safety margin ϵ. Fig. 15 (a) shows that as we vary these parameters, the tradeoff curves mostly stay flat (except for at extreme values of these parameters), implying that the PANDA player maintains good responsiveness as the stability is improved. A few factors contribute to this advantage of PANDA: First, as the bandwidth estimation by probing is quite accurate, one does not need to apply strong smoothing. Second, since after a bandwidth drop, the video bitrate reduction is made proportional to the TCP throughput reduction, PANDA is very agile to bandwidth drops. On the other hand, for both the FESTIVE and the conventional players, the buffer undershoot significantly increases as the scheme becomes more stable. Overall, PANDA has the best tradeoff between stability and responsiveness to bandwidth drop, outperforming the second best conventional player by more than 75% reduction in instability at the same buffer undershoot level. It is worth noting that the conventional player uses exactly the same smoothing and quantization steps as PANDA, which implies that the gain achieved by PANDA is purely due to the improvement in the estimating and
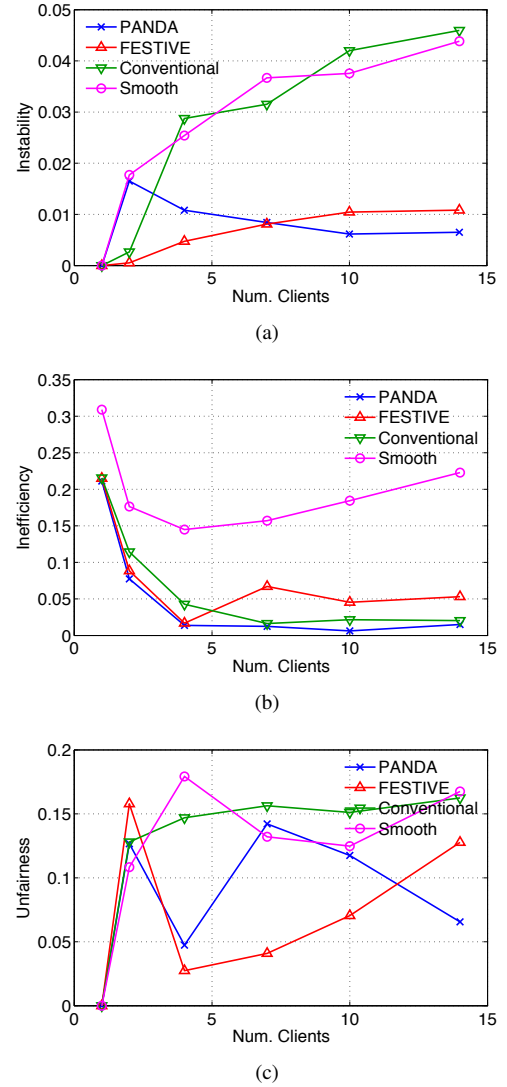
scheduling steps. FESTIVE has the largest buffer undershoot. We believe this is because the design of FESTIVE has mainly concentrated on stability, efficiency and fairness, but ignored responsiveness to bandwidth drops. As we do not have access to the Smooth player's buffer, we do not have its buffer undershoot measure in Fig. 15 (a).

Fig. 15 (b) shows that PANDA has the lowest inefficiency over all as we vary its instability. The probing mechanism ensures that the bandwidth is most efficiently utilized. As the instability increases, the inefficiency also increases moderately. This makes sense intuitively, as when the bitrate fluctuates, the average fetched bitrate also decreases. The efficiency of the conventional algorithm underperforms PANDA, but outperforms FESTIVE. Lastly, the Smooth player has the highest inefficiency.

Lastly, Fig. 15 (c) shows that in terms of fairness, FESTIVE achieves the best performance. This may be due to the randomized scheduling strategy of FESTIVE. PANDA and the conventional players have similar fairness; both of them outperform the Smooth player.
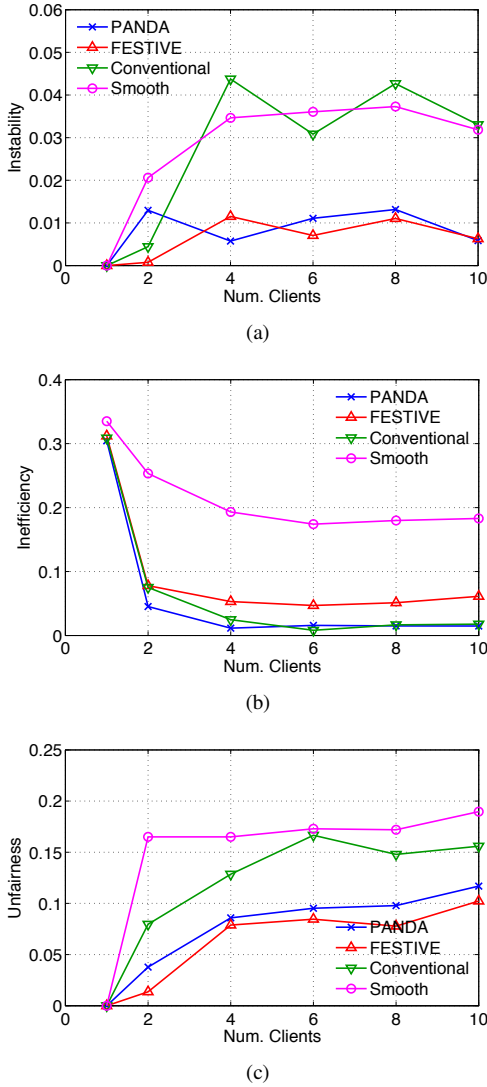
Fig. 17. Instability, inefficiency and unfairness as the number of clients increases. The link bandwidth increases with the number of players, with the bandwidth-player ratio fixed at 1 Mbps/player.

D. Increasing Number of Players

In this section, we focus on the question of how the number of players affects instability, inefficiency and unfairness at steady state. Two scenarios are of interest: i) we increase the number of players while fixing the link bandwidth at 10 Mbps, and ii) we increase the number of players while varying the bandwidth such that the bandwidth/player ratio is fixed at 1 Mbps/player. Fig. 16 and Fig. 17 report results for these two cases, respectively. Each data point is obtained by averaging over 20 runs.

Refer to Fig. 16 (a). In the single-player case, all four schemes are able to maintain their fetched video bitrate at a constant level, resulting in zero instability. As the number of players increases, the instability of the conventional player and the Smooth player both increase quickly in a highly consistent way. We speculate that they have very similar underlying structure. The FESTIVE player is able to maintain its stability at a lower level, due to the strong smoothing effect (smoothing window at 20 samples by default), but the instability still

grows with the number of players, likely due to the bandwidth overestimation effect. The PANDA player exhibits a rather different behavior: at two players it has the highest instability, then the instability starts to drop as the number of players increases. Investigating into the experimental traces reveals that this is related to the specific bitrate levels selected. More importantly, via probing, the PANDA player is immune to the symptoms of the bandwidth overestimation, thus it is able to maintain its stability as the number of clients increases. The case of varying bandwidth in Fig. 17 (a) exhibits behavior fairly consistent with Fig. 16 (a), with PANDA and FESTIVE exhibiting much less instability compared to the Smooth and the conventional players.

Fig. 16 (b) and Fig. 17 (b) for the inefficiency metric both show that PANDA consistently has the best performance as the number of players grow. The conventional player and FESTIVE have similar performance, both outperforming the Smooth player by a great margin. We speculate that the Smooth player has some large bitrate safety margin by design, with the purpose of giving cross traffic more breathing room.

Lastly, let us look at fairness. Refer to Fig. 16 (c). We have found that when the overall bandwidth is fixed, the unfairness measure has high dependence on the specific bitrate levels chosen, especially when the number of players is small. For example, at two players, when the fair-share bandwidth is 5 Mbps, the two PANDA players end up in steady state with 5.3 Mbps and 3.7 Mbps, resulting in a high unfairness score. At three players, when the fair-share bandwidth is 3.3 Mbps, the three PANDA players each end up with 3.7, 3.7 and 2.5 Mbps for a long period of time, resulting in a lower unfairness score. FESTIVE exhibits lowest unfairness overall, which is consistent with the results obtained in Fig. 15 (c). In the varying-bandwidth case in Fig. 17 (c), The unfairness ranking is fairly consistent as the number of players grow: FESTIVE, PANDA, the conventional, and Smooth.

E. Competing Mixed Players

One important question to ask is how PANDA will behave in the presence of different types of other players? If it behaves too conservatively and cannot grab enough bandwidth, then the deployment of PANDA will not be successful. To answer this question, we take the four types of players of interest and have them compete on a 10-Mbps link. For the Smooth player, we test it with both a Microsoft IIS server running on Windows 7, and an Apache HTTP server running on Ubuntu 10.04. A single trace of the fetched bitrates for 500 seconds is shown in Fig. 18. The plot shows that the Smooth player's ability to grab the bandwidth highly depends on the server it streams from. Using the IIS server, which runs on Windows 7 with an aggressive TCP, it is able to fetch video bitrates over 3 Mbps. With the Apache HTTP server, which uses Ubuntu 10.04's conservative TCP, the fetched bitrates are about 1 Mbps. The conventional, PANDA and FESTIVE players all run on the same TCP (Red Hat 6), so their differences are due to their adaptation algorithms. Due to bandwidth overestimation, the conventional player aggressively fetches high bitrates, but the fetched bitrates fluctuate. Both PANDA and FESTIVE are able to maintain a stable fetched bitrate at about the fair-share level of 2 Mbps.
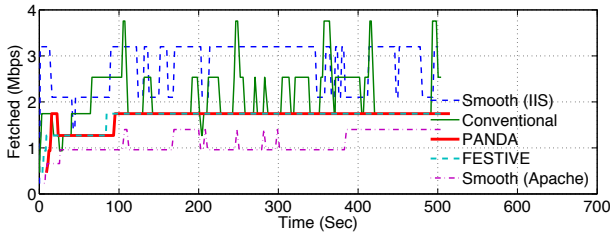
Fig. 18. PANDA, Smooth (w/ IIS), Smooth (w/ Apache), FESTIVE and the conventional players compete at a bottleneck link of 10 Mbps.
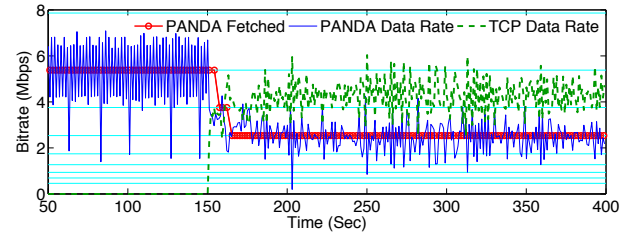

Fig. 19. PANDA competes with a long-running TCP. The link bandwidth is kept at 7 Mbps. The long-running TCP starts at time 150 seconds.

F. Competing with Long-running TCP

In this experiment, we test PANDA's competency against a long-running TCP stream. Long-running TCP is not only ubiquitous in today's Internet, it can also be treated as the aggressiveness upper bound of any HAS clients. Thus, the experimental result also provides a sense of how PANDA performs when competing with the most aggressive HAS client. Fig. 19 shows the trace of PANDA competing with one long-running TCP stream at a bottleneck link of 7 Mbps. After the TCP starts at time 150 seconds, the PANDA stream settles to steady state at fetched bitrate of 2.5 Mbps, while the TCP settles at around 4 Mbps. This result shows that PANDA is fairly competitive against the long-running TCP, or the most aggressive HAS client possible.

There are a few reasons contributing to PANDA's loss of 1 Mbps bitrate compared to the fair-share rate of 3.5 Mbps. First, during the off-intervals, PANDA's TCP congestion window would stop growing, while a long-running TCP's congestion window would keep on growing. Second, both the discrete nature of the video bitrate and the quantization safety margin ϵ (see Table II) would discount the competency of PANDA. Note that these two factors also apply to a conventional HAS client. As we are trying to build a better client algorithm that behaves more properly and results in an improved performance, winning over an aggressive client algorithm is not our primary goal. That said, we believe PANDA still show fair competency against aggressive clients.

G. Summary of Performance Results

- The PANDA player has the best stability-responsiveness tradeoff, outperforming the second best conventional player by 75% reduction in instability. PANDA also has the best bandwidth utilization. In competing with long-running TCP or the most aggressive HAS client, PANDA shows fair competency.
- The FESTIVE player has been tuned to yield high stability, high efficiency and good fairness. However, it underperforms other players in responsiveness to bandwidth drops.
- The conventional player yields good efficiency, but lacks in stability, responsiveness to bandwidth drops and fairness.
- The Smooth player underperforms in efficiency, stability and fairness. When competing against other players, its ability to grab bandwidth is a consequence of the aggressiveness of the underlying TCP stack.

VIII. CONCLUDING REMARKS

This paper identifies an emerging issue for HTTP adaptive streaming, and lays out a solution direction that can effectively address this issue and achieve significant improvements in stability at no cost in responsiveness. Our main contributions in this paper can be summarized as follows:

- We have identified the bandwidth cliff effect as the root cause of the bitrate oscillation/fluctuation phenomenon and revealed the fundamental limitation of the conventional reactive measurement based rate adaptation algorithms.
- We have envisioned a general probe-and-adapt principle to directly address the root cause of the problems, and designed and implemented PANDA, a client-based rate adaptation algorithm, as an embodiment of this principle.
- We have proposed a generic four-step model for an HAS rate adaptation algorithm, based on which we have fairly compared the proposed approach with the conventional approach.

Our proposed PANDA algorithm not only is an effective solution to address the bitrate oscillation problem, but can also serve as a basis for more advanced video quality optimization schemes, such as [16].

As a final remark, we would like to provide a briefly justification for our design choice of admitting the on-off data downloading pattern in our solution. An intriguing question raised by our readers is why we would not simply eliminate the off-intervals by always keeping the data sending on, hence avoiding the bandwidth overestimation problem. We believe that in HAS, the on-off pattern is a natural consequence of the discrete video bitrate. Also, our experiments show that more aggressive clients (e.g., clients without off-intervals by completely filling the pipe) typically yield slower response to network changes. Additionally, an HAS stream without off-intervals would behave like a long-running TCP, which may lead to bad interactions with poorly designed network queue management equipments (e.g., the bufferbloat problem). Last but not least, our solution demonstrates that it is possible to address the rate oscillation problem from a pure application-layer perspective.

APPENDIX

We perform an equilibrium and stability analysis of PANDA. For simplicity, we only analyze the single-client case where the system has an equilibrium point.

Analysis of $\hat{x}$

Assume the measured TCP download throughput $\tilde{x}$ is equal to the link capacity C. Consequently, (8) can be re-written in two scenarios (refer to Fig. 5) as:

$$\frac{\hat{x}[n] - \hat{x}[n-1]}{T[n-1]} =$$
$$\begin{cases} \kappa \cdot w, & \text{if } \hat{x} < C, \\ \kappa \cdot w - \kappa \cdot (\hat{x}[n-1] - C + w), & \text{otherwise.} \end{cases} \quad (18)$$

At equilibrium, setting (18) to zero leads to $\hat{x}_o = \tilde{x}_o = C$. Considering the simplifying assumptions of $\hat{y} = \hat{x}$ (no smoothing) and a quantizer with a margin Δ such that $r = \hat{y} - \Delta$, we need $r_o = \hat{x}_o - \Delta \leq C$ at equilibrium. Therefore, the quantization margin of the quantizer needs to satisfy

$$\Delta \geq \hat{x}_o - C = 0,$$

so that the system stays on the multiplicative-decrease side of the equation at steady state.

Note that close to equilibrium, the intervals between consecutive segment downloads match the segment playout duration $T[n-1] = \tau$, therefore, calculation of the estimated bandwidth $\hat{x}$ follows the simple form of a difference equation:

$$\hat{x}[n] = a \cdot \hat{x}[n-1] + b.$$

The two constants are: $a = 1 - \kappa \cdot \tau$ and $b = \kappa \cdot (C + w) \cdot \tau$. The sequence converges if and only if $|a| < 1$. Hence, we need: $-1 < 1 - \kappa \cdot \tau < 1$, leading to the stability criterion:

$$0 < k < \frac{2}{\tau}.$$

Analysis of $\hat{T}$ and B

Assume no smoothing $\hat{x} = \hat{y}$. During transient states, when $T = \hat{T} > \tilde{T}$, update of $\hat{T}$ pushes the playout buffer size in the right direction: $\hat{T} < \frac{r[n] \cdot \tau}{\hat{x}[n]}$ leads to a steady growth in buffer size when $B < B_{\min}$. At steady state, $\hat{T}_o = \tau = \hat{T}_o > \tilde{T}_o$. It can be derived from (11) that:

$$B_o = B_{\min} + \left(1 - \frac{r_o}{\hat{x}_o}\right) \cdot \frac{\tau}{\beta}$$

where B_o is playout buffer at equilibrium.

REFERENCES

[1] History of Move Networks. Available online: http://www.movenetworks.com/history.html.

[2] S. Akhshabi, S. Narayanaswamy, Ali C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptive video players over HTTP. *Signal Processing: Image Communication*, 27:271–287, 2012.

[3] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen. What happens when HTTP adaptive streaming players compete for bandwidth. In *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'12)*, Toronto, Ontario, Canada, 2012.

[4] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'13)*, 2013.

[5] Alex Zambelli. IIS Smooth Streaming Technical Overview. http://tinyurl.com/smoothstreaming.

[6] Luca De Cicco, Saverio Mascolo, and Vittorio Palmisano. Feedback control for adaptive live video streaming. In *Proc. ACM Multimedia Systems Conference (MMSys'11)*, pages 145–156, San Jose, CA, USA, February 2011.

[7] Cisco White Paper. Cisco visual networking index - forecast and methodology, 2011-2016. http://tinyurl.com/ciscovni2011to2016.

[8] Luca De Cicco and Saverio Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *Proc. 6th International Conference on HCI in Work and Learning, Life and Leisure: Workgroup Human-computer Interaction and Usability Engineering*, USAB'10, pages 447–464, Berlin, Heidelberg, 2010. Springer-Verlag.

[9] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 362–373, New York, NY, USA, 2011. ACM.

[10] Remi Houdaille and Stephane Gouache. Shaping HTTP adaptive streams for a better user experience. In *Proc. 3rd Multimedia Systems Conference*, pages 1–9, 2012.

[11] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proc. 2012 ACM conference on Internet measurement*, 2012.

[12] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[13] Dmitri Jarnikov and Tanir Ozcelebi. Client Intelligence for Adaptive Streaming Solutions. *EURASIP J. Signal Processing: Image Communication, Special Issue on Advances in IPTV Technologies*, 26(7):378–389, August 2011.

[14] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE. In *Proc. 8th international conference on Emerging networking experiments and technologies (CoNEXT)*, 2012.

[15] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):237–252, 1998.

[16] Zhi Li, Ali C. Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. Streaming video over HTTP with consistent quality. In *Proc. ACM Multimedia Systems Conference*, 2014.

[17] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. Rate Adaptation for Adaptiv HTTP streaming. In *Proc. ACM Multimedia Systems Conference (MMSys'11)*, pages 169–174, San Jose, CA, USA, February 2011.

[18] Chenghao Liu, Imed Bouazizi, Miska M. Hannuksela, and Moncef Gabbouj. Rate adaptation for dynamic adaptive streaming over HTTP in content distribution network. *Signal Processing: Image Communication*, 27(4):288–311, April 2012.

[19] Konstantin Miller, Emanuele Quacchio, Gianluca Gennari, and Adam Wolisz. Adaptation algorithm for adaptive streaming over HTTP. In *Proc. 2012 IEEE 19th International Packet Video Workshop*, pages 173–178, 2012.

[20] R.K.P. Mok, E.W.W. Chan, X. Luo, and R.K.C. Chang. Inferring the QoE of HTTP video streaming from user-viewing activities. In *Proc. SIGCOMM W-MUST*, 2011.

[21] Guibin Tian and Yong Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proc. 8th international conference on Emerging networking experiments and technologies (CoNEXT)*, pages 109–120, 2012.

[22] Chao Zhou, Xinggong Zhang, Longshe Huo, and Zongming Guo. A control-theoretic approach to rate adaptation for dynamic HTTP streaming. In *Proc. Visual Communications and Image Processing (VCIP)*, pages 1–6, 2012.

Zhi Li is with the Video and Collaboration Group at Cisco Systems. His current research interests focus on improving video streaming experience for consumers through system design, optimization and control algorithms. He has broad interests in applying mathematical tools and analytical skills in solving real-world engineering problems.

Zhi received his B.Eng. and M.Eng. degrees in Electrical Engineering, both from the National University of Singapore, Singapore, in 2005 and 2007, respectively, and a Ph.D. degree from Stanford University, California, USA in 2012. His doctoral work focuses on source and channel coding methods in multimedia networking problems. He was a recipient of the Best Student Paper Award at IEEE ICME 2007 for his work on cryptographic watermarking, and a co-recipient of Multimedia Communications Best Paper Award from IEEE Communications Society in 2008 for a work on multimedia authentication.
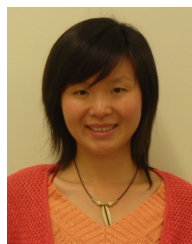
Xiaoqing Zhu is a Technical Leader at the Enterprise Networking Lab at Cisco Systems Inc. She received the B.Eng. degree in Electronics Engineering from Tsinghua University, Beijing, China. She earned both the M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, California, USA. Prior to joining Cisco, Dr. Zhu interned at IBM Almaden Research Center in 2003, and at Sharp Labs of America in 2006. She received the best student paper award in ACM Multimedia 2007.

Dr. Zhu's research interests span across multimedia applications, networking, and wireless communications. She has served as reviewer, TPC member, and special session organizer for various journals, magazines, conferences and workshops. She has contributed as guest editor to several special issues in IEEE Technical Committee on Multimedia Communications (MMTC) E-Letter, IEEE Journal on Selected Areas in Communications, and IEEE Transactions on Multimedia.

Joshua Gahm is a Principal Engineer in the Service Provider Video Technology Group at Cisco. During 15 years at Cisco and more than 25 years overall in the areas of IP networking and video, Joshua has led numerous product development and applied research teams, spanning areas including IP-TV, DOCSIS, HTTP Adaptive Bitrate Streaming, and MPLS. Joshua's interests include network QoS, transport protocols, IP video delivery systems, caching algorithms, and real-time software design. Joshua holds an A.B. in Mathematics from Harvard University.

Rong Pan has been active in the networking field for more than ten years while working at Cisco Systems, Stanford University, Bell Laboratories, and startups. Currently she is a Distinguished Engineer at Cisco where she heads a team in ENG CTO office. She is an author for 30+ technical papers and an inventor of 30+ patents. Her work and innovations have been recognized and adopted widely in the industry. Currently, she is working on the buffer bloat problem in the Internet and leading Ciscos effort at IETF and DOCSIS on this topic. Rong has won a best paper award and served as program committee members at IEEE conferences, and she will serve as the technical co-chair at IEEE High Performance Switching and Routing Conference 2014.
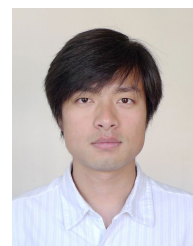
Hao Hu received the B.S. degree from Nankai University and the M.S. degree from Tianjin University in 2005 and 2007 respectively, and the Ph.D degree from Polytechnic Institute of New York University in Jan. 2012.

He is currently with ENG Labs, Cisco Systems, San Jose, CA. He interned in the Corporate Research, Thomson Inc., NJ in 2008 and Cisco Systems, CA in 2011. His research interests include video QoE, video streaming and adaptation, and distributed systems.

Ali C. Begen is with the Video and Content Platforms Research and Advanced Development Group at Cisco. His interests include networked entertainment, Internet multimedia, transport protocols and content delivery. Ali is currently working on architectures for next-generation video transport and distribution over IP networks, and he is an active contributor in the IETF and MPEG in these areas.

Ali holds a Ph.D. degree in electrical and computer engineering from Georgia Tech. He received the Best Student-paper Award at IEEE ICIP 2003, the Most-cited Paper Award from Elsevier Signal Processing: Image Communication in 2008, and the Best-paper Award at Packet Video Workshop 2012. Ali has been an editor for the Consumer Communications and Networking series in the IEEE Communications Magazine since 2011 and an associate editor for the IEEE Transactions on Multimedia since 2013. He is a senior member of the IEEE and a senior member of the ACM. Further information on Alis projects, publications and presentations can be found at http://ali.begen.net.
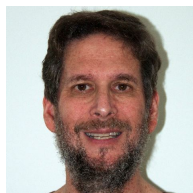
David Oran is a Fellow at Cisco Systems. His technical interests lie in the areas of Quality of Service, Internet multimedia, routing, and security. He was part of the original team that started Cisco's Voice-over-IP business in 1996 and helped grow it into a multi-billion dollar revenue stream. He is currently working on adaptive bitrate video over IP networks, and advanced research on Information Centric Networks.

Prior to joining Cisco, Mr. Oran worked in the network architecture group at Digital Equipment, where he designed routing algorithms and a distributed directory system. Mr. Oran has led a number of industry standards efforts. He was a member of the Internet Architecture Board, co-chair of the Speech Services working group, and served a term as area director for Routing in the IETF. He was on the board of the SIP Forum from its inception through 2008. He also serves on the technical advisory boards of a number of venture-backed firms in the networking and telecommunication sectors.

Mr. Oran has a B.A. in English from Haverford College.