Post-Streaming Rate Analysis—A New Approach to Mobile Video Streaming with Predictable Performance

Yan Liu and Jack Y. B. Lee, *Senior Member, IEEE*

Abstract—Fueled by the growth of 3G/4G mobile networks, mobile video streaming has become one of the main applications in the mobile Internet. Due to mobile networks' inherent bandwidth fluctuations, the industry as well as researchers have developed many adaptive streaming algorithms to compensate for such fluctuations to improve streaming performance. Given the wide range of network settings, it is not surprising that existing algorithms can and do perform differently across different network and system conditions. This work breaks away from the conventional one-size-fits-all approach to designing adaptive streaming systems by developing a new framework called PSRA where past throughput trace data - captured as a by-product of streaming, are analyzed to construct a statistical model to automatically tune the adaptation algorithm for future streaming sessions according to the underlying network and system configurations. Compared to existing approaches, the PSRA-optimized streaming algorithm can achieve predictable, consistent, and controllable streaming performance across a wide-range of network and system configurations. Moreover, PSRA offers to service provider a new tool to precisely control the tradeoff between video quality and streaming performance. Results from extensive trace-driven simulations as well as experiments verified PSRA's performance under real-world mobile network and system configurations.

Index Terms—Dynamic adaptive streaming over HTTP, mobile network, performance guarantee, video streaming

1 Introduction

Mobile video streaming has grown tremendously in recent years and it is now one of the main applications in the mobile Internet [1]. Unlike fixed networks, bandwidth availability in mobile data networks are known to exhibit far more fluctuations, thereby posing significant challenges to the provisioning of bandwidth-sensitive streaming services. In response, the industry developed adaptive streaming systems to dynamically adjust the video bitrate to compensate for the network's bandwidth variations. Most notably is Apple's HTTP Live Streaming (HLS) protocol [2] which has been widely deployed not only in Apple devices, but are also supported by recent Android devices as well. Similar solutions have also been developed by Microsoft [3], Adobe [4], Netflix [5], and the research community [6], [7], [8], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28].

Nevertheless, while Apple's HLS protocol was successful in reducing video interruptions, also known as playback rebuffering [7], [8], it is done at the expense of reduced video quality-previous works have shown that the protocol was designed to be conservative in its video bitrate selections [6]. Moreover, even with such conservative approach, playback rebuffering is still unavoidable from time to time—ultimately the actual network conditions still dictate the resultant streaming performance.

• *The authors are with the Department of Information Engineering, Chinese University of Hong Kong, Shatin, NT, Hong Kong SAR. E-mail: liuyan.adams@gmail.com, jacklee@computer.org.*

Manuscript received 19 May 2016; revised 27 Mar. 2017; accepted 9 Apr. 2017. Date of publication 14 Apr. 2017; date of current version 1 Nov. 2017. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2017.2694418

This last point is the crux of the challenge—current adaptive streaming algorithms were designed to function in mobile networks of *all* sizes and shapes, ranging from 3G networks with a few Mbps peak bandwidth all the way to LTE networks with 100+ Mbps peak bandwidth; from lightly loaded, well-covered mobile cells to crowded/congested cells; and so on. In other words, current designs were all intended to be *one-size-fits-all*.

While this approach to designing adaptive streaming algorithms can still improve performance over non-adaptive streaming, we argue that it has two fundamental limitations. First, most, if not all, adaptive streaming algorithms have internal parameters which require tuning to achieve the desired performance tradeoff (e.g., streaming performance versus video quality). It is easy to see that a *single* set of tuned parameters is unlikely to be optimal for the wide range of networks across the world.

Second, it follows that an adaptive streaming algorithm will likely perform *differently* in different networks/systems. Indeed this is the norm today as it is almost an accepted notion that streaming performance can and do *vary* with the network conditions. In other words, streaming the same video from the same mobile operator even in the same location could still result in difference performances depending on the *specific* network conditions at the time.

This work investigates a radically-different approach to the problem of adaptive video streaming. In addition to developing an adaptive streaming algorithm to compensate for real-time bandwidth fluctuations, we propose to *tune* the adaptive streaming algorithm *itself* for the specific network it operates in to exploit the inherent characteristics of the underlying network. This breaks way from the one-size-fits-all approach and opens a new paradigm to network-optimized adaptive

video streaming. We call this the Post-Streaming Rate Analysis (PSRA) framework.

Through extensive trace-driven simulations and also experiments conducted in production mobile networks, we show that PSRA can overcome the two aforementioned limitations. First, by exploiting knowledge of network characteristics, PSRA-tuned adaptive streaming outperformed existing protocols across a wide range of network conditions. Second, PSRA enables the service provider to *explicitly* control the streaming performance, e.g., by specifying the target rebuffering probability *directly*, and it will then incorporate system-wide characteristics including the underlying network's bandwidth properties, the video bitrate choices available, video duration, and so on, to *automatically* tune the adaptive streaming algorithm to achieve the target rebuffering probability *consistently*.

The rest of this paper is organized as follows. Section 2 reviews some previous related works; Section 3 investigates performance variations of existing adaptation algorithms; Section 4 presents the PSRA framework; Section 5 evaluates PSRA using trace-driven simulations; Section 6 tackles the challenge due to ultra-long streaming sessions; Section 7 reports experimental results from our prototype implementation of PSRA; and Section 8 summarizes the study and outlines some future work.

2 Background and Related Work

In recent years much work has been done in the area of mobile video streaming. Beginning with non-adaptive streaming [9], [10], [11] the industry soon realized that the inherent bandwidth fluctuations in mobile networks posed significant challenges to bandwidth-sensitive streaming services. This led to intense research in recent years on the design of adaptive streaming systems where the video bitrate is dynamically adjusted to compensate for network bandwidth variations.

A detailed review of existing adaptive streaming algorithms is beyond the scope of this work. Interested readers are referred to a recent survey by Seufert et al. [12]. In the following we briefly summarize recent works in the literature and developments in the industry.

2.1 Literature Review

We can classify adaptive streaming algorithms into three categories: bandwidth-based, buffer-based, and hybrid bandwidth-buffer-based algorithms.

In bandwidth-based algorithms, video bitrate selection is mainly driven by bandwidth *measured* at the application layer (i.e., throughput) or by bandwidth estimated from lower-layer methods. For example, Liu et al. [13] proposed an adaptive streaming algorithm (henceforth called LBG) using a smoothed HTTP throughput measured based on the segment fetch time to detect available bandwidth and then employ step-wise increase/aggressive decrease method for bitrate switches to avoid video rebuffering. Jiang et al. [14] (henceforth called FESTIVE) applied harmonic mean to past segment downloading throughput to reduce the error introduced by outliers. There are many other examples (e.g., [15], [16]). While the exact methods vary the general principle is to predict future bandwidth availability from past throughput measurements.

Buffer-based algorithms make use of client video buffer occupancy to drive the adaptation of video bitrate. For example, De Cicco et al. [17] proposed an algorithm to track a specified target buffer occupancy using a feedback controller to prevent playback rebuffering. Huang et al. [18] proposed an algorithm (henceforth called BBA2) to adapt video bitrate using only client buffer occupancy, arguing that sudden bandwidth changes can cause throughput-based algorithms to overestimate/underestimate network capacity [19], resulting in unnecessary video rebuffering.

Naturally, bandwidth and buffer information can also be combined in bitrate adaptation. For example, besides throughput estimate, Tian et al. [20] employed feedback control using buffer occupancy as signal to adjust video bitrate selection, aiming to achieve agile and smooth video adaptation. Yin et al. [21] (henceforth called FastMPC) used both throughput estimate and buffer occupancy measurement in their proposed predictive control algorithm to adapt video bitrate to maximize users' quality of experience. Spiteri et al. [22] (henceforth called BOLA-U) employed Lyapunov optimization technique to design adaptive video streaming algorithm based on measured throughput and buffer occupancy.

As opposed to passive throughput measurements, Mok et al. [23] employed active probing using packet trains to estimate the available bandwidth and use buffer occupancy as a cushion to avoid sharp video quality drop. Xie et al. [24] went further to develop an adaptive video streaming framework that is integrated with the LTE infrastructure so that information such as PHY-layer resource allocation can be used in bandwidth estimation. In addition to using recently measured bandwidth, Hao et al. [25], Riiser et al. [26], Yao et al. [27] and Bokani et al. [28] investigated the use of location-based/trip-based bandwidth information for adaptive streaming under mobility (e.g., in a car).

2.2 Industry Developments

Adaptive video streaming has been implemented widely by several companies, including Apple's QuickTime Player using HTTP Live Streaming (HLS) [2], Adobe's OSMF player [16] using HTTP Dynamic Streaming [4], Microsoft's Smooth Streaming [13], Akamai http-based adaptive video streaming [29], and Netflix's player [5]. Moreover, the MPEG committee has recently ratified the Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [30], [31] standard to enable interoperability between encoders, servers, and players from different vendors.

Apart from adaptation algorithms implemented in Adobe's OSMF player [32] and in the Android OS *Stagefright* [33] (henceforth called Stagefright), which are available as open source projects, the adaptation algorithms implemented by other commercial players are proprietary. Interested readers are referred to the studies by Akhshabi et al. [34], Riiser et al. [6], and De Cicco et al. [17], [35] for evaluation of commercial implementations.

2.3 Discussions

Existing measurement studies by Akhshabi et al. [34], Riiser et al. [6], and De Cicco et al. [17], [35] and also our own investigations all show that existing adaptation algorithms have widely different *design tradeoffs*, resulting in very different streaming performance and video quality even under the *same* network condition. Moreover, *none* of the existing streaming algorithms were designed to maintain a *given* target streaming performance over different network conditions. Consequently, as will be demonstrated in the next section,

existing algorithms can and do perform very differently under different network conditions, different video duration, or even different video bitrate compositions. These are a direct consequence of the one-size-fits-all design approach.

By contrast, the proposed PSRA framework is designed to automatically *tune* itself to the underlying network and system conditions so that it not only can achieve *predictable* and *consistent* streaming performance over a wide range of system and network parameters, but it also enables the service provider to explicitly *control* the tradeoff between video quality and streaming performance.

An early version of the PSRA framework was reported in [35], [36], [37]. This study extends the previous works in three significant aspects. First, we extended the original PSRA framework which was directly coupled with the adaptation algorithm using a single performance metric (i.e., rebuffering probability), to a generalized version that can be used to optimize different rate-adaptation algorithms according to a wide range of performance metrics. In particular, we developed two new generalized metrics called rebuffering ratio and rebuffering numbers, both of which subsumes rebuffering probability (as used in our previous studies) as a special case.

Second, this study investigated the challenge in streaming ultra-long videos (e.g., 2 hours) with predictable and consistent performance. The newly generated results revealed that performance predictability could degrade in streaming ultra-long videos. This problem was addressed using a feedback mechanism presented in Section 6.

Third, in this study we set out to conduct a systematic and thorough evaluation of PSRA's performance and compare it to leading streaming algorithms. To this end we first enhanced the simulator's fidelity by incorporating random user arrivals, variable video duration, trace data concatenation, etc., to capture real-world operational scenarios. Next we implemented 7 existing streaming algorithms to compare to PSRA quantitatively. Last but not least, we reported experimental results from an implementation of the PSRA framework streaming real video data using Apple's HLS protocol to smartphones over a production mobile network. The experimental results are significant in that: (a) it verified PSRA's performance predictability in a real implementation running in a production mobile network; and (b) it demonstrated PSRA's feasibility for use in today's adaptive video streaming platforms.

3 Mobile Video Streaming Re-Examined

In this section we demonstrate the limitations of the one-size-fits-all approach to designing adaptive streaming algorithms. We employed extensive trace-driven simulations with real-world mobile network HTTP/TCP throughput trace data to evaluate the performance of 7 state-of-the-art adaptive streaming algorithms, two from the industry: OSMF [32] and *Stagefright* [33]; and 5 from the research community: LBG [13], FESTIVE [14], BBA2 [18], FastMPC [21], and BOLA-U [22]. Although we were not able to re-implement Apple's proprietary adaptation algorithm in iOS, we were able to compare to it experimentally via our prototype implementation in Section 7.

All 7 streaming algorithms employed HTTP-based video streaming where a video is encoded into multiple bitrate versions and then delivered in fixed-duration segments. The encoder generates a playlist listing the available bitrates and their URLs for all video segments. A client fetches the playlist and then begins the streaming session by sequentially requesting video segments of available bitrates according to its rate-adaptation algorithm.

TABLE 1
Bitrate Profiles Used in Simulations

Source	Video Encoding Bitrates (kbps)
Apple [42] (default)	200, 400, 600, 1,200, 3,500, 5,000, 6,500, 8,500
Adobe [43]	300, 600, 900, 1,200, 1,500, 2,000, 2,500, 3,000, 3,500, 4,000, 5,000, 6,000, 8,000
Stefan et al. [44]	50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 900, 1,200, 1,500, 2,000, 2,500, 3,000, 4,000, 5,000, 6,000, 8,000

3.1 Experiment Setup

Mobile network bandwidth characteristics can vary widely in practice and so far no known existing model can accurately capture their characteristics [38], [39]. Therefore in this work we primarily employed *trace-driven* simulations as a mean to evaluate different streaming algorithms in a realistic, consistent, and repeatable manner.

We collaborated with a mobile operator to setup a platform in a production 3G/HSPA network to collect TCP throughput trace data. The server host runs Linux with the Apache httpd [40] serving video data over TCP CUBIC [41]. The client is a notebook computer running Microsoft Windows 7, equipped with a 3G/HSPA USB modem for connecting to the mobile network. We developed a custom software to automatically initiate long TCP sessions to measure the *actual throughput achievable* over the mobile network. The full packet traces were captured at the client side. A total of 30 weeks' trace data were captured in three locations ($\sim$3 months for each location) for the equivalent of around 60,000 5-min streaming sessions. It is worth noting that these captured throughput trace data were subject to all kinds of interferences such as radio signal quality variations, competing users sharing the same mobile cell, radio interferences, time-of-day and day-of-the-week traffic variations, and so on, thereby enabling realistic evaluation of streaming algorithms under real-world network conditions.

The trace-driven simulator was developed in C. It replicated the throughput between the server and the client using the captured throughput trace data and simulated HTTP-based video streaming with a prefetch-buffering of 20 seconds video after which playback begins (21 seconds for BOLA-U as it employed 3-second segment duration). In case the player's buffer becomes empty then playback will be suspended, i.e., rebuffering, until one complete video segment is received. Unless stated otherwise video duration was 5 minutes and the available video bitrates followed the Apple profile listed in Table 1.

3.2 Performance in Real-World Mobile Networks

A key observation from the experiments is that a streaming algorithm's performance, despite being adaptive, can and do vary *substantially* across different networks/system parameters. Table 2 lists the rebuffering probability—defined as the proportion of streaming sessions which suffered one or more rebuffering events, for the 7 streaming algorithms in three network locations. Not surprisingly, the performance varied significantly across streaming algorithms, e.g., in location #3 Stagefright achieved rebuffering probability of zero versus LBG's 0.326.

TABLE 2
Rebuffering Probabilities over Three Locations

Loc. ID	#1	#2	#3
LBG	0.068	0.244	0.326
OSMF	0.077	0.248	0.001
Stagefright	0.003	0.012	0.000
FESTIVE	0.004	0.005	0.000
BBA2	0.003	0.002	0.000
BOLA-U	0.064	0.250	0.000
FastMPC	0.390	0.696	0.051

TABLE 3
Average Bitrate (Mbps) over Three Locations

Loc. ID	#1	#2	#3
LBG	3.26	1.90	2.78
OSMF	4.46	2.32	2.38
Stagefright	2.49	1.19	1.28
FESTIVE	3.00	1.54	1.79
BBA2	2.24	1.30	1.70
BOLA-U	4.55	2.44	3.13
FastMPC	4.85	2.62	3.30

TABLE 4
Rebuffering Probabilities for Different Video Durations

Duration (sec)	300	1200	3600	7200
LBG	0.068	0.128	0.175	0.217
OSMF	0.077	0.099	0.113	0.122
Stagefright	0.002	0.014	0.033	0.038
FESTIVE	0.004	0.014	0.037	0.068
BBA2	0.003	0.006	0.012	0.023
BOLA-U	0.064	0.081	0.127	0.180
FastMPC	0.390	0.733	0.875	0.938

TABLE 5
Rebuffering Probabilities Using Different Bitrate Profiles

Bitrate Profile	Apple [38]	Adobe [39]	Stefan [40]
LBG	0.068	0.016	0.001
OSMF	0.077	0.093	0.066
Stagefright	0.002	0.010	0.001
FESTIVE	0.004	0.006	0.003
BBA2	0.003	0.002	0.006
BOLA-U	0.064	0.069	0.048
FastMPC	0.390	0.485	0.469

This does not necessarily imply LBG is inferior to Stagefright however. We list in Table 3 the average video bitrate *delivered*. Clearly Stagefright's perfect streaming performance is achieved at the expense of low video bitrate (e.g., 1.28 Mbps versus 2.78 Mbps for LBG in location #3). Similar observations can also be drawn for the other algorithms. This result demonstrates that different streaming algorithms *implicitly* achieve a different set of tradeoffs between video quality (as measured by delivered video bitrate) and streaming performance (as measured by rebuffering probability).

More importantly, even the same algorithm's performance can also vary widely across different locations. For example, FastMPC's rebuffering probability varied from 0.051 in location #3 to 0.696 in location #2. This shows that despite the streaming algorithm's efforts in adapting to the changing network conditions, their streaming performance can still vary significantly across different locations. Interested readers are referred to Appendix I, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2017.2694418 for more details.

In addition to network locations, our experiments also revealed some unexpected dependencies. First, we investigated the impact of video duration on streaming performance—an often neglected system parameter in existing work. The results in Table 4 demonstrate one common property for all five protocols—the rebuffering probability increases with longer video duration. This is because longer video duration generally offers more opportunities for playback rebuffering. This reveals an inherent limitation of existing algorithms, i.e., their streaming performances can deteriorate substantially for longer videos. For example, compared to 300-second video duration the rebuffering probability increased from 0.068 to 0.217 for LBG and from 0.064 to 0.18 for BOLA-U when applied to 2-hour videos.

Second, we investigate in Table 5 the impact of available bitrate choices – another often neglected system parameter. In addition to the bitrate profile after Apple [42] we tested two other bitrate profiles after Adobe [43] and Stefan *et al*. [44]. Despite the fact that all three profiles offer a wide range of bitrates, two of the algorithms, namely LBG and Stagefright, exhibited significant (one order of magnitude) variations across the three profiles. Thus even the available bitrate choices can have significant impact to streaming performance. Without an understanding of the interactions between bitrate choices and the rate adaption algorithm, it will be difficult, if not impossible, to optimize one for the other.

The results in the previous section clearly demonstrated that the performance of existing adaptive streaming algorithms can and do vary widely in real-world network environments. This presents a significant challenge to the service provider as it is simply not possible to *predict*, let alone *control*, the streaming performance such that a *consistent* user experience can be delivered. This may be acceptable to users streaming user-generated videos but will become a significant issue for the emerging premium higher-quality paid mobile streaming services. We present the PSRA framework in the next section to tackle this challenge.

4 Post-Streaming Rate Analysis (PSRA)

PSRA is built upon two principles. First, by simulating and analyzing *past* streaming sessions PSRA constructs a statistical model to quantify the relation between *streaming parameters* and *streaming performance*. This is known as the *analysis phase* and is to be performed periodically (e.g., daily) using a sliding window of past trace data (e.g., 28 days). Second, equipped with the statistical model PSRA can then apply it to the selection of streaming parameters for *new* streaming sessions in the *prediction phase* to achieve the desired target streaming performance.

In the following we first present the system model, using an intuitive rate-adaptation algorithm to illustrate the relation between streaming parameters and streaming performance. Next we present the operations and deployment options of PSRA in Section 4.2 and 4.3, and discuss its salient properties in Section 4.4.

4.1 System Model

Although PSRA makes use of throughput trace data in its analysis phase, it does not require separate throughput

measurements. For example, in our prototype implementation (c.f. Section 7) we added a simple module to the HTTP streaming server to record the amount of video data delivered over fixed intervals of Δ seconds (e.g., $\Delta = 100$ ms). In this way the TCP throughput trace data can then be captured as a by-product of actual streaming sessions, thereby eliminating the need to carry out separate measurements. Note that throughput capture can also be performed at the client (i.e., playback software) or even independently via a network monitoring device. The exact implementation is likely to be dictated by practical considerations (c.f. Section 4.3).

We first introduce an intuitive rate-adaptation algorithm to illustrate the PSRA framework. Let $C_{i,j}$ denotes the average throughput of interval j in session i; and $t = 0$ be the time streaming begins. Then the cumulative amount of data delivered to the client (ignoring packets inflight, etc.) at time t, denoted by $A_i(t)$, is given by

$$A_i(t) = \sum_{j=0}^{\lfloor t/\Delta \rfloor} C_{i,j}\Delta \tag{1}$$

At the start of a video session there is no recent throughput information available and so the rate-adaptation algorithm selects a pre-configured video bitrate, denoted by V, for the first M segments during prefetch, i.e.,

$$\hat{r}_{i,k} = V,\ k = 0, 1, .., M-1 \tag{2}$$

where $\hat{r}_{i,k}$ denotes the selected video bitrate for the k-th segment in session i.

After delivering the first M segments, it can then estimate the future throughput in transferring segment $k\,(k \geq M)$, denoted by $S^*_{i,k}$, from the arithmetic mean [38] of the throughput in transferring the past M video segments:

$$S^*_{i,k} = \frac{1}{M}\sum_{m=0}^{M-1} S_{i,k-m},\ \text{where } k \geq M \tag{3}$$

where $S_{i,k-m}$ is the actual average throughput in transferring video segment $k - m$ in session i which can be measured directly by the client or the server in receiving/sending the video data.

Together with information on the current client buffer occupancy, denoted by $D(t_k)$—the amount of video data (measured in playback time) buffered at the client at the time of requesting segment k, i.e., t_k, it can then select the video bitrate for the future segment k, denoted by $r_{i,k}$, from

$$r_{i,k} = \gamma S^*_{i,k} \cdot \frac{D(t_k) + U}{U} \tag{4}$$

where U is the video segment duration; and γ is a parameter controlling the tradeoff between video quality and streaming performance. Increasing γ raises video bitrate at the expense of higher playback rebuffering probability, and vice versa. Intuitively, if $\gamma = 1$ and the predicted throughput is exact then (4) will choose a video bitrate such that the buffered video data are all consumed by the time the new segment is downloaded, thereby maximizing video quality without impacting streaming performance. In practice there will likely be errors in throughput prediction and thus PSRA through analyzing past trace data will select a smaller $\gamma < 1$ to achieve the target rebuffering probability.

Note that PSRA can also be applied to other adaptation algorithms. For example, instead of applying a linear relation between client buffer occupancy and selected bitrate, we can also apply a second-order mapping function as follows:

$$D^*(t_k) = (D(t_k))^2/\omega \tag{5}$$

where $\boldsymbol{\omega}$ is a configurable parameter to control the sensitivity of bitrate selection to the client buffer occupancy.

The intuition is that if the client buffer occupancy is low then it is prone to rebuffering. Hence the adaptation algorithm should be more conservative in bitrate selection. By contrast, if client buffer occupancy is large then it is relatively safe for the adaptation algorithm to be more aggressive in its bitrate selection.

On the other hand, PSRA can also support additional constraints on the rate adaptation algorithm. For example, we can easily extend PSRA to support *explicit* control of bitrate switching frequency to tradeoff between video quality and streaming performance. Intuitively, more frequent bitrate switches allows the streaming algorithm to more easily adapt to the fluctuating bandwidth but at the expense of degraded video quality due to more frequent video quality changes and vice versa.

We introduce a new parameter called *bitrate switching period*, denoted by $\boldsymbol{\tau}$, to control the *minimum* time between bitrate changes in PSRA where $\boldsymbol{\tau}$ is chosen as positive integer multiples of the segment duration U. The bitrate switching period thus sets the upper bound for the bitrate switching frequency. The extended rate adaptation algorithm becomes:

$$r_{i,k} = \begin{cases} \gamma S^*_{i,k}\frac{D(t_k)+\tau}{\tau},\ if(k-M) \bmod m = 0\ \&\ k \geq M \\ r_{i,k-1}, \text{otherwise}, \end{cases} \tag{6}$$

where the modulus operator accounts for bitrate adjustments once every m segments (except for the first M segments during prefetch).

The previous computed bitrate is from a continuous bitrate spectrum. As there are only a finite number of discrete bitrate versions available at the server, we need to map the computed bitrate to one of the available bitrates. The mapped video bitrate selection for segment k, is computed from

$$\hat{r}_{i,k} = v_x, \tag{7}$$

where

$$x = \max\{n | v_n \leq r_{i,k}\}, \tag{8}$$

i.e., mapping to the highest available video bitrate not exceeding $r_{i,k}$. The sequence of bitrate selections $\{\hat{r}_{i,k}\}$ thus captured the behavior of the rate-adaptation algorithm.

Next we model the video data consumption process. As the client commences playback after receiving M video segments at say, time t_0, the cumulative amount of video data consumed by playback at time t ($t > t_0$), denoted by $B_i(t)$, is given by

$$B_i(t) = \sum_{k=0}^{\lfloor (t-t_0)/U \rfloor} \hat{r}_{i,k} \cdot U. \tag{9}$$

Note that we assume the video player decode a video segment only *after* it has been completely received. Finally, playback will be continuous without rebuffering if the client never runs into buffer underflow, i.e.,

$$A_i(t) \geq B_i(t), \forall t \geq 0. \tag{10}$$

Otherwise the client will suspend playback until the next video segment is completely received. Let $u_{i,j}$ and $v_{i,j}$ be the duration and occurring time for rebuffering event $j, j = 0,$

$1, \ldots, n_i$, in session i. Then we can extend the consumption equation in (9) to incorporate rebuffering events from

$$B_i^*(t) = B_i\left(t - \sum_{\forall j | v_{i,j} < t} u'_{i,j}(t)\right) \quad \text{where } u'_{i,j}(t) = \begin{cases} u_{i,j}, \text{if } v_{i,j} + u_{i,j} < t \\ v_{i,j} + u_{i,j} - t, \text{otherwise,} \end{cases} \tag{11}$$

which accounts for playback time deferred by rebuffering events.

Now we can measure streaming performance in terms of rebuffering ratio—defined as the ratio of total rebuffering time to video duration T, from

$$T^{-1} \sum_{\forall j} u_{i,j}. \tag{12}$$

Given a target rebuffering ratio Φ, PSRA will then determine the maximum value of γ, denoted by $\gamma_i^{\max}$, that can be used from

$$\gamma_i^{\max} = \max\left\{\gamma \middle| T^{-1} \sum_{\forall j} u_{i,j} \leq \Phi\right\}, \tag{13}$$

which maximizes video quality subject to the rebuffering ratio constraint. Note that (13) generalized the rebuffering probability metric used in our previous studies [15], [36], [37] where it reduces to rebuffering probability when $\Phi = 0$. To solve (13) we note that given a value for γ one can then run the rate-adaptation formula in (4) to (8) to compute $A_i(t)$, $B_i^*(t)$, and the performance metric in (12). The maximum γ can then be obtained using standard techniques such as binary search.

In addition to rebuffering ratio, another common metric is in terms of number of rebuffering events which can also be easily incorporated into the PSRA framework as follows:

$$\gamma_i^{\max} = \max\{\gamma | n_i \leq \Psi\}, \tag{14}$$

where Ψ is the target number of rebuffering events. Clearly there are many other possible performance metrics and they can be incorporated into PSRA in a similar fashion.

It is worth reiterating that the computed $\gamma_i^{\max}$ incorporated the impact of all network and system parameters, e.g., throughput variations via $C_{i,j}$; the rate-adaptation algorithm via $\hat{r}_{i,k}$; video duration via limit of t; and the set of available video bitrates via $\{v_x\}$ and the target streaming performance Φ or Ψ.

4.2 System Operation

In the analysis phase PSRA computes a statistical model for the control parameter $\gamma_i^{\max}$ over a period of past throughput trace data. The resultant statistical model is then used in the prediction phase to directly control the rate adaptation algorithm via its control parameter, i.e., γ. These two phases are then repeated periodically, e.g., daily, to update the statistical model with new throughput trace data, thereby enabling it to adapt to long-timescale evolution in the network's properties.

Analysis Phase—The goal of the analysis phase is to characterize the relation between streaming performance and the adaptation algorithm's control parameter, i.e., γ. Our analysis revealed that network characteristics are throughput-dependent. To exploit this discovery we divide the past

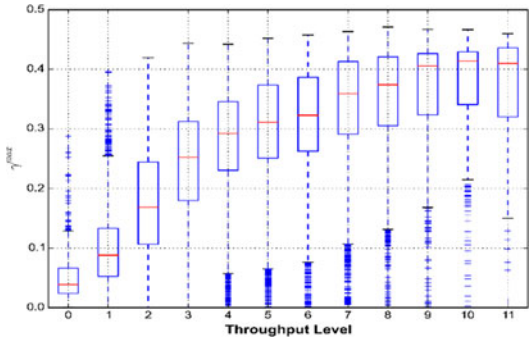

Fig. 1. Comparison of γ^{max} metric for different throughput levels. (The box represents the middle two quartiles, with the band inside marking the median. The whiskers at the top/bottom extremes mark the highest/lowest data points within 1.5 inter-quartile ranges of the upper/lower quartiles [45].).

H days' throughput trace data into L throughput levels, with level l, $L > l \geq 0$, comprising sessions where the average throughput in transferring the first M segments are within $(lC_{\max}/L, (l+1)C_{\max}/L]$.

For each throughput level PSRA computes the resultant $\gamma_i^{\max}$ for all past streaming sessions using (13) or (14) by executing the adaptive streaming algorithm using past throughput trace data. Naturally the computed $\gamma_i^{\max}$ will vary from session to session due to throughput variations but we can generate the *empirical* cumulative distribution function (CDF) for $\gamma_i^{\max}$, denoted by $F_l(\cdot)$, for throughput level l. The distribution $F_l(\cdot)$ thus captures the statistical relation between the control parameter γ and the target streaming performance.

Fig. 1 plots the throughput statistics across $L = 12$ throughput levels, with $C_{\max} = 12$ Mbps. We observe that the median $\gamma_i^{\max}$ varies substantially from 0.04 at level 0 to 0.41 at level 11. This demonstrates that to achieve the target streaming performance consistently across all throughput levels one will need to optimize the streaming parameter for each throughput level. We conjecture that this is a reflection of different types of network conditions. For example, a low average throughput is indicative of poor network condition (e.g., poor coverage, peak hours, etc.) which tends to exhibit higher coefficient-of-variation in throughput, thereby lowering the resultant $\gamma_i^{\max}$. PSRA captures and exploits this knowledge by generating the $\gamma_i^{\max}$ distribution separately for each throughput level.

Prediction Phase—In the prediction phase PSRA applies the collected statistical model, i.e., $\{F_l(\cdot) \,|\, l = 0, 1, \ldots, L-1\}$, in bitrate selection. As discussed earlier it is desirable to offer a tool for the service provider to explicitly *control* the tradeoff between video quality and streaming performance. PSRA supports this by allowing the service provider to specify a *target* streaming performance in terms of the probability to exceed the rebuffering ratio Φ (c.f. (13)), denoted by α, for its service. The challenge is in finding a way to relate α to the rate adaptation algorithm's control parameter γ.

Specifically, the set of CDFs $\{F_l(\cdot) | l = 0, 1, \ldots, L-1\}$ can be interpreted as the probability distribution of the control parameter γ which resulted in satisfying the rebuffering ratio Φ over the past H days. If this statistical property remains consistent for new streaming sessions, then we can determine the desired γ for throughput level l, denoted by γ_l, to achieve the target streaming performance α, from

$$\gamma_l = F_l^{-1}(\alpha). \tag{15}$$

Note that α needs not be fixed. A service provider can dynamically select different targets for different services or content types, e.g., lower for music videos. PSRA simply computes a different set of γ_l for use in each service class using the *same* statistical model obtained from the analysis phase. Finally, for a new streaming session the system first determines its throughput level by measuring the average throughput in downloading the first M video segments during startup and then applies the corresponding operating parameter γ_l to its rate adaptation algorithm as described in Section 4.1.

4.3 Deployment Options

PSRA can be implemented at the server side, at the client side, or a combination of the two. For example, it is common for content providers to co-locate their servers inside service provider's network. In this case the server hosting the contents can be easily extended to record throughput data while it delivers the video data to the clients over HTTP/TCP. One can then apply PSRA to generate the statistical model from the throughput trace data to optimize the parameter for the rate adaptation algorithm.

If the rate adaptation logic is implemented at the client then it is simply a matter of sending the optimized parameters (e.g., γ's for each throughput level) to the client's video playback software. This can be done by embedding the parameters into the meta-data file (e.g., m3u8 playlist) in existing streaming protocols. The client can then measure its session's throughput level during prefetch and select the corresponding parameter for use in the adaptation algorithm.

Alternatively, rate adaptation can also be implemented at the server (or a proxy). In this case the client does not execute any adaptation and simply plays whatever video segments sent by the server. The server adapts video bitrate simply by sending video segments with bitrates as determined by the rate-adaptation algorithm and parameters optimized by PSRA. This was the approach employed in our prototype implementation as it does not require modification to the client or its video player.

Finally, we note that PSRA can also be applied to streaming services delivering encrypted contents. Existing streaming protocols such as Apple's HLS has provisions for content encryption. Such encryption will not affect PSRA as PSRA does not need to decode the video data at all to function.

4.4 Discussions

The PSRA framework is designed for practical deployment. This is reflected in three design choices. First, in applying the framework to a rate-adaptation algorithm the computation complexity should be low as bitrate decision needs to be performed frequently. This is achieved by dividing PSRA into two phases where most of the computations are performed in the analysis phase which is only done, say, once a day. In contrast, in the prediction phase the only computations needed are: (a) throughput measurement during prefetch; and (b) computation of the control parameterγ_l. Both are not computationally expensive and are performed only once at the beginning of the streaming session.

Second, in determining the throughput level and the control parameter we choose to make use of the average throughput in downloading the first M segments during prefetch, as opposed to using the throughput of the previous streaming session [15], [36], [37]. This new approach eliminates potential errors in cases where the previous session was separated by a long time or was impacted by conditions specific to that session.

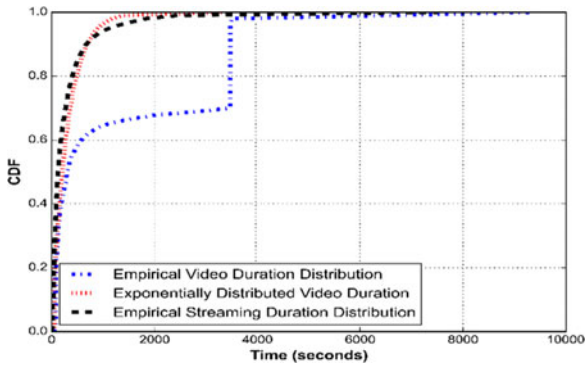

Fig. 2. Video duration and streaming duration distributions.

Third, PSRA is designed to complement (rather than replace) the underlying rate-adaptation algorithm to enable predictable streaming performance. Therefore PSRA can be applied to any rate-adaptation algorithm as long as the latter exports a control parameter (e.g., γ) that tradeoffs between streaming performance and other metrics (e.g., picture quality). This differs from an early version of PSRA [15], [36], [37] where the adaptation algorithm was an integral part of the framework.

5 Performance Evaluation

In this section we evaluate performance of the proposed PSRA framework and compare it to 7 existing adaptive streaming algorithms. We first present an extended trace-driven simulator developed for the performance evaluations and then discuss the results obtained.

5.1 An Extended Trace-Driven Simulator

As discussed in Section 3 the use of real throughput trace data enables us to accurately recreate real mobile network conditions in a *repeatable* manner. Beyond that we extended the simulator with two new features to further improve its fidelity.

Video duration distribution—In the literature it is common to evaluate streaming performance using fixed-duration video sessions. Obviously fixed video duration is rarely the case in real-world streaming services and different services will likely have different video duration distributions. We attempt to improve the fidelity of the simulator in two ways. First, motivated by the study by Shafiq *et al.* [46] we modelled the video duration by an exponential distribution to introduce video duration variations into the simulator. Second, we collaborated with a mobile operator to measure the empirical video duration distribution (Fig. 2) in one of their mobile streaming servers. This empirical dataset offers a data point on the impact of real-world video duration on streaming algorithms.

PSRA incorporates the impact of video duration by computing a separate statistical model (i.e., represented by the set of distributions $\{F_l(\cdot)|l = 0, 1, \ldots, L-1\}$ and the choice of γ) for a given video duration. This poses a problem if the video duration is not fixed. To address this issue we can precompute distributions and determine the γ for a small number of fixed video durations and then apply log-linear interpolation to compute the corresponding γ for the exact video duration in each streaming session.

User arrival process—In a real streaming service users will not arrive in a back-to-back manner but are likely to be

TABLE 6
Comparison of Rebuffering Probability in Three Locations ($\alpha = 0.05$ for PSRA)

Loc. ID	#1	#2	#3
PSRA	0.048	0.041	0.057
LBG	0.080	0.294	0.289
LBG*	0.060	0.162	0.270
OSMF	0.074	0.222	0.001
Stagefright	0.012	0.012	0.000
FESTIVE	0.013	0.015	0.000
FESTIVE*	0.002	0.005	0.000
BBA2	0.005	0.011	0.000
BOLA-U	0.079	0.274	0.000
BOLA-U*	0.078	0.273	0.000
FastMPC	0.458	0.645	0.167

TABLE 7
Comparison of Rebuffering Probability for Different Video Durations ($\alpha = 0.05$ for PSRA)

Duration (sec)	300	1200	3600	7200
PSRA	0.044	0.049	0.052	0.073
LBG	0.068	0.128	0.175	0.217
LBG*	0.064	0.107	0.120	0.130
OSMF	0.077	0.099	0.113	0.122
Stagefright	0.002	0.014	0.033	0.038
FESTIVE	0.004	0.014	0.037	0.068
FESTIVE*	0.002	0.003	0.004	0.004
BBA2	0.003	0.006	0.012	0.023
BOLA-U	0.064	0.081	0.127	0.180
BOLA-U*	0.062	0.081	0.127	0.179
FastMPC	0.390	0.733	0.875	0.938

TABLE 8
Comparison of Rebuffering Probability Using Different Bitrate Profiles ($\alpha = 0.05$ for PSRA)

Bitrate Profile	Apple	Adobe	Stefan
PSRA	0.048	0.053	0.050
LBG	0.080	0.029	0.002
LBG*	0.060	0.009	0.001
OSMF	0.074	0.091	0.063
Stagefright	0.012	0.033	0.021
FESTIVE	0.013	0.017	0.011
FESTIVE*	0.002	0.003	0.000
BBA2	0.005	0.014	0.007
BOLA-U	0.079	0.087	0.063
BOLA-U*	0.078	0.086	0.063
FastMPC	0.458	0.501	0.494

separated by some random time intervals. We modelled the time between the end of a streaming session and the beginning of the next session by an exponentially-distributed interval of mean $1/\lambda_W$ seconds (default 300s).

PSRA operation—With 3 months' trace data for each location we ran PSRA's analysis phase beginning at week 5, using the past 4 weeks' trace data to compute the $F_l(\cdot)$ distributions. The distributions were then used to determine the control parameter γ according to the target streaming performance (i.e., α and Φ) for use in the next 24 hours. The analysis phase will be repeated in the next day using trace data from the latest past 4 weeks.

The available video bitrates are listed in Table 1, with the Apple bitrate profile being the default unless stated otherwise. The initial video bitrate for the first $M = 10$ segments is set to 1,200 Kbps—a common bitrate for 480p video resolution [47]. The client video player begins playback after receiving the first ten 2-second segments.

5.2 Streaming Performance Predictability

We first investigate PSRA's streaming performance predictability across different system parameters. Results from Section 3 demonstrated that the performance of existing protocols can vary substantially across different locations, video durations, and available bitrates. We repeated the experiments in Section 3 using the enhanced simulator and summarized the results in Table 6 to Table 8. The target streaming performance is set at $\alpha = 5\%$ with $\Phi = 0$ which is equivalent to the probability of experiencing at least one playback rebuffering in a streaming session.

Note that in the original FESTIVE [14] and BOLA-U [22] designs the client buffer size was limited to 30 and 25 seconds respectively. In the case of LBG [13] it was dynamically determined according to the selected bitrate for each video segment. As other algorithms did not impose a client buffer size limit we also simulated modified versions of LBG, FESTIVE and BOLA-U without client buffer size constraints, denoted by LBG*, FESTIVE* and BOLA-U*, for fair comparisons.

First, none of the existing algorithms supports explicit control of streaming performance so not surprisingly their performance varied substantially from one another, e.g., from 0.005 (FESTIVE*) to 0.645 (FastMPC) in location #2. More importantly, even the same algorithm can exhibit varying performance across the three locations, e.g., 0.000 (location #3) to 0.274 (location #2) for BOLA-U.

In contrast, PSRA achieved actual rebuffering probability of 0.048, 0.041, and 0.057 for the three locations respectively which are consistent with the target of $\alpha = 0.05$. This was accomplished by tuning the statistical model to the specific location's network characteristics as shown in Fig. 3a.

Second, results for four different video durations in Table 7 show that the rebuffering probabilities of existing algorithms generally increase with video duration. By contrast, PSRA was able to achieve consistent rebuffering probability for video durations 300 s, 1,200 s, and 3,600 s, again via tuning of the control parameter γ as depicted in Fig. 3b. However for the ultra-long video duration of 7,200 s the actual rebuffering probability did deviate from the target of 0.05 more significantly. We will address this problem in Section 6.

Third, results in Table 8 show that unlike existing algorithms, PSRA is insensitive to the composition of available bitrates and was able to achieve actual rebuffering probabilities close to the target. The differences in the control parameter distribution are relatively small (c.f. Fig. 3c) but nonetheless were sufficient to compensate for the bitrate profile differences.

Finally, we note that FastMPC exhibited relatively high rebuffering probabilities. Our analysis of the trace data suggests that this is due to FastMPC's use of a sophisticated QoE metric which incorporated not only rebuffering, but also video quality. In fact in its QoE metric video quality appears to be weighted significantly more than rebuffering and hence it tends to favor higher video bitrate at the expense of more rebuffering. Interested readers are referred to Appendix II, available in the online supplemental material for a more detailed analysis.

All in all the above results verified one of PSRA's key design goals—to provide *predictable* and *consistent* streaming performance across a wide range of system parameters.

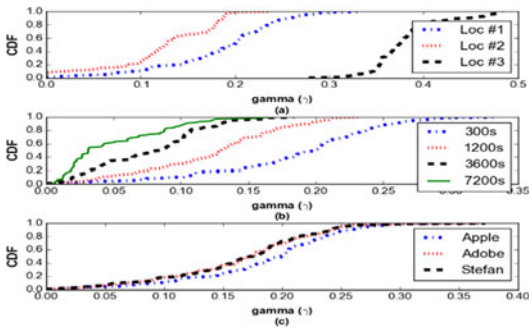

Fig. 3. Distribution of γ generated by PSRA for different (a) locations, (b) video durations, and (c) video bitrate profiles.

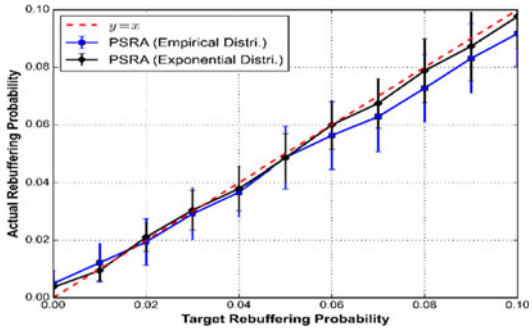

Fig. 4. Target versus actual rebuffering probability (weekly-average) achieved by PSRA over two video duration distributions (error bars show 90 percent confidence interval).

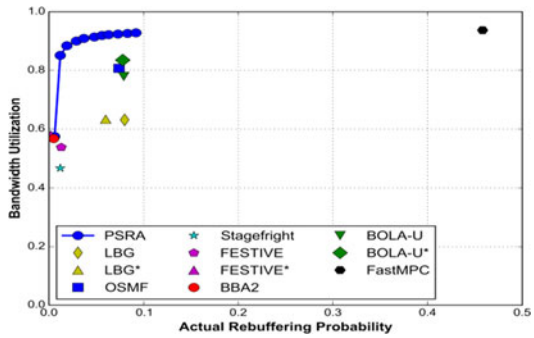

Fig. 5. Tradeoff between video quality (i.e.,bandwidth utilization) and streaming performance (i.e., rebuffering probability).

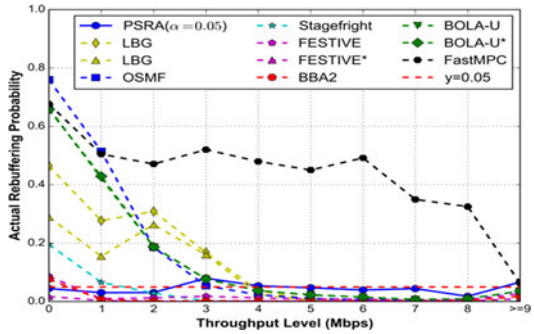

Fig. 6. Variations in rebuffering probability across different throughput levels (level x is from x Mbps to (x+1) Mbps, except level 9 which includes throughput $\geq$ 9 Mbps).

As a side benefit it also enables the service provider to more freely configure *other* system parameters such as available video bitrates without impacting streaming performance in an unpredictable manner.

5.3 Controlled Performance Tradeoff

In addition to streaming performance predictability and consistency, PSRA also offers, for the first time, a tool for service providers to explicitly control the tradeoff between streaming performance and video quality. To evaluate this capability we plot in Fig. 4 the actual versus target rebuffering probabilities over a range from 0 to 10 percent.

The results show that PSRA was able to achieve actual rebuffering probabilities reasonably close to the target. Results for the two different video duration distributions are similar up to $\alpha = 0.06$, beyond which the actual rebuffering probabilities for the empirical video distribution deviated slightly more than the exponential counterpart.

The tradeoff to higher streaming performance will be lower average video quality. The latter can be measured via bandwidth utilization—the ratio of actual throughput utilized over the amount available. A streaming session may not utilize all available bandwidth if it completely delivered all video data *before* playback ends, e.g., by selecting video bitrates too conservatively.

Fig. 5 plots the controlled tradeoff between actual rebuffering probability and bandwidth utilization. Existing algorithms were not designed to allow controlled tradeoff between the two metrics and hence each achieved one specific point of tradeoff. This is a significant limitation in practice as difference services and even different contents can have very different performance requirements. The results also revealed the very different design priorities of existing algorithms, with some such as Stagefright trading off video quality for low rebuffering probability while others such as FastMPC achieving the opposite.

In comparison, PSRA generally achieved better tradeoffs, i.e., higher bandwidth utilization at the same rebuffering probability and vice versa, than existing algorithms. More importantly, bandwidth utilization levels off for actual rebuffering probabilities over 0.03—suggesting that it is not necessary to significantly sacrifice streaming performance to achieve good video quality—a target of 0.03$\sim$0.05 for example, would provide a good tradeoff between the two conflicting performance metrics.

5.4 Variation Across Network Conditions

As discussed in Section 4.2 the average throughput of a streaming session is a good indicator of the underlying network conditions, and this is why PSRA generates a separate statistical model for each throughput levels. To validate this we plot in Fig. 6 the actual rebuffering probabilities for each of the 10 throughput levels, with level $x = 1, 2, \ldots, 8$ collecting sessions with average throughput within (x, $x + 1$] Mbps, plus level 9 with average throughput $\geq$ 9 Mbps.

We observe that most existing algorithms exhibited significantly higher rebuffering probability at lower throughput levels. In contrast, PSRA achieved substantially more consistent streaming performance across all throughput levels. To see why, consider the bandwidth utilization in Fig. 7 which shows that only PSRA lowered the bandwidth utilization at lower throughput levels, i.e., it becomes more conservative in its bitrate selections, so that the target streaming performance can be maintained under the poorer network conditions. This unique and desirable property is a result of PSRA's statistical model in which a separate $F_l(\cdot)$ distribution

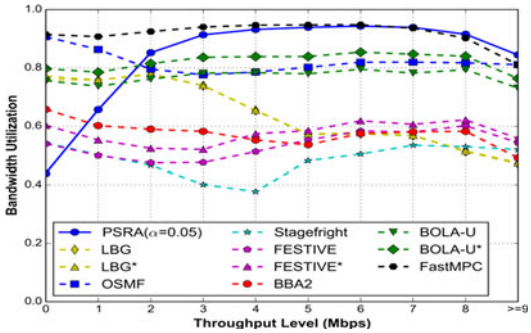

Fig. 7. Comparison of bandwidth utilization versus throughput levels.

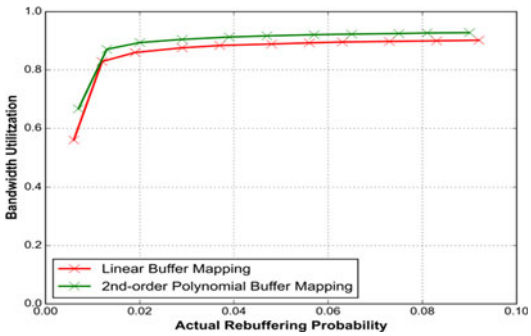

Fig. 8. Tradeoff between rebuffering probabilty and bandwidth utiltization.

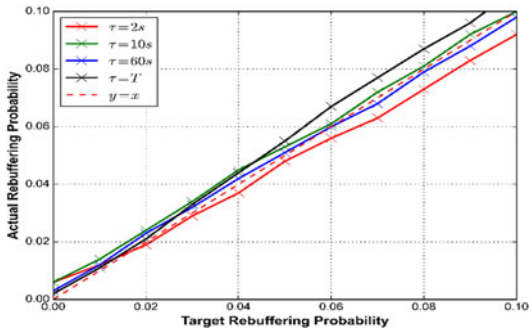

Fig. 9. Impact of bitrate-switching interval on streaming performance predictability.

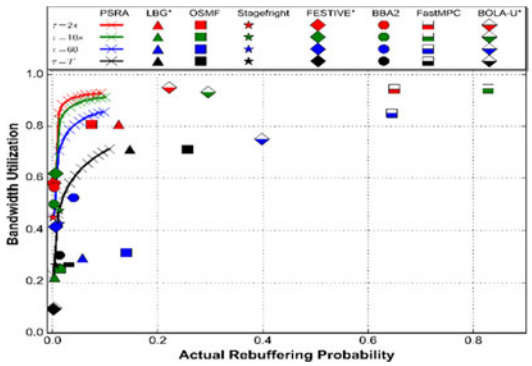

Fig. 10. Performance comparison over bitrate switching intervals of 2 s, 10 s, 60 s, and T (i.e., video duration) of different algorithms.

is generated for each throughput level, thereby enabling it to capture and compensate for the different *degrees* of bandwidth variations under different network conditions.

5.5 Application to Different Adaptation Algorithms

As discussed in Section 4.1 PSRA can be easily extended to support different types of bitrate adaptation algorithms. To illustrate we simulated PSRA with the modified second-order mapping function for client buffer occupancy in (5) and plot the results in Fig. 8. Interestingly the modified second-order buffer mapping function did achieve slightly higher bandwidth utilization. In this case the slightly more aggressive adaptation algorithm enables one to achieve better video quality while maintaining the same target streaming performance.

In a second experiment we investigated the impact of bitrate switching frequency on the performance tradeoffs between video quality and streaming performance. We first investigate in Fig. 9 the impact of bitrate switching period on rebuffering probability consistency. Interestingly, varying the bit-rate switching period appeared to have little impact. Even setting the bitrate switching interval to equal to the video duration, i.e., only *one* bitrate selection at the beginning after prefetch and then fixed afterwards, did *not* cause PSRA to fail achieving the target rebuffering probability.

Further analysis revealed that the real impact is elsewhere. Fig. 10 plots the bandwidth utilization and the results reveal that lengthening the bitrate switching period did degrade bandwidth utilization, i.e., average video quality. Intuitively, the longer the delay for a streaming algorithm to react to bandwidth variations, the more conservative it will become when trained in the analysis phase of PSRA.

In other words, with longer interval between bitrate switches PSRA automatically tuned the rate-adaptation algorithm to become more conservative in video bitrate selection to compensate for the increased likelihood of playback rebuffering.

More interestingly, we observe that the bandwidth utilization levels off rather quickly with shorter bitrate switching intervals. For example, PSRA's bandwidth utilization reached around 0.8 even with an interval of 60 seconds. This is in contrast to the current industry practice where the bitrate switching period is typically 10 s or even shorter. This result suggests that with PSRA one can adopt a much longer bitrate switching interval to improve the user experience while still achieving the target streaming performance.

In comparison, the impact of bitrate switching interval on the existing algorithms is far less consistent. In fact, few of the existing algorithms exhibited consistent increase in bandwidth utilization with shorter bitrate switching intervals. For example, shortening the bitrate switching interval from 60 s to 10 s *degraded* the bandwidth utilization for LBG, OSMF and BBA2. This likely reflects the fact that these existing algorithms were not designed to support configurable bitrate switching interval and thus directly tuning the latter may not necessary lead to the expected tradeoffs.

Overall with the same bitrate switching interval PSRA outperforms existing algorithms across a wide range of actual rebuffering probabilities. For example, with a bitrate switching interval of 60 s PSRA can achieve bandwidth utilization 174.2, 166.9, 62.3, 6.3, 54.8, 0.5, 14.0 percent higher than that of LBG*, OSMF, Stagefright, FESTIVE*, and BBA2, FastMPC, BOLA-U* respectively, while at the same time achieving equal or lower actual rebuffering probability.

5.6 Application to Different Performance Metrics

Section 4.2 defines three streaming performance metrics: (a) rebuffering probability—the probability for a streaming

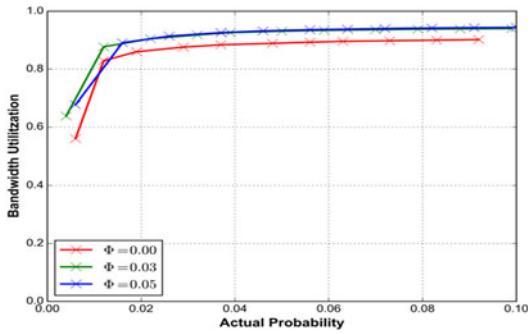

Fig. 11. Tradeoff between rebuffering ratio probabilty and bandwidth utiltization.

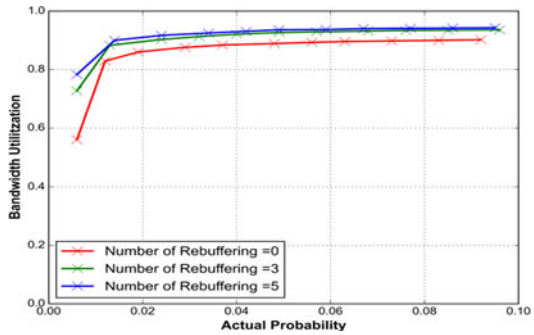

Fig. 12. Tradeoff between rebuffering number probabilty and bandwidth utiltization.

session to encounter one or more rebuffering events; (b) rebuffering ratio—the probability for a streaming session to encounter rebuffering ratio equal to or larger than Φ (cf. (13)); and (c) rebuffering number—the probability for a streaming session to encounter number of rebuffering events equal to or larger than Ψ (cf. (14)).

Clearly these three different metrics measure different aspects of degradation in streaming quality. Our goal here is not to determine the optimal metric which is likely to be dependent on the services and contents being provisioned. Instead we want to demonstrate that the PSRA framework can be applied to systems with different streaming performance metrics and can consistently achieve the streaming performance target.

We first investigate the second metric—rebuffering ratio for $\Phi = \{0.00, 0.03, 0.05\}$. Note that $\Phi = 0.00$ implies no rebuffering and thus the probability is equivalent to the special case of rebuffering probability. If the video duration is 5 minutes (i.e., 300 s) then $\Phi = 0.03$ represents a total of 9 seconds of rebuffering time (i.e., suspended playback) which can occur in one or more rebuffering events.

The results (not shown) confirmed that PSRA can achieve actual streaming performance close to the target for all three cases. Now the bandwidth utilization plot in Fig. 11 reveals the impact of the target rebuffering ratio Φ - allowing some rebuffering (i.e., when $\Phi = 0.03, 0.05$) resulted in slightly higher bandwidth utilization. This is expected as it allows a slightly more aggressive choice of higher video bitrates which PSRA exploited automatically.

Next we consider the third metric—number of rebuffering events encountered in a streaming session. Again the observations are similar, i.e., good streaming performance predictability across rebuffering numbers of 0, 3, and 5 (not shown); and slight gains in bandwidth utilization for larger rebuffering numbers as shown in Fig. 12.

5.7 Computation Complexity

In PSRA, the primary processing task is in generating the cumulative distribution for $\gamma_i^{\max}$ in the analysis phase. The amount of computation required is primarily determined by the number and length of video durations, and bitrate switching frequency. Note that the size of the past window (i.e., H days) where PSRA computes its statistical model does not impact its complexity (except for the first iteration) as the statistical model can be updated *incrementally*, e.g., if the analysis phase is executed daily then computations are required only for the new trace data obtained in the previous day.

For example, in our experiments it took an Intel Core-i7 2600 3.4 Ghz Linux machine a total of 408 seconds to generate the distributions of $\gamma_i^{\max}$ for 21 video durations ranging from 50 s to 10,800 s using just one of the CPU cores. For a mobile network with 2,000 locations PSRA can update its statistical model in around 18 hours using just a single PC with 12 CPU cores. Note that the computation can also be done as trace data are captured so the computations can be spread over time. In practice, one can simply run the analysis within the same servers operating the streaming service as the latter's I/O-bound nature nicely complements PSRA's compute-bound analysis.

6 Long Streaming Sessions

Results from Section 5.1 revealed one limitation of PSRA, i.e., its performance consistency degraded for ultra-long video sessions (e.g., 0.073 versus a target of 0.05 for 2-hour video). While such long video sessions are uncommon in today's mobile streaming services we are interested in its cause and to explore possible solutions.

We conjecture that the deviation is due to the way throughput trace data were collected for use in the analysis phase. Specifically, as throughput trace data were captured as a by-product of actual streaming sessions, they are *not* necessarily of the required duration. Therefore in determining $\gamma_i^{\max}$ for a given video duration in the analysis phase we may need to *concatenate* multiple sessions' throughput trace data to accumulate the required duration. However, when applied to the prediction phase the new streaming session is obviously a whole session rather than one concatenated from multiple shorter sessions. This difference will increase with longer video durations, thereby explaining why the deviation only occurs at the ultra-long video duration of 7,200 s.

Unless one perform continuous throughput measurement to generate the trace data for PSRA's analysis phase, concatenating trace data from multiple separate streaming sessions is unavoidable. Therefore we explore the use of feedback as a way to mitigate the problem.

6.1 PSRA with Feedback

Feedback is a common technique for reducing errors in a system with one or more control parameters. We investigate the use of Proportional Integral (PI) controller [48] for use in PSRA. A PI controller has a process variable and a setpoint. The process variable represents the current state of the system, i.e., *actual* rebuffering probability, while the setpoint is the target for the system state, i.e., *target* rebuffering probability. The purpose of the PI

TABLE 9
Comparison of Weekly-Averaged Rebuffering Probability (90% CI in Brackets) for PSRA and Feedback-PSRA Streaming 7200-s Videos ($\alpha = 0.05$)

Location	PSRA	Feedback-PSRA
#1	0.073 ($\pm$0.035)	0.055 ($\pm$0.022)
#2	0.051 ($\pm$0.012)	0.049 ($\pm$0.012)
#3	0.048 ($\pm$0.023)	0.053 ($\pm$0.021)

controller is to reduce the error between process variable and setpoint by adjusting system inputs. The controller output $u(t)$ is defined as follows:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau, \tag{16}$$

where $e(t)$ represents the error between process variable and setpoint at time t. K_p and K_i are tuning parameters representing the proportional gain and integral gain respectively. Interested readers are referred to [48] for more details.

To apply a PI controller to PSRA we adjust the *daily* target rebuffering probability, denoted by α_k for day k, at the beginning of each day based on the difference (i.e., error) between the actual probability realized and the target α.

To avoid potential error due to insufficient number of testing sessions, we initialize $\alpha_k = \alpha$ for day 0 to day 6. From day 7 onwards, i.e., $k \geq 7$, we compute the new daily target probability by subtracting the PI controller's output from the daily target of the previous day:

$$\alpha_k = \alpha_{k-1} - u_{k-1}. \tag{17}$$

The PI controller output u_{k-1} is computed from a discrete version of (16):

$$u_{k-1} = K_p e_{k-1} + K_i \sum_{x=0}^{k-1} e_x, \tag{18}$$

where e_k is the difference between the average actual probability of the previous 7 days, denoted by $\hat{\alpha}_k$, and the target, i.e.,

$$e_k = \hat{\alpha}_k - \alpha. \tag{19}$$

The parameters K_p and K_i are tuned according to the Ziegler–Nichols method [49] to obtain $K_p = 0.45$ and $K_i = 0.54$. Interested readers are referred to Appendix III, available in the online supplemental material for more details.

6.2 Evaluation

We first compare the actual rebuffering probabilities for 7,200-s video in Table 9. Evidently the PI controller is very effective in bringing the actual rebuffering probability close to the target, e.g., from 0.073 to 0.048 at a target of 0.05 for location #1. The impacts to the other two locations are negligible as they have little deviation to begin with. Next we evaluate in Fig. 13 the time needed for the feedback-assisted PSRA to converge to the streaming performance target.

In the first week there is no feedback applied and hence location #1 and #3 exhibited larger deviation from the target. From week 2 onwards the streaming performance quickly approached the target of 0.05 and remained so afterwards. As the convergence process is needed only once at the beginning of deploying PSRA the impact of the initial deviations (only for ultra-long videos) is limited. Moreover, the computation overheads incurred by the feedback processing is negligible and hence can be easily implemented in practice.

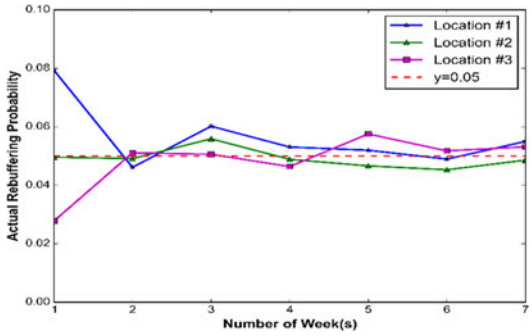

Fig. 13. Illustration of convergence for feedback-assisted PSRA.

7 Implementation

We implemented the PSRA framework into a server-based adaptive streaming system employing Apple's HLS protocol and iOS's QuickTime Video Player. Specifically, the server can operate in two modes—HLS and PSRA. In HLS mode the server just operates as a standard HTTP server configured to support Apple's HLS adaptive streaming protocol. This allows a direct performance comparison to Apple's proprietary implementation.

In PSRA mode the server implements PSRA's analysis phase and prediction phase as described in Section 4. To keep the rest of the system the same we adopted the same HLS protocol and used the same video player at the client (an iPhone 5c running iOS 9.1). As we cannot modify the client's adaptation algorithm we implemented bitrate adaptation entirely at the server side by always sending to the client a m3u8 playlist [50] with only *one* video bitrate available, effectively disabling the client's rate adaptation. At the server there are 10 bitrate versions available and the server will select the bitrate version to send based on the rate adaptation algorithm as described in Section 4. Table 10 summarizes the experimental parameters adopted.

We completed a total of 6,362 streaming sessions in a production 3G/HSPA network. Table 11 summarizes the results for PSRA with three different target rebuffering probabilities and compares that to Apple's HLS implementation. The results for Apple's HLS is consistent with previous studies [6], showing its conservativeness in bitrate selection, achieving only 1,196 kbps with a low rebuffering probability.

For PSRA, the experimental results verified its design goal to achieve predictable and consistent streaming performance. The actual rebuffering probabilities are all within 0.004 of the target in all three cases. Unlike Apple's HLS, PSRA enables controlled tradeoff between video bitrate

TABLE 10
System Parameters Used in Experiments

System Variable	System Parameter
Video prefetch size	2 segments
Segment size	10 seconds
Initial video bitrate	1,200 kbps
Max buffer size	60 seconds
Min bitrate switching period	60 seconds
Bitrate versions	Apple's [38] profile plus 10 Mbps and 12 Mbps
Video Duration	300 seconds

TABLE 11
Experimental Results

Algorithms:	PSRA ($\alpha = 0.01$)	PSRA ($\alpha = 0.05$)	PSRA ($\alpha = 0.09$)	iPhone QT Player
Number of sessions	1,627	1,574	1,534	1,627
Actual rebuffering probability	0.009	0.046	0.092	0.001
Mean bitrate (kbps)	6,253	7,614	7,885	1,196

and streaming performance. This eliminates the need to be overly conservative as in Apple's HLS such that significantly higher video quality can be achieved. For example, at an actual rebuffering probability of only 0.009, PSRA can achieve an average video bitrate of 6,253 kbps which is significantly higher than Apple's HLS at 1,196 kbps. Therefore with PSRA service providers will be able to take advantage of the higher bandwidth available in current and future mobile networks to offer high-quality streaming services to their subscribers.

8 Summary and Future Work

The PSRA framework investigated in this work offers a new approach to the design and optimization of mobile video streaming algorithms. Instead of designing and optimizing an adaptive streaming algorithm for *all* mobile networks, PSRA tunes the adaptation algorithm specifically for the network it operates in so that streaming performance can be improved, controlled, and predicted. The results demonstrated that mobile networks, despite their well-known bandwidth fluctuations, do exhibit sufficiently consistent properties that enable PSRA to automatically optimize the adaptation algorithm based on past traffic traces.

This work is only a first step in this direction. There are many opportunities for future research. For example, as the rate-adaptation algorithm is decoupled from the PSRA framework it means one can replace the former to achieve different design goals and to support the tradeoff between different performance metrics. On the other hand, as PSRA optimizes the system's operating parameters for the network it operates in, it enables one to more precisely characterize the performance impact of different design choices (e.g., choice of input metrics such as bandwidth, buffer occupancy, etc.) and system constraints (e.g., number of video bitrate levels, bitrate switching interval, maximum bitrate changes, client buffer size, etc.). These insights will offer guides to the design of a new generation of adaptation algorithms.

Acknowledgments

The authors wish to thank the associate editor and the anonymous reviewers for their insightful comments in improving this paper. This work was funded in part by a research grant (GRF14203714) from the HKSAR Research Grant Council and the Research Committee of CUHK.

References

[1] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019*, Cisco, Feb. 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[2] *Apple HTTP Live Streaming*. (May 2016). [Online]. Available: https://developer.apple.com/resources/http-streaming/

[3] *Microsoft Smooth Streaming*. (May 2016). [Online]. Available: http://www.microsoft.com/silverlight/smoothstreaming/

[4] *Adobe HTTP Dynamic Streaming*. (May 2016). [Online]. Available: http://www.adobe.com/products/hds-dynamic-streaming.html

[5] Netflix. (May 2016). [Online]. Available: https://www.netflix.com/global

[6] H. Riiser, H.S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz, "A comparison of quality scheduling in commercial adaptive HTTP streaming solutions on a 3G network," in *Proc. 4th Workshop Mobile Video*, Feb. 2012, pp. 25–30.

[7] F. Dobrian, et al., "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM*, Aug. 2011, pp. 362–373.

[8] R. K. Mok, E. W. Chan, and R. K. Chang, "Measuring the quality of experience of HTTP video streaming," in *Proc. IFIP/IEEE Int. Symp. Integrated Netw. Manage.*, May 2011, pp. 485–492.

[9] A. Rao, A. Legout, Y.S. Lim, D. Towsley, C. Barakat, and W. Dabbous, "Network characteristics of video streaming traffic," in *Proc. 7th Conf. Emerging Netw. Experiments Technol.*, Dec. 2011, Art. no. 25.

[10] P. Ameigeiras, J. J. Ramos-Munoz, J. Navarro-ortiz, and J. M. Lopez-Soler, "Analysis and modelling of YouTube traffic," *IEEE Trans. Emerging Telecommun. Technol.*, vol. 23, no. 4, pp. 360–377, Jun. 2012.

[11] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 2, pp. 16:1–16:22, May 2008.

[12] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld and P. Tran-Gia, "A survey on quality of experience of HTTP adaptive streaming," *IEEE Commun. Surveys Tutorials*, vol. 17, no. 1, pp. 469–492, Jan.-Mar. 2015.

[13] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate adaptation for adaptive HTTP streaming," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, Feb. 2011, pp. 169–174.

[14] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with FESTIVE," in *Proc. Conf. Emerging Netw. Experiments Technol.*, Dec. 2012, pp. 97–108.

[15] Y. Liu and J. Y. B. Lee, "Providing predictable streaming performance in mobile video streaming," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2014, pp. 2276–2281.

[16] Adobe Open Source Media Framework (OSMF). (May 2016). [Online]. Available: http://www.osmf.org

[17] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback Control for Adaptive Live Video Streaming," in *Proc. ACM Conf. Multimedia Syst.*, Feb. 2011, pp. 145–156.

[18] T. Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM SIGCOMM*, Aug. 2014, pp. 187–198.

[19] T.Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: Picking a video streaming rate is hard," in *Proc. ACM Conf. Internet Measurement Conf.*, Nov. 2012, pp. 225–238.

[20] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic HTTP streaming," in *Proc. Int. Conf. Emerging Netw. Experiments Technol.*, Dec. 2012, pp. 109–120.

[21] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, Aug. 2015, pp. 325–338.

[22] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[23] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang, "QDASH: A QoE-aware DASH system," in *Proc. Multimedia Syst. Conf.*, Feb. 2012, pp. 11–22.

[24] X. Xie, X. Zhang, S. Kumar, and L. Li, "piStream: Physical layer informed adaptive video streaming over LTE," in *Proc. MobiCom*, Sep. 2015, pp. 413–425.

[25] J. Hao, R. Zimmermann, and H. Ma, "GTube: Geo-predictive video streaming over HTTP in mobile environments," in *Proc. ACM Multimedia Syst. Conf.*, Mar. 2014, pp. 259–270.

[26] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Video streaming using a location-based bandwidth-lookup service for bitrate planning," *ACM Trans. Multimedia Comput. Commun. Applicat.*, vol. 8, no. 3, pp. 24:1–24:19, Jul. 2012.

[27] J. Yao, S. S. Kanhere, and M. Hassan, "Improving QoS in high-speed mobility using bandwidth maps," *IEEE Trans. Mobile Comput.*, vol. 11, no. 4, pp. 603–617, Apr. 2012.
[28] A. Bokani, M. Hassan, S. Kanhere, and X. Zhu, "Optimizing HTTP-based adaptive streaming in vehicular environment using Markov decision process," *IEEE Trans. Multimedia*, vol. 17, no. 12, pp. 2297–2309, Dec. 2015.
[29] Akamai HD for the Adobe Flash Platform. (May 2016). [Online]. Available: http://wwwns.akamai.com/hdnetwork/demo/flash/default.html
[30] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the internet," *IEEE Multimedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.
[31] T. Stockhammer, "Dynamic adaptive streaming over HTTP: standards and design principles," in *Proc. ACM Conf. Multimedia Syst.*, Feb. 2011, pp. 133–144.
[32] OSMF Algorithm. (May 2016). [Online]. Available: https://sourceforge.net/adobe/osmf/svn/2491/tree/osmf/trunk/framework/OSMF/org/osmf/net/httpstreaming/DownloadRatioRule.as#l79
[33] The Android Open Source Project. Feb. 2015. [Online]. Available: https://android.googlesource.com/platform/frameworks/av/+/master/media/libstagefright/httplive/LiveSession.cpp
[34] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. ACM Conf. Multimedia Syst.*, Feb. 2011, pp. 157–168.
[35] L. De Cicco and S. Mascolo, "An experimental investigation of the akamai adaptive video streaming," *HCI Work Learning, Life Leisure*, vol. 6389, pp. 447–464, 2010.
[36] Y. Liu and J. Y. B. Lee, "On adaptive video streaming with predictable streaming performance," in *Proc. IEEE Global Communications Conf.*, Dec. 2014, pp. 1164–1169.
[37] Y. Liu and J. Y. B. Lee, "Streaming variable bitrate video over mobile networks with predictable performance," in *Proc. IEEE Wireless Comm. Netw. Conf.*, Apr. 2016, pp. 1–7.
[38] Y. Liu and J. Y. B. Lee, "An empirical study of throughput prediction in mobile data networks," in *Proc. IEEE Global Commun. Conf.*, Dec. 2015, pp. 1–6.
[39] K. Liu and J. Y. B. Lee, "On improving TCP performance over mobile data networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 10, pp. 2522–2536, Oct. 2016.
[40] Apache HTTP Server Project. (May 2016). [Online]. Available: http://httpd.apache.org/
[41] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM Operating Sys. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.
[42] Apple Inc., Best Practices for Creating and Deploying HTTP Live Streaming Media for the iPhone and iPad, Jan. 2015. [Online]. Available: https://developer.apple.com/library/ios/technotes/tn2224/_index.html
[43] Maxim Levkov, *Video encoding and transcoding recommendations for HTTP Dynamic Streaming on the Adobe® Flash® Platform*, Oct.2010. [Online]. Available: http://download.macromedia.com/flashmediaserver/http_encoding_recommendations.pdf
[44] L. Stefan, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proc. ACM Multimedia Syst. Conf.*, Feb. 2012, pp. 89–94.
[45] Y. Benjamini, "Opening the box of a boxplot," *American Statistician*, vol. 42, no. 4, pp. 257–262, 1988.
[46] M. Shafiq, J. Erman, Lu Ji, A. Liu, J. Pang, and J. Wang, "Understanding the impact of network dynamics on mobile video user engagement," in *Proc. ACM SIGMETRICS*, Jun. 2014, pp. 367–379.
[47] YouTube, *Advanced encoding settings*. (May 2016). [Online]. Available: https://support.google.com/youtube/answer/1722171
[48] K. H. Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 4, pp. 559–576, Jul. 2005.
[49] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Trans. ASME*, vol. 64, no. 11, pp. 759–765, Nov. 1942.
[50] R. Pantos and W. May, "HTTP live streaming," *IETF Internet-Draft Draft-Pantos-Http-Live-Streaming-19, IETF*, Apr. 2016.
[51] M. Christopher, *MPEG-DASH versus Apple HLS versus Microsoft Smooth Streaming versus Adobe HDS*, Mar.2015. [Online]. Available: https://bitmovin.com/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/
[52] Y. Liu, S. Dey, D. Gillies, F. Ulupinar, and M. Luby, "User experience modeling for DASH video," in *Proc. Int. Packet Video Workshop*, Dec. 2013, pp. 1–8.
[53] A. Balachandran, V. Sekar, A. Akella, S. Seshan, and I. Stoica, "A quest for an internet video quality-of-experience metric," in *Proc. ACM Workshop Hot Topics Netw.*, Oct. 2012, pp. 97–102.

Yan Liu received the BEng degree in communication engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012 and the PhD degree in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 2016. He is currently a senior engineer at Huawei Technologies Co., Ltd., where he participated in the research and development of multimedia technology.

Jack Y. B. Lee (M'95–SM'03) received the BEng and PhD degrees in information engineering from the Chinese University of Hong Kong, Shatin, Hong Kong, in 1993 and 1997, respectively. He is currently an associate professor in the Department of Information of the Chinese University of Hong Kong. His research group focuses on research in multimedia communications systems, mobile communications, protocols, and applications. He specializes in tackling research challenges arising from real-world systems. He works closely with the industry to uncover new research challenges and opportunities for new services and applications. Several of the systems research from his lab have been adopted and deployed by the industry.