Computer Networks 133 (2018) 90–103

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A Practical Evaluation of Rate Adaptation Algorithms in HTTP-based Adaptive Streaming

Ibrahim Ayad, Youngbin Im*, Eric Keller, Sangtae Ha

University of Colorado, Boulder, United States

ARTICLE INFO

Article history:
Received 16 July 2017
Revised 14 January 2018
Accepted 16 January 2018
Available online 3 February 2018

Keywords:
HTTP-based Adaptive Streaming
Dynamic Adaptive Streaming over HTTP
MPEG-DASH
Evaluation

ABSTRACT

The HTTP-based Adaptive Streaming (HAS) techniques are widely used in Internet video streaming services, including YouTube and Netflix. The Dynamic Adaptive Streaming over HTTP (DASH) is the latest international standard that facilitates the interoperability of different HAS techniques of various vendors. DASH specification defines the media presentation description (MPD), which describes a list of available content, URL addresses, and the segment format. The rate adaptation algorithms, however, are not part of the standard, and the details of the algorithms are left to vendors. As a result, there are many different algorithms adopted in both commercial and open source players while the detailed algorithms and their performance are barely understood. In this paper, we investigate the detailed operations of the different players by code level analysis and through reverse engineering. Specifically, we present the pseudo codes of 3 open source players and devise a method to obtain the detailed operation information, e.g., bitrate and buffer amount, of popular streaming players whose source codes are not publicly available. We conduct extensive experiments on our testbed and provide suggestions based on the behaviors of these players, including the repeated over-estimation of the available bandwidth, unfair bitrate selection when multiple players compete for the bandwidth, and insensitivity of Quick UDP Internet Connections (QUIC) protocol to the varying network bandwidth.

1. Introduction

Recent studies show that video traffic on the Internet has grown exponentially over the years. Cisco [1] anticipated consumer Internet video traffic will be 80 percent of all consumer Internet traffic in 2019, up from 64 percent in 2014. It will reach 89,319 petabytes (one petabyte equivalent to one million gigabytes) per month in 2019. This increase in video traffic is mainly due to recent developments of devices and streaming technologies for video services.

Initially, Internet video streaming was implemented by using traditional streaming protocols such as Real Time Protocol (RTP) over User Datagram Protocol (UDP). But since firewalls usually block UDP packets in the networks, making it hard to deliver the content to the users, the use of HTTP over TCP for video streaming was the natural next step for the streaming industry. In fact, HTTP provides several advantages over its predecessors. It enables the reuse of existing web infrastructure such as web proxies and content distribution networks (CDNs). The HTTP-based progressive download was used for video dissemination as was leveraged by YouTube in its early days, but it had no adaptability to varying network conditions, leading to the adoption of HTTP-based Adaptive Streaming (HAS) solutions.

In HAS, the video is segmented into several-second time intervals (2 to 10 seconds, typically) and encoded into multiple bitrates. Then the HAS client selects the best bitrate that fits the current network conditions. The MPEG Dynamic Adaptive Streaming over HTTP (DASH) [2] is a standard for HAS solutions. MPEG-DASH, however, standardizes only the meta information of the video content, specifying the details of each video segment such as duration, encoding rates, and the URL address, not the rate adaptation [3].

Even though extensive work has been done in the HTTP adaptive streaming arena, earlier experimental works [4–6] only focus on the performance of streaming players in constrained experimental scenarios. They do not perform a detailed study of rate adaptation algorithms, which are player specific and thus are not well-known. Our work differs from previous works in several ways. First, while many of the previous HAS evaluation works centered on the comparison of proprietary HAS players, our evaluation also covers systems developed under the MPEG DASH standard. Second, the operational information of commercial streaming players investigated in previous works was limited, but we introduce

* Corresponding author.
E-mail addresses: ayad.ibrahim@colorado.edu (I. Ayad), youngbin.im@colorado.edu (Y. Im), eric.keller@colorado.edu (E. Keller), sangtae.ha@colorado.edu (S. Ha).

https://doi.org/10.1016/j.comnet.2018.01.019
1389-1286/

novel mechanisms to obtain detailed information from commercial streaming players whose source codes are not public. In this paper, we make the following contributions:

1. We provide the detailed operation of rate adaptation of several existing DASH clients by code level analysis. MPEG does not standardize the rate adaptation, so it is different on each implementation.
2. We devise a method to obtain the detailed operation information (e.g., bitrate, buffer amount) of popular commercial streaming players whose source code are not publicly available. We utilize a browser extension, WebDriver [7], to programmatically collect and control the behavior of the players.
3. On our HAS experiment testbed, we conduct extensive experiments by varying different parameters and find several observations on the performance of 6 up-to-date representative HAS/DASH clients: (i) there are significant differences among the players in terms of bitrate adaptation, primarily due to the differences in the video type of service (i.e., YouTube, Netflex, or Vimeo), (ii) some clients repeatedly over-estimate the available bandwidth, (iii) the bitrate selection is unstable and unfair when multiple clients compete, even with the clients of homogeneous algorithms, and (iv) the Quick UDP Internet Connections (QUIC) protocol used by YouTube player is insensitive to the varying network bandwidth and arrivals of other TCP flows, reducing its fairness with other players.
4. We identify one of the main causes of rate adaptation misbehaviors in the HAS players, which we call “buffer effect,” and present a solution to mitigate this effect. The presented solution is generally applicable to history-based bandwidth estimation algorithms.
5. Based on the observations from experiments, we suggest several guidelines to help improve the performance (i.e., bandwidth estimation and fairness) of rate adaptation algorithms on different players.

The rest of the paper is organized as follows. We briefly discuss related works in Section 2. Then we introduce our experimental framework and the detailed operation of HAS clients we used for the experiments in Section 3. We examine the operation of several HAS clients in various network environments in Section 4. Section 5 evaluates the performance of HAS clients in more advanced scenarios with varying bandwidth, background traffic, and multiple players. Section 6 provides the details of buffer effect we identified and presents its solution. Section 7 summarizes the key observations and suggests several solutions, while Section 8 discusses the limitations of the paper and future work. Finally, we conclude the paper in Section 9

2. Related Work

Many HAS solutions have been proposed for seamless video services in varying network conditions. Representative solutions are Adobe Systems HTTP Dynamic Streaming (Adobe HDS) [8], Apple HTTP Live Streaming (Apple HLS) [9], and Microsoft Smooth Streaming (Microsoft SS) [10]. The MPEG standardizes these proprietary solutions under the name of Dynamic Adaptive Streaming over HTTP (DASH) [2]. In DASH standards, a video server keeps multiple versions of the same video, encoded at different bitrates and quality levels, and splits the video into a set of multiple small segments. Before transmitting the video, the server provides DASH clients with a Media Presentation Description (MPD) manifest file. The MPD describes the video chunk information such as timing, language, timed text, and media characteristics such as video resolution and bitrate. Clients make the decision on which segment to download and sequentially request video chunks based on network conditions, device capabilities, and other factors [11].

Many studies have been conducted with respect to HAS. Seufert et al. [12] surveys the technical development of HAS, including both open standardized and proprietary solutions, focusing especially on users’ Quality of Experience (QoE). Among previous works, several papers are related to experimental evaluation of HAS players. The vast majority of them focused on mainly proprietary adaptive streaming systems [4–6]. Akhshabi et al. [4] evaluated two commercial HAS players (Netflix and Microsoft Sliverlight) and an open source player. They found significant inefficiencies with regards to the adaptation to changing network conditions in all of the evaluated clients. Mueller et al. [5] experimentally evaluated several HAS systems such as Apple HLS, Adobe HDS, and Microsoft SS. They compared their implementation of MPEG-DASH with these proprietary solutions in vehicular environments. Akhshabi et al. [6] investigated the problem of contention among players and revealed that the typical behavior of a HAS player in the steady state, including ON-OFF periods, can be the leading cause of the fairness problem in adaptive video streaming. Zabrovskiy et al. [13] introduce an adaptive video streaming evaluation framework to enable the automated testing of video players with various conditions. The focus of the framework is to provide tools for easy and rapid experimentation of players and algorithms. Stohr et al. [14] also provide a public execution environment for DASH players to enable a systematic comparison among them. These two works, [13,14], are similar to our approach in that they provide a systematic evaluation framework for adaptive streaming players, but they do not provide a source code level analysis of popular DASH players and a support for popular commercial streaming players.

Other major research focus for HAS has been on the adaptation algorithms. Jiang et al. [15] developed bitrate adaptation techniques that consider the tradeoffs among stability, fairness, and efficiency. Li et al. [16] showed that the discrete characteristics of the video bitrates make it hard for clients to perceive their fair shares of bandwidth and proposed an adaptation algorithm similar to TCP congestion control. Huang et al. [17] proposed a buffer-based adaptation algorithm in which the capacity estimation is used only during the startup phase. Yin et al. [18] proposed a model-predictive control algorithm that combines the throughput and buffer occupancy information. Wisniewski et al. [19] proposed an approach based on the estimated probability of video rebuffering, which is calculated by using an analytic model of playout buffer and characteristics of segment download time. Spiteri et al. [20] formulated the bitrate adaptation into a utility maximization problem and proposes an online control algorithm which uses the Lyapunov optimization technique for minimizing rebuffering and maximizing video quality.

There have been novel approaches on the adaptive streaming, especially focusing on different network environments. Seema et al. [21] proposed the wireless sensor compatible DASH (WVSNP-DASH) framework in which each video segment can be played without any reference to any file or segment to support light-weight video streaming for sensors. Detti et al. [22,23] presents an Information Centric Networking (ICN) based P2P live streaming application in which neighboring cellular devices can increase the quality of video playback with the help of other devices.

3. Experimental Framework

In this section, we present our testbed and the implementation details of the HAS players that we used for the evaluation.

3.1. Metrics and Setup Description

In our experiment settings, we obtain the variations of bitrate, the amount of buffers, and the throughput of each player. Additionally, we extract the delay of the video/player initialization process. From the data obtained, we calculate the performance metrics that significantly affect the QoE for HAS users.

- **Average Bitrate:** Since users normally prefer the video of higher quality, this metric is an important factor that can measure the satisfaction of users. However, notice that the utility to users does not increase proportionally as the bitrate increases [24].
- **Number of bitrate changes:** The frequency of bitrate changes adversely affects the users' satisfaction. Therefore, a video of lower bitrates with fewer bitrate changes may provide higher satisfaction to users than the video of higher bitrates with more bitrate changes [25,26].
- **Rebuffering time:** When the quantity of buffered video is not sufficient, the players stop the playback and resume after a sufficient amount of video is buffered. We estimate the total time in which the buffer length is less than one second as the rebuffering time. For players that we cannot directly obtain the buffer information, we estimate the buffer length by considering the reception of the segments and playout time of each segment. Let t be the current time, $B(t)$ be the estimated buffer length at time t. Also, let us assume that t_i is the video playout length of ith segment, s_j is the jth rebuffering time, and r_0 is the reception time of the first segment. We represent the number of segments received up to t as $N(t)$ and the number of rebufferings up to t as $M(t)$. Then the estimated buffer length $B(t)$ can be represented by

$$B(t) = \sum_{i=1}^{N(t)} t_i - \left(t - \sum_{j=1}^{M(t)} s_j - r_0 \right).$$

- **Startup delay:** The startup delay of video significantly affects the users' abandonment rate when playing the video [27]. Therefore, we obtain the startup delay by letting two users manually measure it while playing the video.

3.2. Goals / Objectives

Our experiments aim at finding the key characteristics and problems of HAS players in various network environments. We tested the players in different scenarios to see how HAS players adapt in (i) Standalone scenarios (Section 4), and (ii) Advanced scenarios (Section 5). We also provide a solution to an incorrect bandwidth estimation issue causing misbehavior of the rate adaptation by (iii) Enhancing the accuracy of DASH's bandwidth estimation (Section 6).

3.3. Experiment and Testbed Setup

Intensive work has been done in the HTTP adaptive streaming arena, but a number of components require additional investigation. First, many of the previous HAS evaluation works, e.g., [4–6], centered on the comparison of proprietary HTTP Adaptive Streaming players. On the other hand, our evaluation covers systems developed under the MPEG DASH standard. We consider today's state-of-the-art DASH solutions such as DASH Industry Forum Reference Client [28] (we call it DASH-IF player in short), Google's MPEG-DASH Media Source demo [29] (we call it DASH-Google player for short), and Bitmovin adaptive streaming demo player for MPEG-DASH [30] (we call it Bitmovin player for short)[1].

[1] The Bitmovin player in [30] is a demo implementation, and it does not reflect the adaptation logic of the actual commercial Bitmovin player.

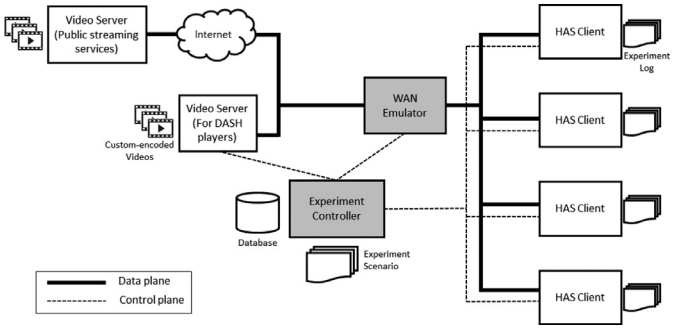


Fig. 1. Experimental framework. Dark gray components are added to the general HAS systems.

Second, previous works such as [4–6] do not conduct the experiments with enough repetition to identify tangible observations. In our work, we repeat each experimental setting with several runs to find the general characteristics of player behaviors. Third, earlier experimental works, e.g., [4–6], did not test the HAS players with various factors that can affect the performance. Our experiments consider not only various system parameters, but also realistic network environments, including background traffic, changing bandwidth, and coexistence of multiple HAS algorithms.

We designed our experiment framework as shown in Fig. 1. The testbed consists of a video server, an experiment controller, a WAN emulator, and several HAS clients. The experiment controller initiates an experiment by passing the experiment settings described in an experiment scenario to the WAN emulator and HAS clients. After the experiment is completed, the experiment controller collects the log files and stores them in the database, then displays the graphs with the experiment results. The WAN emulator provides a realistic network environment by reflecting factors such as the bandwidth, packet loss, and delay. It is implemented using *netem* [31] and Linux *tc* [32]. The video server resides on the testbed for 3 DASH players, i.e., DASH-IF, DASH-Google, and Bitmovin. On the other hand, the video servers of commerical streaming players we tested (i.e., Netflix [33], YouTube [34], Vimeo [35]) are located at the data center or CDN (content delivery network) for the service providers.

Table 1 shows the content informaton used for our evaluations. We used Big Buck Bunny video clip [36] for the source video in the case of 3 DASH players. The playout duration of the video source was 9 minutes 56 seconds, and the resolution was 1920x1080. We used Bitmovin's encoding service as guided in [37] to encode the video for DASH-IF and Bitmovin players. We encoded the video using an H.264 video codec, and into 5 bitrates, 400, 800, 1200, 2400, and 4800 Kbps, with 4-seconds segment durations for these two players. To generate the video files and MPD suitable for DASH-Google player, we used a free media encoder *ffmpeg* [38] and *DASH segmenter* (MP4Box-GPAC) [39]. The video was encoded using an H.264 video codec with 5 different bitrates, 522, 1200, 2100, 2500, and 4200 Kbps, and fragmented into chunks of 2 seconds. For the 3 commercial streaming players, we arbitrarily selected videos with a play duration longer than that of Big Buck Bunny video clip (1 hour 34 minutes for Netflix, 10 minutes 34 seconds for YouTube, and 10 minutes 36 seconds for Vimeo). The video for Vimeo was encoded with 7 different bitrates, 301, 357, 713, 1043, 2650, 5590, and 15052 Kbps, with 6-seconds segment durations. The video of YouTube was encoded with 8 different bitrates, 109, 245, 363, 709, 1347, 2383, 5729, and 10649 Kbps, while that of Netflix was encoded with 9 different bitrates, 100, 290, 370, 560, 870, 1270, 1780, 2610, and 3830 Kbps. For YouTube and Netflix, we couldn't find the exact segment duration since the video is encrypted. We ran the

Table 1
Content information used for our evaluations.

Media player	Segment duration (secs)	Play duration (mins:secs)	Codec	Source Video	Bitrates (Kbps)
DASH-IF	4	9:56	H.264	Big Buck Bunny	400,800,1200,2400,4800
DASH-Google	2	9:56	H.264	Big Buck Bunny	522,1200,2100,2500,4200
Bitmovin	4	9:56	H.264	Big Buck Bunny	400,800,1200,2400,4800
Vimeo	6	10:36	H.264	Arbitrarily selected	301,357,713,1043,2650,5590,15052
Netflix	Unknown	94:00	H.264/AVC and HEVC	Arbitrarily selected	100,290,370,560,870,1270,1780, 2610,3830
YouTube	Unknown	10:34	H.264/VP8	Arbitrarily selected	109,245,363,709,1347,2383, 5729,10649

experiments for 10 minutes for all players so that the experiment time fits the minimum play duration of the selected videos.

In order to automate the experiment, we used WebDriver [7] to systematically interact with HAS clients. With WebDriver, we can open a HAS video web page, obtain the values of specific elements on the page that contain information such as bitrate, buffer size, and click buttons to enable certain functions of the player. For Vimeo player, since it does not provide elements to display the information of bitrate or buffer length, we use speedprofile [40]. It opens the player web page, plays the video, and returns the information on the downloaded video segments such as URLs, size, and timing, which are used to obtain the bitrate and buffer length.

To calculate the throughput variation, we utilize *libnetfilter_queue* [41], a userspace library providing an API to access the packets queued by the kernel packet filter. We implement a program using this API that accesses every packet for the HTTP stream used for the video streaming; gets the time, size information of packets; and calculates the throughput variation of the video stream.

3.4. Tested HAS Players

In this section, we list the HAS players that we used for the experiments and explain how we obtained the information about the operations for players which require special measurement methods. For players whose source code is available, we present rate adaptation algorithms through a code level analysis. For all algorithms, we use the following common notations. i is the current segment number, and each algorithm decides the bitrate of $(i+1)$th video segment. N_i represents the bitrate of ith segment, while I_i represents the bitrate index of ith segment. bw_i is the measured bandwidth of ith transmitted video segment (by using transmission time and segment size). We assume that the number of encoded bitrates is m, and the encoding rate of index k is R_k. The encoding rate of larger index is higher than that of smaller index (for $1 \leq j < k \leq m$, $R_j < R_k$).

3.4.1. Open Source DASH Players

DASH-IF Player: Algorithm 1 is the pseudo-code for the bitrate adaptation algorithm of DASH Industry Forum Reference Client 2.3.0 [28]. The algorithm determines the bitrates using a combination of multiple rules. The throughput rule averages the throughputs of several downloaded segment samples and changes the bitrate to the maximum available if the buffer is sufficient. The insufficient buffer rule changes to the lowest bitrate if re-buffering happens and decreases the bitrate by increasing the differences if the buffer length is between 4 and 8 seconds. The buffer occupancy rule chooses the highest bitrate if the buffer length is larger than 20 seconds. The abandon requests rule tracks the throughput of each segment. If the estimated time for downloading the current segment is significantly larger than the segment duration and the remaining bytes are larger than the total estimated bytes of a candidate rate for switching, the algorithm abandons the download and switches to the candidate rate. The candidate rate is decided as the maximum bitrate that can be maintained, considering the throughput history. This method is to prevent buffer underflow in case of abrupt bandwidth drops. In addition, DASH Industry Forum Reference Client implements the adaptation algorithm presented in [20].

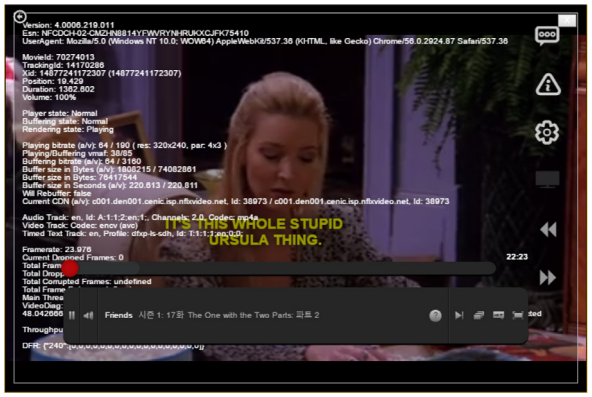


Fig. 2. Netflix player with a plug-in to display the detailed video information is shown.

DASH-Google Player: The adaptation algorithm of Google's MPEG-DASH Media Source demo [29] is shown in Algorithm 2 . It maintains two bandwidth estimation variables. It uses different exponential, moving average coefficients for these two variables to reflect small scale and large scale bandwidth variations. It chooses the minimum of the two when determining the bitrates.

Bitmovin Player: Algorithm 3 is the pseudo-code for the rate adaptation of Bitmovin player [30]. It consists of two switching methods: preferred startup switching and rate-based switching. The preferred startup switching overrides the rate suggested by other switching methods with a maximum rate that is equal to or smaller than the preferred startup rate if two conditions are met: the player is in the startup phase and the suggested rate is smaller than the preferred rate. The rate-base switching method selects the maximum rate that is smaller than the estimated bandwidth. The bandwidth is estimated by averaging the measured bandwidths of recently received segments with higher weights on more recent segments.

3.4.2. Commercial Streaming Players

Netflix Player: Netflix player [33] is one of the most popular video streaming players. Information such as bitrate selected, buffer length, and throughput is not directly obtainable from the player's user interface. Therefore, we use a Chrome extension program called SuperNetflix [42] (shown in Fig. 2), which provides detailed information about the video, playout status, and CDN where the video is downloaded. When playing the video, the information is not shown automatically. We use Selenium WebDriver [43] in order to find the web component that represents the button for opening the information panel on the browser and to click it programmatically. We also use Selenium WebDriver to obtain the information shown on the information panel, which is in the form of text, and parse it to obtain the performance metrics.

Algorithm 1: Bitrate adaptation algorithm of DASH Industry Forum Reference Client.

$n \leftarrow 3$ // the number of bandwidth estimation samples (3 for VOD)
$T_{low} \leftarrow 4$ // the threshold for deciding that the buffer length is low
$T_{rich} \leftarrow 20$ // the threshold for deciding that the buffer length is sufficient
$s \leftarrow 1$ // the step down factor for decreasing the bitrate when the buffer length is low
$B_{total} \leftarrow$ // the total bytes for segment i
$B_{cur} \leftarrow$ // the total received bytes up to now for segment i
$T_{elapsed} \leftarrow$ // the elapsed time from the download of first byte for segment i
$TH_{arr} \leftarrow$ // the throughput array used in abandon requests rule (in bps)
$l \leftarrow 5$ // the minimum length to average the throughput in Abandon requests rule
$T_{grace} \leftarrow 0.5$ // the grace time threshold used in abandon requests rule
$C_{aba} \leftarrow 1.8$ // the constant for the decision of abandoning the segment in abandon requests rule
$D \leftarrow$ // the segment duration

Throughput rule:
$sum \leftarrow 0$
for $j \leftarrow 0$ *to* $n-1$ **do**
  $sum \leftarrow sum + bw_{i-j}$
$Bandwidth \leftarrow sum/n$ // Estimated bandwidth
if $Buffer\_len >= T_{low} \cdot 2$ **then**
  for $k \leftarrow m$ *to* 1 **do**
    if $Bandwidth >= R_k$ **then**
      $N_{i+1} \leftarrow R_k$
      break

Insufficient buffer rule:
if *rebuffering* **then**
  $N_{i+1} \leftarrow R_1$
else if $T_{low} < Buffer\_len < T_{low} \cdot 2$ **then**
  $N_{i+1} \leftarrow R_{l_i - s}$
  $s \leftarrow s + 1$

Buffer occupancy rule:
if $Buffer\_len > T_{rich}$ **then**
  $N_{i+1} \leftarrow R_m$

Abandon requests rule (called multiple times for a segment):
add $B_{cur}/T_{elapsed}$ to TH_{arr}
if $length(TH_{arr}) > l$ *and* $T_{elapsed} > T_{grace}$ *and* $B_{cur} < B_{total}$ **then**
  $TH_{avg} \leftarrow$ average of TH_{arr}
  $T_{DownEst} \leftarrow B_{total} * 8/TH_{avg}$ // estimated download time
  if $T_{DownEst} < D * C_{aba}$ **then**
    return
  else
    $R_{new} \leftarrow$ the maximum bitrate that can be covered by TH_{avg}
    $B_{est} \leftarrow B_{total} * R_{new}/N_i$
    $B_{remain} \leftarrow B_{total} - B_{cur}$
    if $B_{remain} > B_{est}$ **then**
      abandon and switch to quality R_{new}

Algorithm 2: Bitrate adaptation algorithm of Google's MPEG-DASH Media Source demo.

$E_{slow} \leftarrow$ // the bandwidth estimation for slow channel variation
$E_{fast} \leftarrow$ // the bandwidth estimation for fast channel variation
$\alpha_{slow} \leftarrow 0.99$ // the exponential moving average coefficient for E_{slow}
$\alpha_{fast} \leftarrow 0.98$ // the exponential moving average coefficient for E_{fast}

Bandwidth estimation update:
while *video data is downloaded* **do**
  $E_{slow} \leftarrow \alpha_{slow} \cdot E_{slow} + (1 - \alpha_{slow}) \cdot download\_throughput$
  $E_{fast} \leftarrow \alpha_{fast} \cdot E_{fast} + (1 - \alpha_{fast}) \cdot download\_throughput$
Bitrate decision:
$Bandwidth \leftarrow \min(E_{slow}, E_{fast})$ // Estimated bandwidth
for $k \leftarrow m$ *to* 1 **do**
  if $Bandwidth >= R_k$ **then**
    $N_{i+1} \leftarrow R_k$
    break

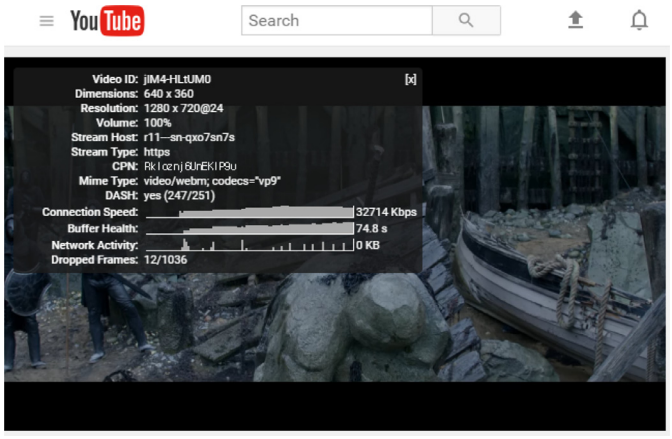


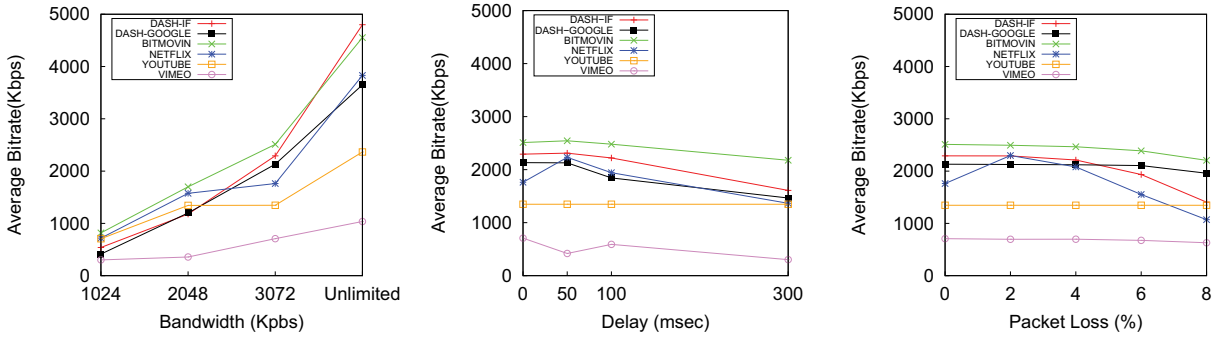
Fig. 3. YouTube player with an option to display the detailed video information enabled is shown.

YouTube Player: The YouTube player [34] provides a function to display detailed video playback information called 'Stats for nerds' (shown in Fig. 3). By default, this function is not enabled. Therefore, we use Selenium WebDriver to find the button on the web page to enable it and click it programmatically as is done for the Netflix player. We also parse the text in the information panel to obtain the performance metrics.

Vimeo Player: We found that Vimeo player [35] uses different values in the specific part of the URL for different bitrates. Before the experiments, we play the video with different fixed bitrates and obtain the mapping between the values of URL parts representing bitrate and corresponding bitrates. During experiments, we use this mapping to figure out the actual selected bitrates. For buffer length estimation, we use the method explained in Section 3.1.

4. Standalone Scenarios

In this section, we evaluate the performance of HAS players described in Section 3.4 by running each video player in different network settings.



(a) Average bitrate versus bandwidth. (b) Average bitrate versus delay. (c) Average bitrate versus packet loss.

Fig. 4. The average bitrates for 6 players are compared in different bandwidth, delay, and packet loss conditions. The default bandwidth, delay, packet loss rate are 3 Mbps, 0 msec, 0%. We vary only bandwidth in (a), delay in (b), loss rate in (c).

Algorithm 3: Bitrate adaptation algorithm of Bitmovin player.

Preferred startup switching:

$t \leftarrow$ // the time passed from start
$s \leftarrow$ // the index of the rate suggested by other switching methods
$R_{pre} \leftarrow$ // the preferred startup rate
if $t < 10$ **then**
  for $k \leftarrow m$ *to* 1 **do**
    if $k = s$ **then**
      return $N_{i+1} \leftarrow R_s$
    if $R_k <= R_{pre}$ **then**
      return $N_{i+1} \leftarrow R_k$
  return $N_{i+1} \leftarrow R_m$
else
  return $N_{i+1} \leftarrow R_s$

Rate based switching:

// Bandwidth estimation update
$depth \leftarrow$ // the number of downloaded segments to use for bandwidth estimation
$N_{buf} \leftarrow$ // the buffer size in segment numbers
$sum \leftarrow 0$
for $j \leftarrow 0$ *to* $depth - 1$ **do**
  $sum \leftarrow sum + bw_{i-j} \cdot (1 - j/N_{buf})$
$Bandwidth \leftarrow sum/depth$ // Estimated bandwidth
// Bitrate decision
$R_{min} \leftarrow Infinity$
$R_{max} \leftarrow 0$
for *all encoding rate index* k **do**
  if $R_{max} < R_k$ *and* $R_k < Bandwidth$ **then**
    $R_{max} \leftarrow R_k$
  if $R_{min} > R_k$ **then**
    $R_{min} \leftarrow R_k$
if $R_{max} > 0$ **then**
  $N_{i+1} \leftarrow R_{max}$
else
  $N_{i+1} \leftarrow R_{min}$

4.1. Experimental Settings

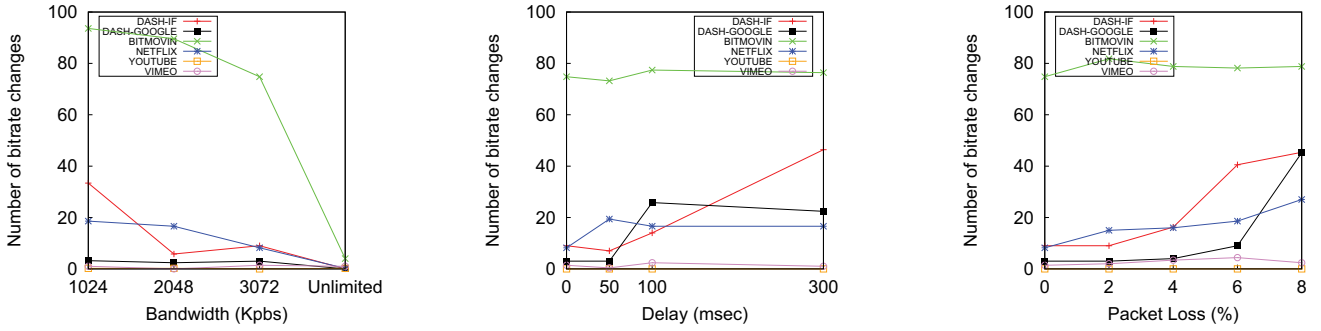
We run one player of each algorithm and change the experimental settings as follows for the evaluation in various network circumstances. We vary the network bandwidth from unlimited to limited bandwidth with various values (1024, 2048, and 3072 Kbps). We also vary the network round trip time (0, 50, 100, and 300 msec) and packet loss rate (0, 2, 4, 6, and 8 %). We repeat every experiment 5 times and average the performance metrics.

4.2. Results

Impact of bandwidth, delay, and packet loss on average bitrate: The average bitrates of the six players are compared at different bandwidths in Fig. 4a. As expected, the average bitrate increases as the network bandwidth increases. However, we observed some players closely follow the assigned bandwidth, e.g., Bitmovin, but some players conservatively adapt to the bandwidth, e.g., Vimeo. By looking into the internal variables of the players, we also find that the bandwidth estimation variables and the bitrates selected by DASH-IF and DASH-Google clients sometimes become very large. We presume that this over-estimation of the bandwidth is due to the buffer effect. Since the player estimates the bandwidth by looking at the retrieval speed of video data, if the data retrieval occurs in a batch, the client overestimates the bandwidth. We discuss this issue in Section 6 in more detail.

The average bitrates of the six clients are compared at different network delays and packet losses in Fig. 4 b and 4 c. As the delay increases, the average bitrate deceases in most players, except several exceptional cases in Netflix and Vimeo players. YouTube does not adapt to different RTTs. We presume that the insensitiveness of the YouTube player to the variation of RTT is due to the characteristics of the transport protocol used by the Chrome browser for YouTube service, i.e., QUIC, different from the normal TCP algorithm. QUIC [44,45] is a transport protocol designed by Google to improve performance of HTTPS. It replaces most of the traditional HTTPS protocol stack and is developed as a user-space transport based on UDP. Even though QUIC has its own rate control algorithm, it is presumed that the algorithm is not so sensitive to the network variation and other competing flows in the network. We conducted various experiments that show the behavior of QUIC in more detail in Section 5. The relation of the average bitrate and packet loss rate is as expected: as the packet loss rate increases, the average bitrate decreases. Again, the YouTube player is not adaptive to the loss rate, unlike other players.

Impact of bandwidth, delay, packet loss on number of bitrate changes: We look into how the bandwidth, delay, and packet loss affect the number of bitrate changes in Fig. 5a,5b, and 5c, respectively. In DASH-IF, Bitmovin, and Netflix players, the number of bitrate changes decreases as the bandwidth increases. We presume that this is because there are more available video bitrate options at lower bandwidth. We can also observe that the number of bi-

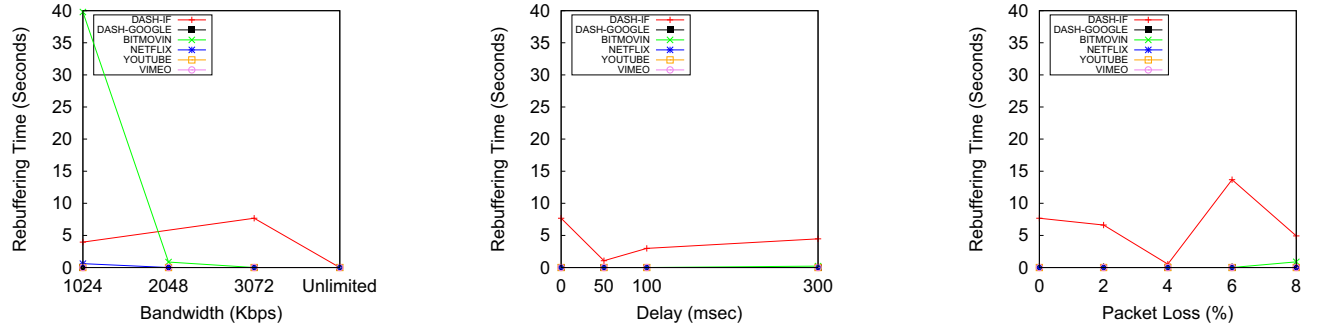


(a) The number of bitrate changes versus bandwidth.

(b) The number of bitrate changes versus delay.

(c) The number of bitrate changes versus packet loss.

Fig. 5. The number of bitrate changes for 6 players is compared at different bandwidth, network delay, and packet loss conditions. The default bandwidth, delay, packet loss rate are 3 Mbps, 0 msec, 0%. We vary only bandwidth in (a), delay in (b), loss rate in (c).



(a) The rebuffering times versus bandwidth.

(b) The rebuffering times versus delay.

(c) The rebuffering times versus packet loss.

Fig. 6. The rebuffering times for 6 players are compared at different bandwidth, delay, and packet loss conditions. The default bandwidth, delay, packet loss rate are 3 Mbps, 0 msec, 0%. We vary only bandwidth in (a), delay in (b), loss rate in (c).

trate changes of Bitmovin is exceptionally higher than other players. We presume the algorithms of the Bitmovin player that use a simplified bandwidth estimation approach without considering the buffer status are related to this unstable behavior. For DASH-Google, YouTube, and Vimeo, the number of bitrate changes are marginal regardless of the bandwidth. We cannot observe a definite relation between the number of bitrate changes and the delay. The relation of the number of bitrate changes and the packet loss is clear in DASH-IF, DASH-Google, and Netflix players. However, other players, including Bitmovin, YouTube, and Vimeo players, are rather insensitive to the packet loss rate.

Impact of bandwidth, delay, and packet loss on rebuffering time: The relations between the rebuffering time and the bandwidth, delay, and packet loss are shown in Fig. 6a,6 b, and 6 c, respectively. Roughly, the rebuffering time increases as the bandwidth decreases and as the network delay increases for DASH-IF and Bitmovin players, but there are lots of exceptions. Other players do not exhibit rebuffering except the case of 1024 Kbps bandwidth with the Netflix player. From Fig. 6c showing that most of players exhibit no rebuffering time in different packet loss rates, we can presume that any algorithm caring for the buffer status has minimized the rebuffering time and that the HAS reaction is effective in reducing the rebuffering time in different channel conditions.

Number of bitrate changes versus rebuffering time: The relationship between the number of bitrate changes and the rebuffering time is shown in Fig. 7. We cannot observe a definite relation between them for all players, but we can see that DASH-IF and Bitmovin players take a large range in both metrics compared to other players.

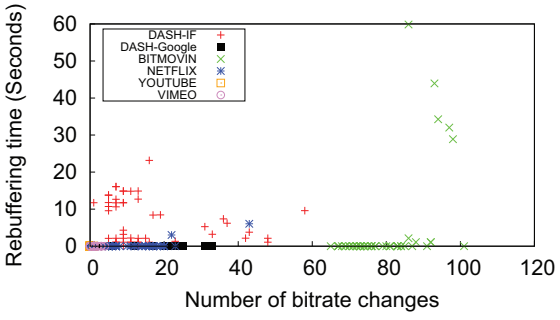


Fig. 7. The relation between the number of bitrate changes and the rebuffering time is shown.

Table 2
Startup delay measured by two users (unit: seconds). We set the bandwidth, network delay, loss rate to 3 Mbps, 0 msec, 0%.

	User 1	User 2
DASH-IF	2.18	2.24
DASH-Google	4.95	4.80
Bitmovin	3.38	3.46
Netflix	7.76	7.68
YouTube	1.75	1.76
Vimeo	1.54	1.47

Startup delay: Table 2 shows the startup delay measured by two users for different HAS players with 3072 Kbps bandwidth and no RTT/packet loss setting. We repeat the measurement 20 times

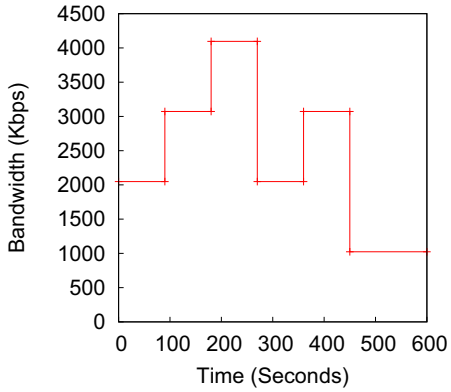


Fig. 8. The variation of bandwidth set for the dynamic bandwidth scenario is shown.

for each player and average the results. We can observe that the values are highly correlated between users, and different players show significantly different values: while Vimeo has a startup delay around 1.5 seconds, that of Netflix is over 7.5 seconds. We presume this variation is due to the difference in the main service targets of players. The Netflix player maintains long-term stability of the video playout by having enough buffered video before it starts to play because it mostly services videos with long play time. On the other hand, Vimeo player puts higher weight on quick start of play since it mostly services videos with short play time. Other players come between these two extremes.

5. Advanced Scenarios

In this section, we test three advanced experiment scenarios: dynamic bandwidth, background traffic, and multiple players scenarios.

5.1. Experimental Settings

For the dynamic bandwidth scenario, we set the bandwidth as shown in Fig. 8. We change the bandwidth every 90 seconds to see how HAS players adapt to a bandwidth variation. For the background traffic scenario, we set the background TCP traffic using the *iperf* traffic generator as shown in Fig. 10. This setting is to test how HAS players operate when the number of background TCP flows is varied between 1, 2, and 3. In multiple players scenario, we observe how players operate in a scenario where multiple adaptive streaming players share a network bottleneck and compete for available bandwidth, as in [6] with up-to-date standard-based players and commercial streaming players. We test two types of scenarios: the homogeneous scenario in which every client node runs the same HAS players and the heterogeneous scenario in which every client node runs different HAS players. We run the experiments for each scenario five times and present the average values for the performance metrics.

5.2. Impact of Dynamic Bandwidth

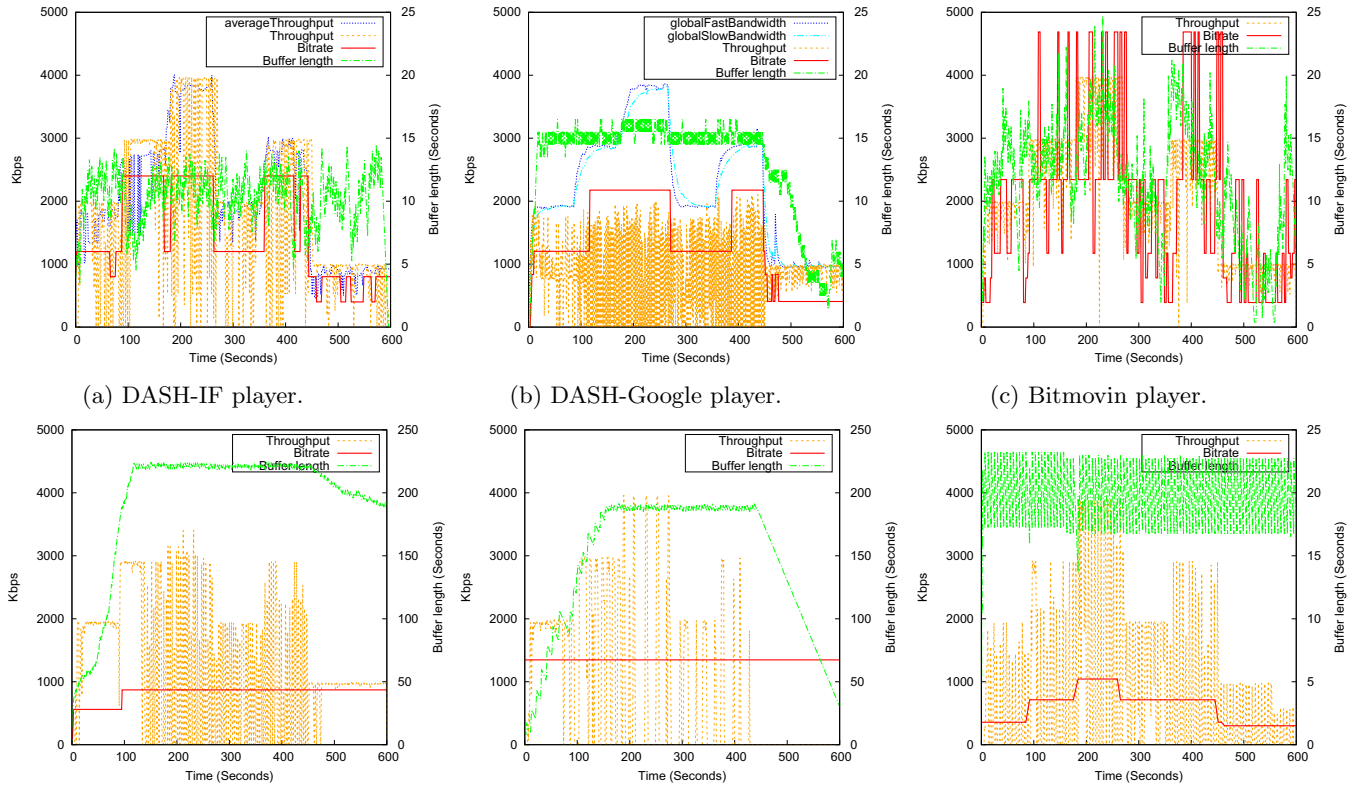
The result of dynamic bandwidth scenario is shown in Fig. 9. DASH-IF client switches between high bitrates and low bitrates frequently, but it follows the bandwidth variation roughly in bitrate selection (Fig. 9a). It maintained a stable buffer length in this experiment but exhibited rebuffering times of 31.0 and 35.8 seconds in 2 of the 5 experiment runs. DASH-Google client (Fig. 9b) very closely estimates the network bandwidth to the bandwidth set for the experiment. However, it sometimes overestimates the bandwidth due to the buffer effect, e.g., during the period between 460 and 480 seconds, and alternates between different bitrates within a short period of time. Despite the overestimation of the bandwidth, DASH-Google player did not trigger a rebuffering. Bitmovin player (Fig. 9c) follows the bandwidth variation most closely among all players, but it changes the bitrate very frequently due to the buffer effect in bandwidth estimation. Bitmovin player shows a rebuffering time of 10.7 seconds and exhibits a rebuffering time larger than zero in all the other 4 experiment runs. Netflix player (Fig. 9d) is somewhat insensitive to the bandwidth variation. It does not change the bitrate according to the bandwidth change, except during the initial phase. We cannot know the detailed reason for this since the source code of the adaptation algorithm is not accessible, but we presume that Netflix player prioritizes the stability of the bitrate selection over the adaptability to the bandwidth variation. Netflix player also maintains a much higher level of buffer length than other players, excluding YouTube, which shows a similar level of buffer length. YouTube player (Fig. 9e) is more insensitive to the bandwidth variation than Netflix player. In addition to the prioritization on stability in bitrate selection of Netflix player, we found that YouTube player considers the resolution of the screen and does not select bitrates higher than the bitrate that the screen can display. Vimeo player (Fig. 9f) adapts to the bandwidth variation very stably. However, this player chooses the bitrate in an excessively conservative way: it chooses a much lower bitrate than the available bandwidth. As a result, the buffer length is stable throughout the duration of the experiment.

5.3. Impact of Background Traffic

The result of background traffic scenario is illustrated in Fig. 11. Similar to the result of dynamic bandwidth scenario, DASH-IF player switches between high bitrates and low bitrates frequently (Fig. 11a). It shows a rebuffering time of 19.0 seconds during the period from 535 to 570 seconds due to an excessively large bitrate selection. DASH-Google client (Fig. 11b) actively selects a bitrate to adapt to the varying throughput due to the different number of flows as times goes. It triggers rebuffering for 2.1 seconds. Bitmovin player (Fig. 11c) changes the bitrate most frequently, as in dynamic bandwidth scenario. In addition, its buffer is not stable and exhibits a rebuffering time of 19.4 seconds. Netflix player (Fig. 11d) maintains a relatively large bitrate and stable buffer length. However, it takes a long time (around 100 seconds) to reach a stable state in the initial phase. YouTube player (Fig. 11e), again, exhibits invariant bitrate selection and stable buffer length as in dynamic bandwidth scenario. Vimeo player (Fig. 11f) adapts to the bandwidth variation very stably as in dynamic bandwidth scenario. However, it also chooses the bitrate very conservatively and, as a result, maintains stable buffer load throughout the duration of the experiment.

5.4. Impact of Multiple Players

The results of the multiple homogeneous players scenarios for 6 tested players are shown in Table 3. The average bitrates of DASH-Google and YouTube are the largest, while that of Vimeo is the smallest. We presume the different sets of bitrates available for each player affected this result. However, as pointed out in Section 4, the QUIC protocol used by YouTube and the adaptation algorithm of DASH-Google, which follows the available bandwidth closely, are expected to have affected the high average bitrate. The Vimeo player's conservative bitrate selection is confirmed in this experiment again. Overall, the number of bitrate changes and rebuffering times are larger than previous experiments. Even the players that showed the most stable behaviors, such as Netflix, YouTube, and Vimeo players, change bitrate much more than in



(a) DASH-IF player. (b) DASH-Google player. (c) Bitmovin player.

(d) Netflix player. (e) YouTube player. (f) Vimeo player.

Fig. 9. Results of dynamic bandwidth scenario. We illustrate the variations of throughput, bitrate, buffer length, and other internal variables according to the player. We vary only the bandwidth according to Fig. 8, and fix the network delay and packet loss rate to 0 msec, 0%, respectively.

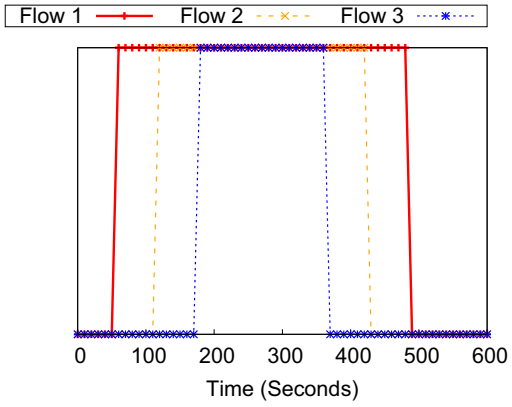


Fig. 10. The initiation and termination times of 3 background traffic for the background traffic scenario is shown.

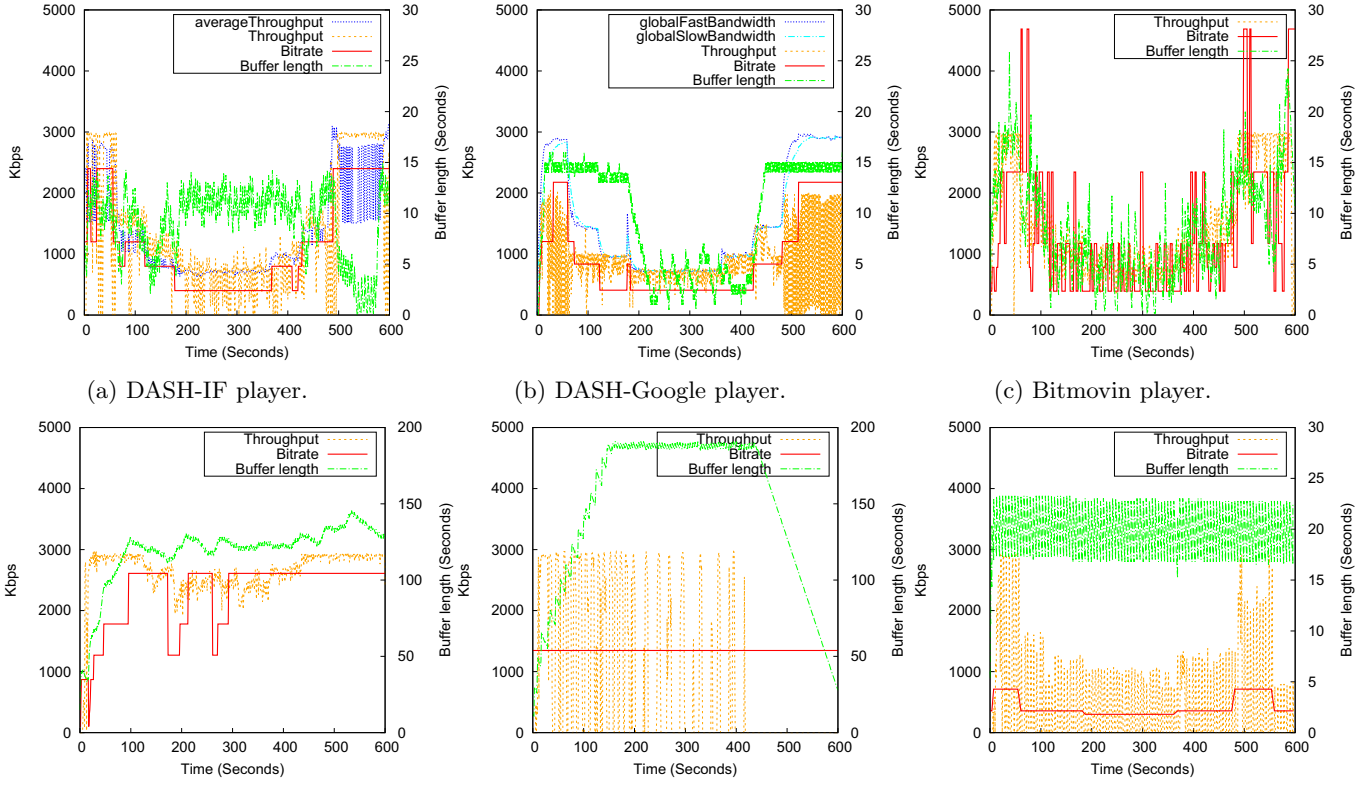
previous experiments and sometimes trigger rebuffering. We presume that the distinct traffic characteristics of HAS players, such as repeated ON-OFF periods, make the available bandwidth for each player variable and difficult to determine a stable bitrate in comparison to previous scenarios (which include only one HAS player for each experiment).

Finally, the results of the multiple heterogeneous players scenarios with three DASH standard-based clients of DASH-IF, DASH-Google, Bitmovin, and three commercial streaming players of Netflix, YouTube, and Vimeo are also shown in Table 3. We can see that the different adaptation algorithms of different players result in significant differences in each performance metric. DASH-Google player chooses the lowest bitrate but experiences low number of bitrate changes and no rebuffering. Bitmovin player takes the highest bitrate among three contending players, but it experiences the highest number of bitrate changes and also long rebuffering times. YouTube player selects the highest bitrate among three contending players, but it does not experience bitrate change or rebuffering. From this, we can observe that QUIC protocol is very aggressive when contending with other TCP flows and not responsive to congestion as much as other TCP flows. Vimeo player experiences a long rebuffering time and large number of bitrate changes despite its low bitrate selection.

6. Enhancing the Accuracy of DASH's Bandwidth Estimation

As we discovered from the experiments in Section 4, the bandwidth estimation algorithms of the HAS clients sometimes overestimate the available bandwidth. This overestimation, in turn, results in the selection of much higher bitrate than the bandwidth. To solve this vicious cycle in an efficient way, we devised a statistical method that eliminates the throughput information that is affected by the buffer effect as in Algorithm 4 . By not considering the outliers from the throughput data in the calculation of the bandwidth available, the method predicts the bandwidth more precisely.

To check whether the proposed algorithm mitigates the buffer effect, we test the original DASH-Google client and a modified DASH-Google client containing our algorithm with the 1024 Kbps bandwidth limitation. The results are shown in Figs. 12 and 13. As shown in Fig. 12, the bandwidth estimation values (globalFastBandwidth and globalSlowBandwidth) of original DASH-Google player sometimes become significantly large due to buffer effect, e.g., at 373, 457, and 587 seconds. The throughput sample values are so large that even if a moving average is used for the calculation



(a) DASH-IF player. (b) DASH-Google player. (c) Bitmovin player.

(d) Netflix player. (e) YouTube player. (f) Vimeo player.

Fig. 11. Results of the background traffic scenario. We illustrate the variations of throughput, bitrate, buffer length, and other internal variables according to the player. We set the bandwidth, network delay and packet loss rate to 3 Mbps, 0 msec, 0%, respectively.

Algorithm 4: Bandwidth estimation method for eliminating the buffer effect.

$S_i \leftarrow$ the moving average for the deviation of the throughput from the bandwidth estimation in ith iteration
$B_i \leftarrow$ the bandwidth estimation of ith iteration
$\alpha \leftarrow$ the exponential moving average coefficient for bandwidth estimation
$\beta \leftarrow$ the exponential moving average coefficient for throughput deviation S_i
$k \leftarrow$ the parameter for giving tolerance to the variation of the bandwidth
$bw_i \leftarrow$ the bandwidth sample in iteration i
$n \leftarrow$ the number of bandwidth samples that exceed the variation range consecutively
$\gamma \leftarrow$ the number of bandwidth samples that we consider as actual bandwidth increase if they exceeds the variation range consecutively
for *each iteration* i **do**
  if $bw_i \geq B_{i-1} + k \cdot S_{i-1}$ **then**
    $n = n + 1$
  else
    $n = 0$
  if $bw_i < B_{i-1} + k \cdot S_{i-1}$ *or* $n \geq \gamma$ **then**
    $B_i = \alpha \cdot B_{i-1} + (1 - \alpha) \cdot bw_i$
    $S_i = \beta \cdot S_{i-1} + (1 - \beta)|B_i - bw_i|$
  else
    $B_i = B_{i-1}$
    $S_i = S_{i-1}$

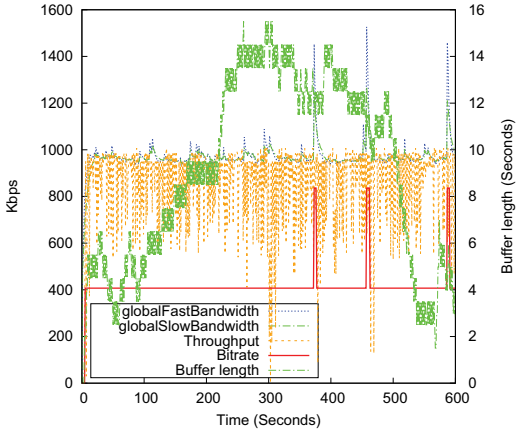


Fig. 12. The variations of throughput, bandwidth estimation variables, buffer amount, and bitrate for original DASH-Google player with 1024 Kbps bandwidth limitation are shown. The network delay, packet loss rate are set to 0 msec, 0%.

of bandwidth estimation variables, the bandwidth estimation is severely distorted. Due to this phenomenon, the bitrates are selected incorrectly; much higher bitrates than the available bandwidth are selected, and the bitrates vary wildly. On the other hand, in the case of modified DASH-Google (Fig. 13), the estimated bandwidth values are not affected by instant increases of the throughput sample value and stably approach the assigned bandwidth. This results in the stable bitrate selection. To further inspect the effect of Algorithm 4, we repeated the dynamic bandwidth scenario in Section 5.2 with the modified DASH-Google client. We observed that the modified DASH-Google client selects the bitrates closely to

Table 3
Results of multiple player scenarios. The bandwidth is set to 4 Mbps, and network delay and packet loss rate are set to 0 msec and 0% respectively.

Scenario Type	Client Number	Average bitrate (Kbps)	Number of bitrate changes	Average buffer length	Rebuffering time (Seconds)
Homogeneous	1 (DASH-IF)	756.81	43.2	8.46	18.09
	2 (DASH-IF)	767.05	45.0	8.58	10.99
	3 (DASH-IF)	772.95	41.0	8.38	19.88
	4 (DASH-IF)	764.96	45.6	8.77	8.64
	Total average	765.44	43.7	8.55	14.40
Homogeneous	1 (DASH-Google)	1059.57	31.6	13.22	0
	2 (DASH-Google)	1230.61	28.6	13.79	0
	3 (DASH-Google)	1262.36	22.4	13.91	0
	4 (DASH-Google)	1101.27	29.8	13.48	0
	Total average	1163.45	28.1	13.60	**0**
Homogeneous	1 (Bitmovin)	880.50	93.2	6.26	60.00
	2 (Bitmovin)	835.97	93.4	7.60	45.68
	3 (Bitmovin)	895.43	91.0	5.78	98.02
	4 (Bitmovin)	913.78	96.2	6.94	29.14
	Total average	881.42	93.4	6.65	58.21
Homogeneous	1 (Netflix)	665.80	26.8	113.14	0
	2 (Netflix)	676.64	19.6	135.83	0
	3 (Netflix)	710.42	20.4	128.39	0.60
	4 (Netflix)	716.20	15.0	146.26	0
	Total average	692.27	20.4	130.90	0.15
Homogeneous	1 (YouTube)	1004.50	6.0	93.91	37.16
	2 (YouTube)	1055.88	5.6	95.25	20.17
	3 (YouTube)	1275.00	0.8	128.12	0
	4 (YouTube)	1154.57	4.8	100.91	22.63
	Total average	1122.49	4.3	104.55	19.99
Homogeneous	1 (Vimeo)	496.20	21.0	17.83	12.54
	2 (Vimeo)	460.29	15.4	18.60	0
	3 (Vimeo)	441.81	22.4	18.07	0
	4 (Vimeo)	364.83	15.8	13.26	16.80
	Total average	440.79	18.6	16.94	7.33
Heterogeneous	1 (DASH-IF)	1168.08	12.8	9.43	4.89
	2 (DASH-Google)	864.61	20.4	13.93	0
	3 (Bitmovin)	1521.99	87.0	9.93	4.34
	Total average	1184.90	40.0	11.09	3.07
Heterogeneous	1 (Netflix)	1206.00	16.2	124.76	0
	2 (YouTube)	1347.00	0	148.72	0
	3 (Vimeo)	543.04	26.4	15.83	27.71
	Total average	1032.01	14.2	96.44	9.23

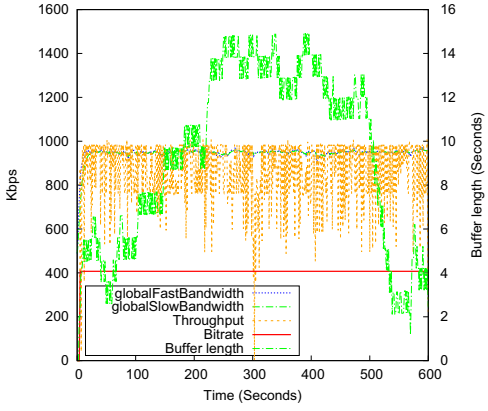


Fig. 13. The variations of throughput, bandwidth estimation variables, buffer amount, and bitrate for modified DASH-Google player with 1024 Kbps bandwidth limitation are shown. The network delay, packet loss rate are set to 0 msec, 0%.

the bandwidth variation without over-estimation of the bandwidth during the time period between around 460 and 475 seconds (results are not shown to save the space).

7. Key Observations and Suggestions

- **Customization of rate adaptation according to the service target:** Through various experiments shown in previous sections, we observed there are significant differences among players in terms of bitrate adaptation. They differ in target buffer length, adaptability to the bandwidth variations, fairness among flows, startup delay, etc. One of the factors that affects these differences is the major target of the service. The commercial streaming players tend to change the bitrate more conservatively than general DASH standard-based players. The players that service mostly short videos, such as Vimeo, tend to maintain shorter buffer length than the players that service mostly long videos, such as Netflix. We will be able to obtain higher user satisfaction if we customize the bitrate adaptation according to the users' preferences or video type/length.
- **Bandwidth estimation:** In Section 4, we observe that the bandwidth estimation is highly inaccurate in some scenarios due to the buffer effect. We may utilize a cross-layer approach in which the HAS player can access the transport layer for throughput information to obtain better bandwidth estimation (e.g., using *TCP_INFO* socket information, which allows applications to access TCP connection information such as transmitted bytes, received bytes, TCP state, and congestion window). We can also use a statistical method in which the throughput values affected by the buffer effect are eliminated from the calculation of the bandwidth estimation, as we suggested in Section 6.
- **Fairness:** The TCP mechanism provides the throughput fairness among competing flows. However, the bitrate selections are unfair in many competition scenarios even with the homogeneous algorithms, due to the native characteristics of HAS (such as the ON-OFF periods pointed out in [6]). The UDP-based transport of YouTube player has a negative impact on fairness. During the competition with Netflix and Vimeo players, YouTube

player obtains the highest constant bitrate, and the other players suffer from low, variable bitrates, or rebuffering (especially the Vimeo player). To solve this problem, we suggest deriving a basic, minimal mechanism for maintaining the fairness among different players and applying it to every player (e.g., the maximum bandwidth increase rate in one step, minimum time for a bitrate change—abrupt increase in rate or frequent bitrate change makes other players hard to estimate the bandwidth available).

8. Limitations and Future Work

In our experiments, we vary the network environment in various ways but confine our experiment environment to the wired network rather than the mobile network. Our focus is on investigating how players behave in different settings within the wired network. Experiments with other HAS players in mobile networks were performed in [46–49], and [50]. We leave the experiment in mobile network with our tested players to our future work. In addition to the mobile network environment, there are various factors that can affect the operation of HAS players, including the video content, video encoding method such as bitrate set, codec, segment duration, variable bit rate/constant bit rate, and video rate time-variability. Some of these issues were addressed in previous work [48,51,52], but we leave thorough investigation of these issues with up-to-date DASH based players and popular streaming service players as our future work.

9. Conclusion

In this paper, we perform an in-depth evaluation of practical streaming players developed under the MPEG DASH standard and popular commercial streaming players. Specifically, we provide the detailed operation of rate adaptation algorithms in several HAS/DASH clients, using code level analysis. In addition, we conduct extensive experiments of these clients on our testbed and make several observations on the performances of 6 representative DASH-based players and popular commercial streaming players. They include significantly different behaviors in bitrate adaptation due to differences in the type of services (e.g., commercial players tend to change the bitrate more conservatively, and the players serving mostly short videos tend to maintain a small buffer length) and the non-optimal bit-rate selection due to frequent over-estimation of the bandwidth. In particular, we identified the cause of one of the player mis-behaviors, which we termed “buffer effect,” and provided a statistical method to mitigate the effect. Based on the observations, we suggested several guidelines for improving the rate adaptation, bandwidth estimation, and fairness of HAS/DASH players.

Acknowledgments

This work was partially supported by NSF grant CNS-1525435. Special thanks go to Shyam Sundar Ramamoorthy and Brett Shouse for their help in the implementation of the testbed and many helpful discussions and comments on the paper.

References

[1] Cisco, Cisco visual networking index: Forecast and methodology, 2014–2019, CISCO White paper (2015).
[2] MPEG DASH standard., http://dashif.org/mpeg-dash.
[3] I. Sodagar, The mpeg-dash standard for multimedia streaming over the internet, IEEE MultiMedia (4) (2011) 62–67.
[4] S. Akhshabi, A.C. Begen, C. Dovrolis, An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http, in: Proceedings of the second annual ACM conference on Multimedia systems, ACM, 2011, pp. 157–168.
[5] C. Müller, S. Lederer, C. Timmerer, An evaluation of dynamic adaptive streaming over http in vehicular environments, in: Proceedings of the 4th Workshop on Mobile Video, ACM, 2012, pp. 37–42.
[6] S. Akhshabi, L. Anantakrishnan, A.C. Begen, C. Dovrolis, What happens when http adaptive streaming players compete for bandwidth? in: Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video, ACM, 2012, pp. 9–14.
[7] WebDriver., https://www.w3.org/TR/webdriver/.
[8] Adobe HTTP Dynamic Streaming., http://www.adobe.com/products/hds-dynamic-streaming.html.
[9] Apple HTTP Live Streaming., https://developer.apple.com/streaming.
[10] Microsoft Smooth Streaming., http://www.iis.net/downloads/microsoft/smooth-streaming.
[11] Wowza, Understanding streaming protocols and output file formats, 2014 (accessed August 11, 2014), (http://www.wowza.com/forums/content.php?621-Understanding-streaming-protocols-andoutput-file-formats).
[12] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hobfeld, P. Tran-Gia, A survey on quality of experience of http adaptive streaming, IEEE Communications Surveys & Tutorials 17 (1) (2015) 469–492.
[13] A. Zabrovskiy, E. Kuzmin, E. Petrov, C. Timmerer, C. Mueller, Advise: Adaptive video streaming evaluation framework for the automated testing of media players, in: Proceedings of the 8th ACM on Multimedia Systems Conference, ACM, 2017, pp. 217–220.
[14] D. Stohr, A. Frömmgen, A. Rizk, Where are the sweet spots? a systematic approach to reproducible dash player comparisons (2017).
[15] J. Jiang, V. Sekar, H. Zhang, Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive, in: ACM CoNEXT, 2012.
[16] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A.C. Begen, D. Oran, Probe and adapt: Rate adaptation for http video streaming at scale, Selected Areas in Communications, IEEE Journal on 32 (4) (2014) 719–733.
[17] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, M. Watson, A buffer-based approach to rate adaptation: Evidence from a large video streaming service, in: Proceedings of the 2014 ACM conference on SIGCOMM, ACM, 2014, pp. 187–198.
[18] X. Yin, A. Jindal, V. Sekar, B. Sinopoli, A control-theoretic approach for dynamic adaptive video streaming over http, in: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, in: SIGCOMM '15, ACM, New York, NY, USA, 2015, pp. 325–338, doi:10.1145/2785956.2787486.
[19] P. Wisniewski, A. Beben, J.M. Batalla, P. Krawiec, On delimiting video rebuffering for stream-switching adaptive applications, in: 2015 IEEE International Conference on Communications (ICC), IEEE, 2015, pp. 6867–6873.
[20] K. Spiteri, R. Urgaonkar, R.K. Sitaraman, Bola: Near-optimal bitrate adaptation for online videos, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, 2016, pp. 1–9, doi:10.1109/INFOCOM.2016.7524428.
[21] A. Seema, L. Schwoebel, T. Shah, J. Morgan, M. Reisslein, Wvsnp-dash: Name-based segmented video streaming, IEEE Transactions on Broadcasting 61 (3) (2015) 346–355.
[22] A. Detti, B. Ricci, N. Blefari-Melazzi, Supporting mobile applications with information centric networking: the case of p2plive adaptive video streaming, in: Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking, ACM, 2013a, pp. 35–36.
[23] A. Detti, B. Ricci, N. Blefari-Melazzi, Peer-to-peer live adaptive video streaming for information centric cellular networks, in: Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on, IEEE, 2013b, pp. 3583–3588.
[24] W. Song, D. Tjondronegoro, M. Docherty, Saving bitrate vs. pleasing users: where is the break-even point in mobile video quality? in: Proceedings of the 19th ACM international conference on Multimedia, ACM, 2011, pp. 403–412.
[25] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, H. Zhang, A quest for an internet video quality-of-experience metric, in: Proceedings of the 11th ACM workshop on hot topics in networks, ACM, 2012, pp. 97–102.
[26] N. Cranley, P. Perry, L. Murphy, User perception of adapting video quality, International Journal of Human-Computer Studies 64 (8) (2006) 637–647.
[27] S.S. Krishnan, R.K. Sitaraman, Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs, IEEE/ACM Transactions on Networking 21 (6) (2013) 2001–2014.
[28] DASH-264 JavaScript Reference Client., http://dashif.org/reference/players/javascript/index.html.
[29] MPEG-DASH / Media Source demo., http://dash-mse-test.appspot.com/.
[30] Bitmovin Adaptive Streaming Player for MPEG-DASH & HLS., https://github.com/bitmovin/bitdash-demo.
[31] netem, (accessed November 19, 2015), (http://www.linuxfoundation.org/collaborate/workgroups/networking/netem).
[32] tc - show / manipulate traffic control settings, (accessed November 19, 2015), (http://linux.die.net/man/8/tc).
[33] Netflix - Watch TV Shows Online, Watch Movies Online., https://www.netflix.com/.
[34] YouTube., https://www.youtube.com/.
[35] Vimeo: Watch, upload and share HD and 4k videos with no ad., https://vimeo.com/.
[36] Microsoft, Bigbcukbunny, 2008 (accessed December, 2014), (http://bigbuckbunny.org/index.php/download).
[37] Setup Adaptive Bitrate Streaming with DASH and HLS., https://bitmovin.com/tutorials/setup-adaptive-bitrate-streaming-dash-hls/.

[38] ffmpeg, Compiling or installing ffmpeg on ubuntu, (accessed January 15, 2015), (http://www.ffmpeg.org).
[39] Mp4box, (accessed March,15 2015), (http://gpac.wp.mines-telecom.fr/mp4box).
[40] Generate HAR and Browser NavigationTimingAPI data headlessly with Chrome and Firefox., https://github.com/parasdahal/speedprofile.
[41] The netfilter.org "libnetfilter_queue" project., http://www.netfilter.org/projects/libnetfilter_queue/.
[42] Super Netflix., https://chrome.google.com/webstore/detail/super-netflix/aioencjhbaolepcoappllicjebblphoc?subflicks.
[43] Selenium WebDriver., http://www.seleniumhq.org/projects/webdriver/.
[44] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., The quic transport protocol: Design and internet-scale deployment, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM, 2017, pp. 183–196.
[45] G. Carlucci, L. De Cicco, S. Mascolo, Http over udp: an experimental investigation of quic, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, ACM, 2015, pp. 609–614.
[46] S. Mekki, S. Valentin, Anticipatory quality adaptation for mobile streaming: Fluent video by channel prediction, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a, IEEE, 2015, pp. 1–3.
[47] S. Colonnese, S. Russo, F. Cuomo, T. Melodia, I. Rubin, Timely delivery versus bandwidth allocation for dash-based video streaming over lte, IEEE Communications Letters 20 (3) (2016) 586–589.
[48] I. Rubin, S. Colonnese, F. Cuomo, F. Calanca, T. Melodia, Mobile http-based streaming using flexible lte base station control, in: World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a, IEEE, 2015, pp. 1–9.
[49] S. Cicalo, N. Changuel, R. Miller, B. Sayadi, V. Tralli, Quality-fair http adaptive streaming over lte network, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2014, pp. 714–718.
[50] L. De Cicco, S. Mascolo, C.T. Abdallah, An experimental evaluation of akamai adaptive video streaming over hsdpa networks, in: 2011 IEEE International Symposium on Computer-Aided Control System Design (CACSD), IEEE, 2011, pp. 13–18.
[51] V. Adzic, H. Kalva, B. Furht, Optimizing video encoding for adaptive streaming over http, IEEE Transactions on Consumer Electronics 58 (2) (2012).
[52] L. Toni, R. Aparicio-Pardo, G. Simon, A. Blanc, P. Frossard, Optimal set of video representations in adaptive streaming, in: Proceedings of the 5th ACM Multimedia Systems Conference, ACM, 2014, pp. 271–282.

Mr. Ibrahim Ayad Ibrahim Ayad is a PhD student in the Interdisciplinary Telecom Program, College of Engineering and Applied Science, University of Colorado at Boulder. He has over 15 years with Cisco Systems where he has architected, designed and deployed complex networking solutions in the service provider and enterprise voice and video domains. His current position is a senior systems engineer at Cisco, where he performed in-depth and high-level technical presentations for customers, partners and prospects on existing Ciscos cloud computing and hosted collaboration services offers. Actively participating as a specialist on Unified Communications, provides consultative support in Collaboration architecture to partner's and Cisco systems engineers. His research interests encompass Networking architecture, Internet protocol stack, in particular TCP/UDP reliable transport, Video coding, and HTTP adaptive streaming.

Dr. Youngbin Im Youngbin Im is a postdoctoral researcher in the Department of Computer Science at University of Colorado Boulder. He received his B.S. and Ph.D. degrees in computer science and engineering from Seoul National University in 2006 and 2014, respectively. During his graduate program, he was a visiting student at Princeton University. His research interest includes mobile data offloading, next-generation Internet, multi-core based content router, video rate adaptation.

Dr. Eric Keller Eric Keller is an Assistant Professor in the Electrical, Computer, and Energy Engineering Department at the University of Colorado, Boulder. He received his PhD in 2011 from Princeton University. His research involves designing and building secure and reliable networked systems using a cross-layer approach that draws from networking, operating systems, distributed systems, and computer architecture. Recent research has focused on enabling and capitalizing on a more dynamic and programmable computing and network infrastructure, via such technologies as virtualization, software-defined networking, and the movement toward cloud based services.

Dr. Sangtae Ha Sangtae Ha is an Assistant Professor in the Department of Computer Science at the University of Colorado at Boulder. He received his Ph.D. in Computer Science from North Carolina State University. He co-founded the Princeton EDGE Lab as its first Associate Director in 2009 and led its research team as an Associate Research Scholar at Princeton University from 2010 to 2013. He is a co-founder and the founding CTO/VP Engineering of DataMi, a startup company on mobile networks, and is a technical consultant to a few startups. His research focuses on building and deploying practical network systems. He is an IEEE Senior Member and serves as an Associate Editor for IEEE Internet of Things Journal. He received the INFORMS ISS Design Science Award in 2014.