QoE-Driven Adaptive K-Push for HTTP/2 Live Streaming

Zhimin Xu, Xinggong Zhang, and Zongming Guo, *Member, IEEE*

***Abstract*— Dynamic adaptive streaming (DAS) over HTTP has been widely deployed over the Internet. However, due to the *pull-based* nature of HTTP/1.1, there exists intolerable streaming latency and high request overhead in the current DAS systems. With dynamic *k-push*, HTTP/2 live streaming promises to achieve low live latency with less overhead and small segment duration. In this paper, we propose a quality of experience (QoE) driven adaptive *k-push* mechanism (QK-Push) for HTTP/2 live streaming. The client just sends one request to set push length (K) and bitrate (v) parameters and the server would push back K segments in a batch. To determine *k-push* parameters, a probabilistic buffer model is first designed to avoid buffer underflow/overflow. Also, three QoE objective functions are designed to ensure the high streaming quality (bitrate), playback continuity, and smoothness. QK-Push casts this multi-objective optimization problem as a *Pareto optimal problem*. To solve it, a *Nash bargaining solution* is designed to balance the needs for video quality, bitrate smoothness, and request overhead. Finally, the segments in each push cycle are selected by solving the *Nash problem* with a *discrete space Lagrangian* method. We implement an HTTP/2 live streaming prototype system, with the QK-Push algorithm over modified *dash.js* and media presentation description. To evaluate the performances, the extensive live streaming experiments are carried out over a controllable network test bed and real Internet trace. The results demonstrate that the proposed QK-Push algorithm is able to improve the average bitrate up to 13%, reduce the bitrate oscillations up to 81%, decrease the startup delay up to 58%, and increase the estimate the mean opinion score up to 12% compared to the current HTTP/1.1 system.**

***Index Terms*— Dynamic adaptive streaming, live streaming, HTTP/2, adaptive *k-push*, fast start.**

I. Introduction

IN RECENT years, Dynamic Adaptive Streaming (DAS) over HTTP/1.1 has been widely used for providing uninterrupted video streaming services over harsh network conditions and heterogeneous devices. However, due to large segment duration, round-trip time (RTT) and playback buffer, the current HTTP/1.1 DAS in live video streaming often leads to high streaming latency [1]–[3]. To lower the live latency, one most straightforward way is to reduce the segment duration [2]. However, it may lead to significant request explosions in HTTP/1.1 *pull-based* live streaming. First, each request or response poses additional HTTP header and overhead to clients, servers and the network infrastructure, the request explosions may severely decrease the performance of system scalability. Second, one RTT duration is needed in each request-response pair. Request explosion may degrade link utilization especially when RTT is large and segment duration is small. Finally, if the segment duration is too small, the TCP congestion windows may not get chance to increase and fully utilize the available bandwidth.

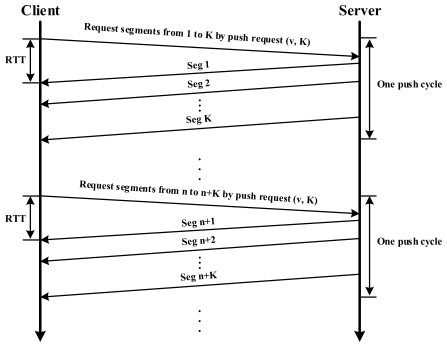

Fig. 1. Flowchart of *k-push* scheme.

Recently, HTTP/2 [4], which promises performance improvements over HTTP/1.1 due to the new features such as *Server Push*, has been standardized. The *Server Push* feature allows one server to push multiple segments with one request. Leveraging this feature, the *k-push* scheme is designed for live video streaming to help address the request explosion problem and thus reduce the live latency [3]. Fig.1 depicts the flowchart of the *k-push* scheme in HTTP/2 live streaming. In one push cycle, the client first determines the parameters of video bitrate vector $\mathbf{v}$ and push length K. Then, the client sends one request with *Push Directive* to initiate a new *k-push* session and whereafter receives K segments that sequentially pushed by the HTTP/2 server. Compared to legacy HTTP/1.1, it only takes 1 request and at least $K-1$ RTTs are saved to receive K segments in one push cycle.

In HTTP/2 *k-push*, the rate adaptation logic plays a critical role in guaranteeing high streaming service, thus attracting many research efforts [3], [5]–[12]. However, most of these methods ignore the issue how to determine the parameters

Manuscript received February 13, 2018; revised May 16, 2018; accepted June 3, 2018. Date of publication June 19, 2018; date of current version June 4, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61471009, in part by the Beijing Culture Development Funding under Grant 2016-288, and in part by the Toutiao Funding under Grant ZN20171224003. This paper was recommended by Associate Editor S. Shirmohammadi. *(Corresponding author: Xinggong Zhang.)*

Z. Xu is with the Institute of Computer Science and Technology, Peking University, Beijing 100080, China (e-mail: zhiminxu@pku.edu.cn).

X. Zhang and Z. Guo are with the Institute of Computer Science and Technology, Peking University, Beijing 100080, China, and also with the Cooperative Medianet Innovation Center, Shanghai 200240, China (e-mail: zhangxg@pku.edu.cn; guozongming@pku.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2018.2849015

($\mathbf{v}$, K) of dynamic *k-push* by the user's quality of experience (QoE). This motivates us to propose a more effective rate adaptation approach of dynamic *k-push* for HTTP/2 live streaming.

In general, designing an efficient *k-push* rate adaptation scheme for HTTP/2 live streaming faces some challenges:

- *Push length,* to cope with bandwidth variations, a small push length is preferred to increase the responsiveness of the system. However, this may lead to a large number of HTTP requests and thus bringing down network utilization due to RTT loss, additional header and processing overhead along with each request.
- *Bitrate smoothness and bandwidth utilization,* generally, there is a fundamental conflict between bitrate smoothness and bandwidth utilization. Exactly matching bandwidth achieves the highest bandwidth utilization while leading to severe bitrate fluctuations and worsening the quality of viewer's experience severely.
- *Playback stalling,* due to the latency constraint, the playback buffer of live streaming is always small. It is a big challenge for dynamic *k-push* in HTTP/2 live streaming to prevent playback stalling over the time-varying channel.
- *Startup delay,* which is defined as the time between the moment when the user clicks on "play" button and the moment when the video actually starts playing. It is a metric to balance the live latency and it is mainly limited by playback buffer size (even a small buffer size), RTT, segment duration and time of parsing Media Presentation Description (MPD) file.

In this paper, we address the rate adaption problem of HTTP/2 live streaming. Considering those factors affecting QoE for an HTTP/2 live video streaming session, including video quality, playback continuity, bitrate smoothness and startup delay, we propose a QoE-driven adaptive *k-push* (QK-Push). To ensure continuous playback, a probabilistic buffer control model is firstly designed. Then, three QoE objective functions are designed and maximized via comprehensively considering critical factors of QoE for an HTTP/2 live video streaming session, including bitrate maximization function to maximize video quality, bitrate oscillation minimization function to improve video bitrate smoothness, and push length maximization function to reduce request overhead and thus to improve link utilization. We cast the above multi-objective optimization problem as *Pareto optimal problem*. Then, a *Nash bargaining solution* is designed to balance the needs for video quality, bitrate smoothness and request overhead. Finally, the segments in each push cycle are selected by *discrete space Lagrangian* method. A *fast start mechanism* and a *push cancel mechanism* also have been designed, which provide low startup delay and responsiveness of HTTP/2 live streaming.

We develop a real HTTP/2 live streaming system with *libnghttp2_asio* [13] and modified *dash.js [14]*, and design extensive experiments on a network test-bed and real Internet trace to investigate the performance of the proposed adaptation approach, including performance under fixed bandwidth, square bandwidth and real Internet trace, and impact of different RTTs and segment durations. The experimental results demonstrate the efficiency of our method. Table. I summarizes the QoE improvement in video bitrate, bitrate oscillation, startup delay and the estimate the Mean Opinion Score (eMOS) [15], [16] of our proposed method compared to existing schemes via real Internet trace experiments. The results show that our proposed *QK-Push* is able to increase average video bitrate by 13%, smoothness by 81%, decrease startup delay by 58% and increase eMOS up to 12% compared to the *HTTP/1.1* method, and reduce bitrate oscillation by 35%, startup delay by 57%, increase average video bitrate by 11% and the eMOS by 9% compared to the *HTTP/2.0* method, which adopts HTTP/2 with *Server Push* feature. Besides, compared to the *DASH2M* method [10], average video bitrate is improved by 5%, unnecessary bitrate oscillation is reduced by 69%, startup delay is decreased by 35% and the eMOS is increased by 18 % in our method.

TABLE I
COMPARED RESULTS

Scheme	Avg. bitrate vs. *QK-Push*	Avg. oscillation vs. *QK-Push*	Avg. startup delay vs. *QK-Push*	eMOS vs. *QK-Push*
HTTP/1.1	0.87×	5.38×	2.41×	0.88×
HTTP/2.0	0.89×	1.53×	2.38×	0.91×
DASH2M	0.95×	3.19×	1.35×	0.82×
QK-Push	1.00×	1.00×	1.00×	1.00×

The main contributions of this paper can be summarized as:

- We propose a QoE-driven adaptive *k-push* algorithm for low latency HTTP/2 live streaming and design three different objective functions to maximize visual quality, including bitrate maximization function, bitrate oscillation minimization function and push length maximization function. And we cast the above multi-objective optimization problem as a *Pareto optimal problem*.
- We leverage a probabilistic buffer model to ensure continuous video playback. Since the network bandwidth can be highly dynamic, which is hardly predicted accurately, to avoid unnecessary playback stalls, probabilistic buffer constraints are built to guarantee that buffer stay in a given range to avoid buffer overflow/underflow.
- We design a *Nash bargaining solution* algorithm to balance the needs for video playback quality, bitrate smoothness and request overhead. By solving the *discrete optimization problem*, we can decide the *k-push* parameters for each push session.
- We implement the proposed QK-Push algorithm in a dynamic *k-push* system prototype and integrate additional mechanisms of the *fast start* and *push cancel*, which reduce startup delay and increase responsiveness.

The rest of the paper is organized as follows. Sec.II surveys the related works. In Sec.III, we introduce the multi-objective optimization problem. In Sec.IV, the probabilistic buffer model is designed. Then solutions are given in Sec.V. We also introduce the design of our real HTTP/2 live streaming system in Sec.VI, and show experimental results in Sec.VII. Finally, we conclude the paper in Sec.VIII.

II. Related Works

DAS over HTTP/1.1 has emerged as a dominant technology to stream video over the Internet [17]–[19], and attracted many research efforts [20]–[33] for its adaptability to heterogeneous network and devices. Most of them switch bitrate either by bandwidth [27] or by buffer occupancy [31], in a request-response downloading paradigm of HTTP/1.1. However, due to the *pull-based* nature of HTTP/1.1, it incurs significant round-trip time (RTT) overheads, which impairs bandwidth utilization especially for live streaming with small segments. Fortunately, the emergence of the HTTP/2 brings a promising solution for the above problem.

The HTTP/2 *Server Push* feature [4] is a mechanism designed for reducing web page load latency initially. Some pioneering works [3], [5], [34] demonstrate the benefits of HTTP/2 *push* in video streaming. Comparing with HTTP/1.1, lower overhead, higher link utilization, lower live latency and lower power cost can be obtained by HTTP/2 *k-push*, i.e. server pushes K video segments in a batch to respond to one request. But, it's still an open issue to determine the push length K and bitrate $\mathbf{v}$ to adapt to the dynamic network.

Some pioneering works [3], [5], [6] propose a *fixed k-push* method, which pushes a fixed number of segments in one response. Sheng and Viswanathan [3] explore that it is possible to achieve low live latency by pushing fixed K segments with small duration, such as 1 second. In [5], Wei and Swaminathan also study the potential of *fixed k-push* in eliminating unnecessary requests and battery power in mobile streaming. Huysegems *et al.* [6] employ a *full-push* scheme which keeps pushing segments continually and controls the streaming like legacy RTSP protocol with the control messages, such as starting, stopping, pausing, resuming. However, these *fixed k-push* schemes may damage the adaptability if network conditions are changed abruptly during the K segments transmission.

Other works [7]–[9], [11], [12] propose *adaptive k-push* schemes, which dynamically adjust the parameter K during one push cycle to adapt to the runtime environment. In [11], Wei *et al.* employ an *adaptive k-push* strategy to reduce power consumption. In [10], Xiao *et al.* design DASH2M, which dynamically adjusts parameters of the push length and bitrate to improve power efficiency in mobile streaming. However, these works only focus on reducing overhead and power consumption but ignores the user's Quality of Experience (QoE) in video streaming.

Recently, some efforts [35]–[37] explore HTTP/2 push mechanism in 360° virtual reality videos streaming. They crop 360-degree video into tiles and push only the tiles in the user's Field-of-View (FoV) to the user. This reduces the bitrate and overhead of streaming. But this scheme is still simple which does not consider the adaptability to the network.

On the other hand, one of the most important objectives of video streaming is QoE. QoE is defined as "the overall acceptability of an application or service, as perceived subjectively by the end-user". The QoE of HTTP streaming is influenced by several factors, such as video playback quality [38]–[40], video bitrate switching frequency and amplitude [41], [42], buffer overflow/underflow [24], [43], [44], request overhead [5], [6], [11], and live latency [3], [45], [46].

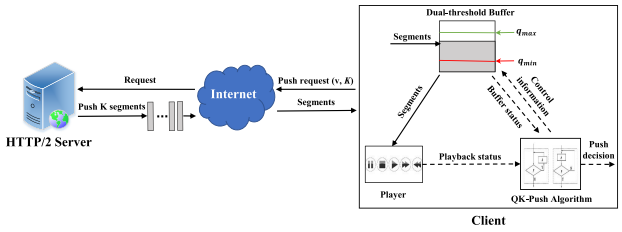

Fig. 2. Flowchart of proposed QoE-driven adaptive *k-push*.

In Claeys *et al.* [15] and Sobhani *et al.* [16] propose and use the QoE model, the estimate the Mean Opinion Score (eMOS), for HTTP Adaptive Streaming (HAS). In eMOS, the average quality of selected representations, the number and magnitude of switches among different representations, and the frequency and duration of freezes are considered as the most important factors that have an impact on QoE. This paper will use these QoE metrics as the *k-push* objectives to find the push parameters $(\mathbf{v},\ K)$.

III. QoE Objectives

In this section, we introduce the designed three QoE objective functions. Fig.2 shows the flowchart of QK-Push.

A. Bitrate Maximization

To provide high video playback quality for a streaming session, the ideal bitrate in each push cycle should be the highest ones that the network can sustain. Meanwhile, the selection of the video bitrate should not cause any stalling. Let R denotes the bitrate set which holds N different quality levels. Let $\mathbf{v}$ denotes a bitrate vector of $v(l)$ with length $K_{\max}$, and $v(l)$ denotes the video bitrate for the l-th segment in the next push cycle, K denotes the number of consecutive segments with non-zero bit-rate from the beginning of $\mathbf{v}$. In the other words, $v(l) > 0$ when $0 < l \leq K$ and $v(l) = 0$ when $K < l \leq K_{\max}$. Then we have the function as

$$\begin{aligned} F_1(\mathbf{v},\ K) &= \frac{r_{\max} - \frac{1}{K}\sum_{l=1}^{K} v(l)}{r_{\max}}, \\ s.t.\ v(l) &> 0, \quad 0 < l \leq K, \\ v(l) &= 0, \quad K < l \leq K_{\max}, \end{aligned} \tag{1}$$

where $v(l)$ denotes the video bitrate for l-th segment in the next push cycle, and $v(l) \in R$, $0 < l \leq K$. By minimizing (1), we can maximize the bitrate.

B. Bitrate Oscillation Minimization

The frequency and amplitude of video bitrate switching have a great effect on QoE, and switching back-and-forth between different bitrate may significantly degrade viewer's experience [47], [48]. We define the bitrate instability metric [49] of the i-th segment as

$$f(i) = \frac{\sum_{l=0}^{d-1} |v(i-l) - v(i-l-1)| \cdot \omega(l)}{\sum_{l=1}^{d} v(i-l) \cdot \omega(l)}, \tag{2}$$

where $v(l)$ denotes the video bitrate for the l-th segment in the next push cycle and $v(l) \in R$, $0 < l \leq K$. $f(i)$ is the

weighted sum of all switch steps observed within the last d segments divided by the weighted sum of bitrates. We use the weight function $\omega(l) = d - l$ to add a linear penalty to more recent bitrate switch. In this paper, we let $d = \max(20, i)$.

To provide smooth video bitrate for a streaming session, we design the bitrate oscillation function as

$$F_2(\mathbf{v},\ K) = \frac{1}{K}\sum_{i=1}^{K} f(i), \tag{3}$$

where $f(i)$ denotes the bitrate instability metric for i-th segment of next push cycle, which can be solved according to the instability model defined as (2). By minimizing (2), the bitrate oscillation can be minimized.

C. Push Length Maximization

Each request or response poses additional header and processing overhead to the client, the server, and the network infrastructure. Besides, the RTT overhead along with each request also will decrease the link utilization and average video quality [6]. To improve the scalability and link utilization of HTTP/2 live streaming, a push length function is designed to minimize the request overhead. The equivalent function is expressed as

$$F_3(K) = \frac{K_{\max} - K}{K_{\max}}. \tag{4}$$

And we can minimize the request overhead by minimizing (4).

IV. Probabilistic Buffer Model

In this section, we propose a probabilistic buffer model to keep playback continuity and design a probabilistic buffer constraint that the chance of buffer occupancy falling out of dual-threshold must be less than the given probability threshold. Under this constraint, all feasible solutions maximizing the aforementioned QoE objectives are solved.

A. Buffer Model

In this work, the buffer occupancy is denoted by the buffered video time, because a video buffer may contain segments from different versions, i.e., different video bitrate. There is no longer a direct mapping between the buffered video size and the buffered video time.

Let $q(t)$ denote the buffered video time at time t, which can be modeled as a queue with constant service rate of unity, i.e., in each second, a piece of buffered video with one second length of playback time is consumed and dequeued from the buffer. The enqueue process is driven by downloading and the selected video version. All versions of video are broken into equal length segments, each of which contains the same playback time of T. Without loss of generality, suppose a client starts to send next request to initiate next push session at t^s. And, the time instant for completing receiving the i-th segment of next push cycle is denoted as t_i^e. Then, we have

$$t_i^e - t^s = \frac{T\sum_{l=1}^{i} v(l)}{C} + rtt \tag{5}$$

where C and rtt denote the available bandwidth and round-trip time (RTT) between server and client respectively. (5) denotes the total consumed time to download all previous i (from first to i-th) segments in the next push cycle. When $i = K$, (5) also denotes the total time consumed to complete the whole push session, which is also the dequeued time from the buffer. Then, the buffer occupancy evolution during each push session becomes

$$q(t_i^e) = q(t^s) + iT - (\frac{T\sum_{l=1}^{i} v(l)}{C} + rtt), \tag{6}$$

where the second term of (6) is the enqueued video time upon the completion of receiving segments 1 to i, and the third term is the consumed video time taken to receive all segments, which reflects the fact that the buffer occupancy is consumed linearly at a rate of one per second.

B. Probabilistic Buffer Constraints

To ensure continuous playback during the whole streaming session, the selected adaptation pair $(\mathbf{v}, K)$ in each push cycle should be able to keep buffer occupancy no less than $q_{\min}$ during the whole push session. On the other hand, to prevent the buffer from overflow, the selected adaptation pair should be able to maintain buffer occupancy no more than $q_{\max}$ during the whole push session. Combining the buffered video time model, we express above buffer control model as

$$q(t_i^e) > q_{\min}, \quad i = 1, 2, \ldots, K, \tag{7}$$

$$q(t_i^e) < q_{\max}, \quad i = 1, 2, \ldots, K. \tag{8}$$

(7-8) denote constraints that during next push session, upon the completion of receiving each segment i, the buffer occupancy must fall in the range $[q_{\min}, q_{\max}]$. To find all adaptation pairs satisfying above constraints, it is a good choice to predict link capability accurately and solve (7-8) directly based on the buffer occupancy estimation model in (6). However, as demonstrated in [50], the network bandwidth is time-varying which is hardly predicted accurately. Thus, we use a probabilistic event to describe this problem. We rewrite the above buffer control model as

$$P(q(t_i^e) < q_{\min}) \le p_\epsilon, \quad i = 1, 2, \ldots, K, \tag{9}$$

$$P(q(t_i^e) > q_{\max}) \le p_\epsilon, \quad i = 1, 2, \ldots, K, \tag{10}$$

where p_ϵ represents a given probability threshold value. (9-10) denote constraints that during next push session, upon the completion of receiving each segment i, the chance of buffer occupancy falling out of dual-threshold must be less than a given probability threshold. Then, each adaption pair $(\mathbf{v},\ K)$ satisfying constraints (9-10) is able to keep buffer occupancy staying in the range $[q_{\min}, q_{\max}]$ during next push session.

C. Probabilistic Bandwidth

One key problem is how we can get the $P(q(t_i^e) < q_{\min})$ and $P(q(t_i^e) > q_{\max})$. Firstly, we let $L_{i,\min}$ and $L_{i,\max}$ denote the margin time to the threshold as

$$L_{i,\min} = q(t^s) + iT - q_{\min} - rtt, \tag{11}$$

$$L_{i,\max} = q(t^s) + iT - q_{\max} - rtt. \tag{12}$$

We define $C_\tau(t)$ as the average end-to-end available bandwidth during the time interval $(t, t+\tau)$. Then, given segment i and video bitrate vector $\mathbf{v}$, according to (6-8) and (11-12), we have

$$P(q(t_i^e) < q_{\min}) = P(C_{L_{i,\min}}(t^s + rtt) < \frac{T\sum_{l=1}^{i} v(l)}{L_{i,\min}}), \quad (13)$$

$$P(q(t_i^e) > q_{\max}) = P(C_{L_{i,\max}}(t^s + rtt) > \frac{T\sum_{l=1}^{i} v(l)}{L_{i,\max}}). \quad (14)$$

As demonstrated in the references [50] and [51], the available bandwidth in a time slot τ can be viewed as a stationary and identically distributed stochastic process, and at any time instant t the process is described by the same random variable $C_\tau(t)$. Furthermore, $C_\tau(t)$ can be viewed as a Gaussian random variable with mean $\widetilde{\mu}$ and variance $\widetilde{\sigma}_\tau^2$. The mean $\widetilde{\mu}$ does not depend on τ. The variance $\widetilde{\sigma}_\tau^2 = Var[C_\tau(t)]$, however, depends strongly on τ. So we have

$$C_{L_{i,\min}}(t^s + rtt) \sim (\widetilde{\mu}, \widetilde{\sigma}^2_{L_{i,\min}}), \quad (15)$$

$$C_{L_{i,\max}}(t^s + rtt) \sim (\widetilde{\mu}, \widetilde{\sigma}^2_{L_{i,\max}}). \quad (16)$$

Under the assumption that $C_\tau(t)$ is independently and identically distributed, the variance decreases inversely proportional with the length of the measuring time duration, i.e., $Var[C_\tau(t)] = Var[C_{k\tau}(t)]/k$. Thus, we can know that $C_{L_{i,\min}}(t^s + rtt)$ and $C_{L_{i,\max}}(t^s + rtt)$ are also Gaussian random variables. If we can get $\widetilde{\mu}$ and $\widetilde{\sigma}_\tau^2$, then we can know $\widetilde{\sigma}^2_{L_{i,\min}}$ and $\widetilde{\sigma}^2_{L_{i,\max}}$ by

$$\widetilde{\sigma}^2_{L_{i,\min}} = \frac{\widetilde{\sigma}_\tau^2 \cdot \tau}{L_{i,\min}}, \ \widetilde{\sigma}^2_{L_{i,\max}} = \frac{\widetilde{\sigma}_\tau^2 \cdot \tau}{L_{i,\max}}. \quad (17)$$

Let $erf(\bullet)$ denote the Gauss error function, we have

$$P(q(t_i^e) < q_{\min}) = P(C_{L_{i,\min}}(t^s + rtt) < \frac{T\sum_{l=1}^{i} v(l)}{L_{i,\min}}) = \frac{1}{2}\left(1 + erf\left(\frac{\frac{T\sum_{l=1}^{i} v(l)}{L_{i,\min}} - \widetilde{\mu}}{\sqrt{2}\widetilde{\sigma}_\tau}\sqrt{\frac{L_{i,\min}}{\tau}}\right)\right), \quad (18)$$

$$P(q(t_i^e) > q_{\max}) = P(C_{L_{i,\max}}(t^s + rtt) > \frac{T\sum_{l=0}^{i} v(l)}{L_{i,\max}}) = \frac{1}{2}\left(1 + erf\left(\frac{\frac{T\sum_{l=1}^{i} v(l)}{L_{i,\max}} - \widetilde{\mu}}{\sqrt{2}\widetilde{\sigma}_\tau}\sqrt{\frac{L_{i,\max}}{\tau}}\right)\right). \quad (19)$$

Finally, we can get the probability $P(q(t_i^e) < q_{\min})$, $P(q(t_i^e) > q_{\max})$ from (18-19) respectively, which means that we can know whether the selected segments vector $\mathbf{v}$ satisfies the conditions in (9-10). Then, we obtain the probabilistic buffer constraints, which not only consider the buffer occupancy, but also consider the available bandwidth.

V. Solution

In this section, we firstly give the Pareto optimality for the proposed QK-Push. Then, we will design a *Nash bargaining solution* to balance the needs for link utilization, bitrate smoothness and low-overhead. Finally, we describe the algorithm to find the Nash *disagreement point*.

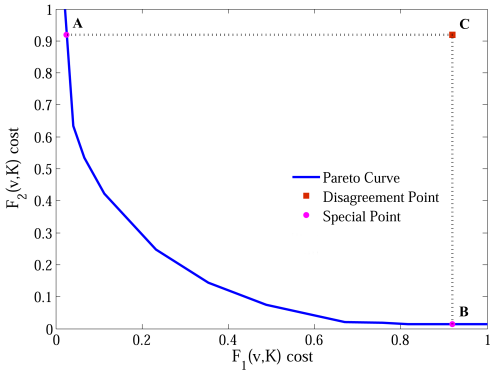

Fig. 3. Pareto optimality and *disagreement point* for bitrate and bitrate oscillation cost.

A. Pareto Optimality

Since the designed three objective functions are conflicting with each other, to measure efficiency in a system with multiple objectives, a common approach is to explore the *Pareto Curve*, which characterizes the trade-off of potentially conflicting goals of different parties. To trace the tradeoff and balance the needs for video quality, bitrate smoothness and request overhead, one simple way is to optimize a weighted sum of the single objective function, which is constructed as a sum of objective functions multiplied by weighting coefficients, which reflect preferences for different optimization objectives under the (9-10) constraints, as follows

$$\min U(\mathbf{v}, K) = w_1 F_1(\mathbf{v}, K) + w_2 F_2(\mathbf{v}, K) + w_3 F_3(K) \quad (20)$$

where $\sum_{j=1}^{3} w_j = 1, w_j > 0, j = 1, 2, 3$, which represent the relative weight of three objectives.

To illustrate the Pareto optimality for the needs among video quality, bitrate smoothness and request overhead, we can plot the achieved $F_1(\mathbf{v}, K)$, $F_2(\mathbf{v}, K)$ and $F_3(K)$ versus each other by varying w_1, w_2, w_3. Then, we can obtain the Pareto surface. Without loss of generality, we illustrate a sample Pareto curve in Fig.3 to observe the change of optimal solution of $F_1(\mathbf{v}, K)$ and $F_2(\mathbf{v}, K)$, which only vary w_1, w_2.

However, it is difficult to figure out which point on the Pareto surface is the best solution. Solving problem in the (20) for each weighting coefficient w_j and tuning w_j in a *trial-and-error* fashion is impractical and inefficient. Furthermore, we need to compute appropriate weight parameters w_j for each combination of $\mathbf{v}$ and K, and tune w_j to explore a broad region of system operating points. It is not straightforward to weigh the trade-off among the three objectives. To overcome these problems, we propose a *Nash bargaining solution* in Sec.V-B.

B. Nash Bargaining Solution

As demonstrated in [52], if all of the weights are positive, as assumed in this study, the optimum solution to the weighted-sum optimization problem depicted in (20) is always

Pareto optimal. However, since there exists a number of Pareto optimal solutions, no single feasible solution exists that simultaneously minimizes all objective functions in the multi-objective optimization problem. Fortunately, we can solve this problem by the *Nash bargaining solution* [53].

The *Nash bargaining solution* solves the following optimization problem under the (9-10),

$$\max(F_1^0 - F_1)(F_2^0 - F_2)(F_3^0 - F_3) \tag{21}$$

where $(F_1,\ F_2,\ F_3)$ is a *Nash bargaining solution* for three objectives $F_1(\mathbf{v},\ K)$, $F_2(\mathbf{v},\ K)$ and $F_3(K)$, and $(F_1^0,\ F_2^0,\ F_3^0)$ is a constant called the *disagreement point*, which is the starting point of the negotiation and the point that the players can expect to receive if negotiations break down. In other words, the cost cannot more than *disagreement point* value for each bargaining party. The selection of the *disagreement point* will be introduced in Sec.V-C.

By optimizing the performance of three objectives, the *Nash bargaining solution* guarantees that the joint system is optimal and fair (it is the only solution that satisfies all of four axioms: *Pareto optimality, Symmetry, Expected utility axiom, Independence of irrelevant alternatives*) [53].

The (21) can be converted to

$$\begin{aligned}
&\max\ log(F_1^0 - F_1) + log(F_2^0 - F_2) + log(F_3^0 - F_3)\\
&s.t.\ P(q(t_i^e) < q_{\min}) \le p_\epsilon, \quad i = 1, 2, \ldots, K,\\
&\quad P(q(t_i^e) > q_{\max}) \le p_\epsilon, \quad i = 1, 2, \ldots, K,\\
&\quad \upsilon(l) > 0, \quad 0 < l \le K,\\
&\quad \upsilon(l) = 0, \quad K < l \le K_{\max},\\
&\quad 1 \le K \le K_{\max}.
\end{aligned} \tag{22}$$

The log function is monotonic and the feasible solution space is same as (21). The (22) is a discrete optimization problem with inequality constraints. There are no gradients or global optimal solution for discrete optimization problems. However, we can find an approximative solution by *discrete space Lagrangian* methods [54]. As in our problem, the value K is from 1 to $K_{\max}$, so we can iterate each K to reduce the solution complexity.

Given a K, we denote $h_i(\mathbf{v},\ K) = P(q(t_i^e) < q_{\min}) - p_\epsilon$ and $g_i(\mathbf{v},\ K) = P(q(t_i^e) > q_{\max}) - p_\epsilon$, then we get *discrete Lagrangian function* as

$$\begin{aligned}
&L_d(\mathbf{v}, K, \boldsymbol{\lambda}, \boldsymbol{\mu})\\
&\quad = log(F_1^0 - F_1) + log(F_2^0 - F_2) + log(F_3^0 - F_3)\\
&\qquad - \sum_{i=1}^{K} \lambda_i h_i(\mathbf{v}, K) - \sum_{i=1}^{K} \mu_i g_i(\mathbf{v}, K),
\end{aligned} \tag{23}$$

where $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ are *Lagrangian multipliers* vector, and $\lambda_i \ge 0$, $\mu_i \ge 0$. Since the Dual problem $\min_{\boldsymbol{\lambda},\boldsymbol{\mu}} L_d(\mathbf{v}, K, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is convex, we iterate λ_i and μ_i with the following price updates

$$\lambda_i(\pi + 1) = [\lambda_i(\pi) - \theta_{\lambda i}(\pi) h_i(\mathbf{v}, K)]^+, \tag{24}$$

$$\mu_i(\pi + 1) = [\mu_i(\pi) - \theta_{\mu i}(\pi) g_i(\mathbf{v}, K)]^+, \tag{25}$$

where $\theta_{\lambda i}(\pi) = 1/\pi$ and $\theta_{\mu i}(\pi) = 1/\pi$ denote the step size in current iteration π.

For each price of $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, we use a direction of maximum potential raise (DMPR) method in discrete space [54] to find an approximate solution for the primal discrete problem, $\max_{\mathbf{v}} L_d(\mathbf{v}, K, \boldsymbol{\lambda}, \boldsymbol{\mu})$. We define $\mathcal{N}(\mathbf{v})$, the neighborhood of point $\mathbf{v}$ in discrete space, as

$$\begin{aligned}
\mathcal{N}(\mathbf{v}) = \{&(\upsilon_-(1), \ldots, \upsilon(K_{\max})),\ (\upsilon_+(1), \ldots, \upsilon(K_{\max})),\\
&\ldots,\ (\upsilon(1), \ldots, \upsilon_-(K), \ldots, \upsilon(K_{\max})),\\
&(\upsilon(1), \ldots, \upsilon_+(K), \ldots, \upsilon(K_{\max}))\},
\end{aligned} \tag{26}$$

where $\upsilon_-(i)$ is the bitrate of preceding quality level compared to $\upsilon(i)$, and the $\upsilon_+(i)$ is the bitrate of following quality level compared to $\upsilon(i)$.

Then, we can iteratively search the neighborhood of point $\mathbf{v}^\tau$ in iteration τ to find the maximal L_d,

$$L_d(\mathbf{v}^{\tau+1}, K, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \max_{\mathbf{v}' \in \mathcal{N}(\mathbf{v}^\tau) \cup \{\mathbf{v}^\tau\}} L_d(\mathbf{v}', K, \boldsymbol{\lambda}, \boldsymbol{\mu}) \tag{27}$$

Then we update $\mathbf{v}^\tau$ with the maximum potential raises neighborhood point $\mathbf{v}'$ iteratively. Until $\mathbf{v}^\tau$ converges, we will find a discrete local maximum. Finally, we only get an approximate solution to our primal problem.

Summarizing, the *Nash bargaining solution* is also the Pareto solution in (20). The pseudo-code of the Nash bargaining algorithm is presented in Algorithm 1.

Algorithm 1 Adaption Algorithm

1: Parameters: available bandwidth C, buffer length $q(t)$, RTT rtt for current time, hyper-parameter p_ϵ, and *disagreement point* $(F_1^0,\ F_2^0,\ F_3^0)$, which can be calculated by the approach in Sec.V-C.
2: Initialization: set $K = 1$, $L_d' = 0$. Let λ_i and μ_i equal to some nonnegative value, and $\upsilon(i)^* \leftarrow r_{\min}$ for all i.
3: **(i).** Set $K \leftarrow K + 1$, and $\pi \leftarrow 1$.
4: **(ii).** Set $\tau \leftarrow 1$, let $\mathbf{v}^\tau$ as last push cycle's value $\mathbf{v}^*$ and $\upsilon^\tau(j) \leftarrow 0$ for all j $(K < j \le K_{\max})$.
5: **(iii).** Calculate $\mathcal{N}(\mathbf{v}^\tau)$ with function in (26) and $L_d(\mathbf{v}^{\tau+1}, K, \boldsymbol{\lambda}, \boldsymbol{\mu})$ with function in (27).
6: **(iv).** Set $\tau \leftarrow \tau + 1$ and go to step **(iii)** (until $\mathbf{v}^\tau$ converges).
7: **(v).** Calculate the value of (23). If the value is greater than L_d', broadcast and update the new value $\mathbf{v}^* \leftarrow \mathbf{v}^\tau$, $K^* \leftarrow K$, and $L_d' \leftarrow L_d(\mathbf{v}^\tau, K, \boldsymbol{\lambda}, \boldsymbol{\mu})$.
8: **(vi).** For each i, update its prices with the function in (24-25).
9: **(vii).** Set $\pi \leftarrow \pi + 1$ and go to step **(ii)** (until converging).
10: **(viii).** Go to step **(i)** (until $K \ge K_{\max}$).
11: **return** $(\mathbf{v}^*,\ K^*)$.

C. Disagreement Point

The *disagreement point* is the value the players can expect to receive if negotiations break down. This could be some focal equilibrium that both players could expect to play. Therefore, it stands to reason that each player should attempt to choose his *disagreement point* in order to maximize his bargaining position. The choice of the *disagreement point* is subject to different economic considerations.

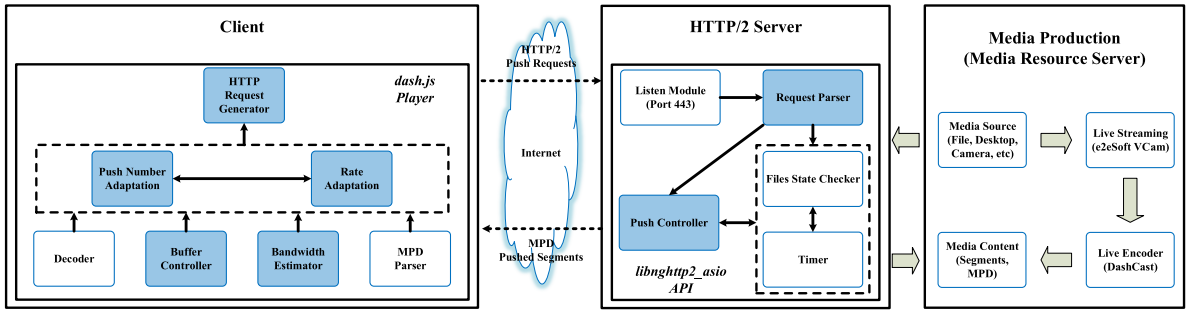

Fig. 4. System architecture.

In this paper, we aim to minimize three objectives, $F_1(\mathbf{v},\ K)$, $F_2(\mathbf{v},\ K)$ and $F_3(K)$. We denote $(F_1^0,\ F_2^0,\ F_3^0)$ as the *disagreement point*. It is often advantageous to increase one's own disagreement payoff while harming the opponent's disagreement payoff. So, we can respectively consider three different objectives, and set it as $(F_1\_\max,\ F_2\_\max,\ F_3\_\max)$, which means we cannot accept the value of F_1, F_2 and F_3, that bigger than $F_1\_\max$, $F_2\_\max$ and $F_3\_\max$. As shown in Fig.3, we plot the *disagreement point* C of F_1 and F_2 by fixing the value of F_3, and all the feasible solutions belong to the closed region ABC.

To solve $(F_1\_\max,\ F_2\_\max,\ F_3\_\max)$, we consider three objectives respectively. For $F_3\_\max$, it is very easy to set $F_3\_\max = (K_{\max} - 1)/K_{\max}$, which means the push length is 1, and the *disagreement point* value is achieved for $F_3(K)$.

For $F_1\_\max$ and $F_2\_\max$, they are mutually exclusive. In order to maximize its own bargaining position, each objective should attempt to choose a *disagreement point* which is the most beneficial for itself. Towards this objective, it is often advantageous to increase one's own disagreement payoff while harming the opponent's disagreement payoff. If we only minimize F_1, then F_2 will be maximum and achieve its own *disagreement point* value, and vice versa. As result of that, we can get the *disagreement point* value $F_1\_\max$ of F_1 by only minimizing F_2, and get the *disagreement point* value $F_2\_\max$ of F_2 by only minimizing F_1. We denote QoE_1 as the minimized problem for F_1, and the QoE_2 as the minimized problem for F_2. Then under the (9-10) constraints and $K = 1$, we can derive these two minimization problems as

$$QoE_1 = \arg\min_{\mathbf{v}} F_1(\mathbf{v},\ K) \tag{28}$$

$$QoE_2 = \arg\min_{\mathbf{v}} F_2(\mathbf{v},\ K) \tag{29}$$

Above functions are easy to solve. We consider a simple case, where the last segment's bitrate is $r_{\min}$, and the available bandwidth C is bigger than the highest bitrate $r_{\max}$. We firstly only optimize QoE_1, which will hurt interests of F_2. Then, we only optimize QoE_2, which will hurt interests of F_1. As shown in Fig.3, if we minimize QoE_1, then QoE_2 will be maximal and we can get $F_2\_\max$ as the point B, we also can get $F_1\_\max$ as the point A vice versa. Further, we can derive the *disagreement point* $(F_1\_\max,\ F_2\_\max)$ as the point C for $F_1(\mathbf{v},\ K)$ and $F_2(\mathbf{v},\ K)$. Then, we can get the *disagree point* $(F_1\_\max,\ F_2\_\max,\ F_3\_\max)$ for three objectives.

VI. System Implementation

In this section, we present the implementation details of QoE-driven adaptive *k-push* (QK-Push) in HTTP/2 live streaming. The system architecture is shown in Fig. 4.

A. Media Production

This part prepares live video segments and Media Presentation Description (MPD). It contains a *Media Source* module with the world's first open movie *Elephants Dream*, a *Live Streaming* module with *e2eSoft VCam*, a *Live Encode* module with open source software of *GPAC DashCast* [55] and a *Media Content* module by modifying the attribute *@mediaPresentationDuration* or *@minimumUpdatePeriod* and *@startNumber* in *MPD* file to support live streaming.

B. HTTP/2 Server Implementation

We implement our HTTP/2 live streaming system in Linux/Unix platforms. The HTTP/2 server is implemented based on *libnghttp2_asio* [13], an open-source high-level HTTP/2 C++ library. Once received a request with *PushDirective*, the server launches a push session, where the corresponding segments are pushed back sequentially. The server side contains a *Listen* module with supporting of SSL/TLS encrypted connection, a *Request Parser* to support *@fast_start_directive*, *@fast_start_ACK_directive*, *@push_directive* and *@push_cancel_directive* push directives, a *Files State Checker* to check the existence of live streaming segment, a *Timer*, and a *Push Controller*.

C. Client Implementation

We implement the video player based on the open source available MPEG-DASH *dash.js* [14] player and implement our QK-Push algorithm in HTTP/2 live streaming scheme. The client consists of an *MPD Parser*, a *Bandwidth Estimator*, a *Buffer Controller* to support our dual-threshold buffer model, a *Decoder*, a *Push Number Adaptation* module, a *Rate Adaptation* module and a *HTTP Request Generator* to generate the request by adding *@fast_start_directive*, *@fast_start_ACK_directive*, *@push_directive* and *@push_cancel_directive* into HTTP request header according to different requirement.

D. Fast Start Mechanism

To decrease startup delay, we can take advantage of the *fast start mechanism*. The playback duration is classified into two period: fast-start period and normal push period. At the fast-start period, the client sends an MPD request which contains a *Fast-Start Directive* to the server. After receiving the MPD request, the server will parse the *Fast-Start Directive* and send MPD response which contains a *Fast-Start ACK Directive*. At the same time, the server will push some segments to the client. When the client received the MPD file, it will parse the MPD response header and get the fast start push information. If the next requested segment is not in the response list, the client will directly request these segments from the server. In normal push period, the client will normally run push session by HTTP request with *Push Directive*.

E. Push Cancel Mechanism

Based on the proposed QK-Push, the adaption pair $(\mathbf{v}^*, K^*)$ obtained in Algorithm.1 is able to provide smooth video bitrate with high quality, while maintaining the low overhead and continuous playback. However, since the bandwidth may be highly dynamic, there still have chances that buffer occupancy falls out of dual-threshold $q_{\min}$ and $q_{\max}$, which means the high risk of buffer overflow/underflow. To prevent the buffer from overflow/underflow, the client must terminate the running push session immediately and launch a new push cycle.

The running push cycle can be canceled via HTTP/2 stream termination mechanism. If a great bandwidth mismatch is found, the client can cancel the promised but not complete streams by sending *RST_STREAM* frames associated with the stream IDs (which can be extracted from *PUSH_PROMISE* frames). Once the running push session is terminated, the client will send a new request to initiate another push session.

VII. Performance Evaluation

In this section, we evaluate our QK-Push in HTTP/2 live streaming by conducting controlled experiments on a network test-bed and experiments over the real Internet trace.

A. Experiment Setup

We implement our HTTP/2 live streaming system in Linux/Unix platforms. Our test-bed consists of four nodes: media resource server, HTTP/2 server with TC controller, router, and DASH client as shown in Fig.5. The HTTP/2 server and the client are deployed on a 64-bit Linux machine respectively. The installed operating systems are Ubuntu 16.04 with Linux Kernel. The TC controller is also installed in the HTTP/2 server to control the network bandwidth and introduce a planned network delay.

In our experiments, the server provides five different versions of video bitrate (345Kbps, 618Kbps, 1.57Mbps, 2.54Mbps, and 3.60Mbps). Each version of the video is divided into video segments with the same length (1sec, 2sec, 4sec, and 6sec). The buffer length is set as 12sec, and the start playback buffer offset is set as 6sec. For performance

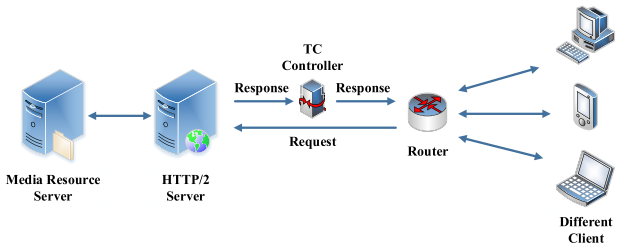

Fig. 5. Network topology in test-bed.

comparison, in addition to our method, we also implement the typical bandwidth-based scheme for DAS over HTTP/1.1. Besides, the bandwidth-based adaption method for DAS over HTTP/2.0 and the similar DASH2M proposed in [10] are also implemented. All the methods of comparison as follows:

1) **HTTP/1.1** method is a bandwidth-based scheme, which aims to maintain the buffer occupancy staying at a high level so as to ensure continuous playback. In this method, the video bitrate is always switched down to an available bandwidth.
2) **HTTP/2.0** method is also a bandwidth-based method, which is similar to the HTTP/1.1 method. In addition to that, it also adopts *Server Push* feature in HTTP/2, and sends K same video bitrate segments to the client in one push cycle. We can find the advantage of HTTP/2 by comparing HTTP/1.1 and HTTP/2.0 method.
3) **DASH2M** aims to stabilize the buffer occupancy around given target level to ensure continuous playback. To optimize playback quality level, the maximal video bitrate under constraints that the estimated buffer is no less than given target level is select.
4) **QK-Push** is our method which aims to avoid buffer underflow/overflow within a certain tolerance. Our method not only balances the needs for video quality, bitrate smoothness and request overhead, but also provides low startup delay and flexibility for live streaming.

In performance comparison, we take several measurement metrics into consideration, *video bitrate*, *Peak signal-to-noise ratio (PSNR)*, *startup delay*, *stalling ratio*, *buffer occupancy*, *instability*, which was calculated by function in (2), and *the estimate the Mean Opinion Score (eMOS)*, which was proposed and used in [15] and [16].

B. Performance Under Fixed Bandwidth

In this section, we show experiments under the case that the available bandwidth is fixed and set to 1.9Mbps, with 100ms RTT and 1sec segment duration, as shown in Fig.6.

When the available bandwidth keeps fixed, the video bitrate with the HTTP/1.1 method should be stable, and maintain buffer occupancy staying at a high level. However, we can find that the bitrate will switch down and up once in a while. First, this is because we control the bandwidth at 1.9Mbps, but the bandwidth could not accurately be 1.9Mbps and it will also fluctuate up and down the 1.9Mbps. Second, we use the variable bitrate (VBR) encoding method which will lead to the segment size fluctuation in the vicinity of bitrate level.

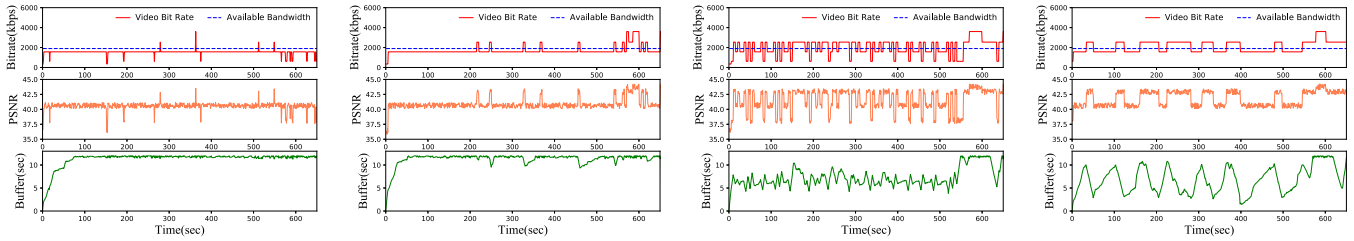

Fig. 6. Performance under fixed bandwidth with 100ms RTT and 1sec segment duration environment. (a) HTTP/1.1. (b) HTTP/2.0. (c) DASH2M. (d) QK-Push.

In the HTTP/2.0 method, the buffer variation is similar to the HTTP/1.1 method. Because the selection of video bitrate for the bandwidth-based scheme is solely based on bandwidth. By this scheme, the selected video bitrate is never allowed to be higher than the available bandwidth and make the buffer up to high level quickly. However, the bitrate and PSNR variation are more stable than the HTTP/1.1 method. This is because the HTTP/2 *Server Push* feature can make the server send segments that the client has not yet requested. In the HTTP/1.1 protocol, the client needs to send each segment request to the server, which will degrade the link utilization (especially when the RTT is long). As the *k-push* mechanism is adopted and the server will push k same bitrate segments to the client in one push cycle, the HTTP/2.0 method will overcome some small spikes in the network bandwidth variation and have a high average video bitrate.

In DASH2M method, the video bitrate and PSNR fluctuate heavily, while it stabilizes buffer occupancy. Because the objective of this method focuses on maximizing average quality level while maintaining buffer occupancy no less than the predefined threshold, however the smoothness of video bitrate is not well considered.

Different from the bandwidth-based method, with QK-Push, the bitrate switches between 1.57Mbps and 2.54Mbps. Because QK-Push method determines adaptive segments based on the probabilistic buffer constraints and the multi-objective optimization decision, where a video bitrate higher than captured bandwidth is allowed, thus avoiding buffer overflow while achieving high bandwidth utilization. In QK-Push, bitrate and PSNR variation are more stable than DASH2M method. What's more, we can observe that the buffer occupancy is well controlled between dual-threshold in our method, because our method takes the smoothness of video bitrate in the multi-objective optimization decision process into account.

We also find that all the method will request high bitrate level segments between the time of 570sec to 620sec, because we use the VBR encoding method, and this duration video content is cast with a black ground. The encoded segment size will be smaller than the bitrate. Therefore, all method will request high bitrate level segments.

In Table.II, the performance of various methods under fixed bandwidth is compared in terms of four critical QoE metrics, including average video bitrate, average PSNR, bitrate instability, and startup delay. We can find that the HTTP/2-based methods will achieve higher average video bitrate and PSNR,

TABLE II
PERFORMANCE UNDER FIXED BANDWIDTH

Scheme	Average PSNR (dB)	Average video bitrate (kbps)	Average Instability	Startup delay (sec)	eMOS
HTTP/1.1	40.50	1535.20	0.0162	5.233	3.94
HTTP/2.0	40.82	1680.39	0.0127	4.873	3.80
DASH2M	41.43	1986.34	0.0516	2.798	3.59
QK-Push	41.79	2089.67	0.0083	2.098	4.21

because the *Server Push* feature can increase the link utilization. For pursuing target buffer, DASH2M will achieve the highest instability. As adopting of *k-push*, HTTP/2.0 will be more stable than HTTP/1.1. However, our QK-Push holds the lowest instability as considering the smoothness of video bitrate. The startup delay for HTTP/1.1 method is the highest, as the client needs to send a request for each segment to the server, which is time-consuming. Our QK-Push achieve the highest average bitrate, PSNR, eMOS and the lowest startup delay as the use of our QoE-driven adaptive *k-push* scheme and *fast start mechanism*.

C. Impact of RTT Variations

In this part, the experimental results under different RTT (0ms, 50ms, 100ms, and 150ms) with fixed bandwidth and the 1sec segment duration are presented.

As shown in Fig.7, we can find that with the RTT getting bigger, the average bitrate will get smaller in the HTTP/1.1 method, which is more sensitive than others methods. As the push method can counteract the influence of RTT in one push cycle, the other HTTP/2-based methods' performance is nearly stable. Comparing HTTP/1.1 with HTTP/2.0 method, we can find HTTP/2-based methods can take full advantage of the available bandwidth by HTTP/2 *Server Push* feature.

When the RTT gets big, the instability for HTTP/1.1 will be small, however, the other HTTP/2-based method's performance nearly stable. Because HTTP/1.1 is sensitive to bandwidth variation, although we control the bandwidth at 1.9Mbps, there are many spikes during whole bandwidth variation, as the RTT get big, the longer overhead time for one segment will make the client overcome these spikes. For DASH2M, it is unstable and holds big instability, as it is a target buffer based method, and it will request a segment, whose bitrate is big or small than last one. On the contrary, our

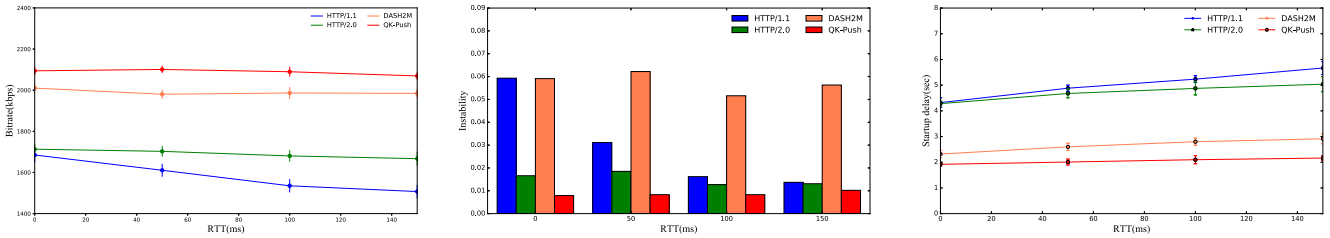

Fig. 7. Impact of different RTT (0ms, 50ms, 100ms, 150ms) under fixed bandwidth with 1sec segment duration environment. (a) Average video bitrate. (b) Average instability. (c) Startup delay.

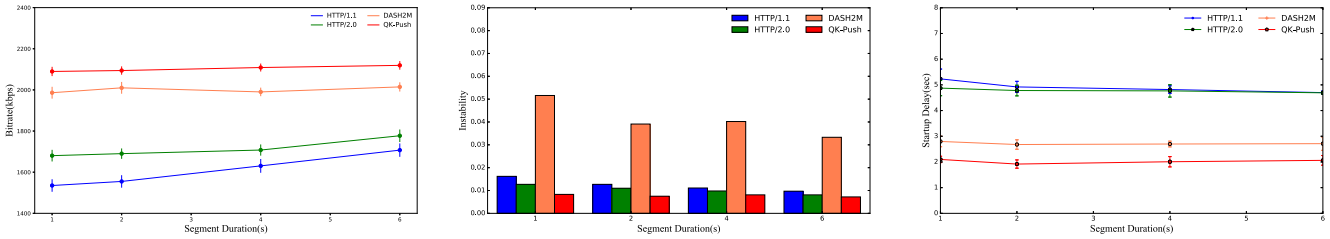

Fig. 8. Impact of different segment duration (1sec, 2sec, 4sec, 6sec) under fixed bandwidth with 100ms RTT environment. (a) Average video bitrate. (b) Average instability. (c) Startup delay.

method is designed with dual-threshold buffer and we consider the bitrate oscillation, so it is more stable.

Statistics suggest that the startup delay of our QK-Push method is the lowest. Because we adopt *fast start mechanism* when the client sends the MPD request with *Fast-Start Directive*, the server will send the client MPD file with *Fast-Start ACK Directive* and some pushed segments, which saves the number of requests and RTT time. For DASH2M, it will request the lowest quality bitrate segment first, so the live latency will be lower than the HTTP/1.1 and the HTTP/2.0 methods. Comparing the HTTP/1.1 with the HTTP/2.0 method, we can find the startup delay is nearly equal when the RTT is small. However, when the RTT gets big, the HTTP/2.0 method can slow down the startup delay compared to the HTTP/1.1 method, because the client needs to buffer some segments before starting to play the video content for all methods. The HTTP/2.0 method will save some requests by HTTP/2 *Server Push* feature, which means the HTTP/2.0 method save some RTT time for these saved requests, so when the RTT gets big, it will save more time.

D. Impact of Segment Duration

In this part, the experimental results under different segment duration (1sec, 2sec, 4sec, and 6sec) with fixed bandwidth and the 100ms RTT are presented.

As shown in Fig.8, we find that as the segment duration gets big, the average bitrate will change into high in the HTTP/1.1 method, and it is more sensitive than other methods. For instance, when the segment duration is 6sec, the HTTP/1.1 method will be similar to the HTTP/2.0 method (where $k = 6$ for *k-push* and segment duration is 1sec), which will save about 83% requests number for a certain duration video and have a higher link utilization. For other methods, they can offset the influence of RTT in one push cycle, when the segment duration gets big, the saved requests number will be small compared to the HTTP/1.1 method. Comparing the HTTP/1.1 with the HTTP/2.0 method, we can find the HTTP/2-based methods can take full advantage of the available bandwidth by HTTP/2 *Server Push* feature when the segment duration is small, which will decrease the startup delay.

We can find that all methods hold low instability level when the segment duration gets big. Because when the segment duration gets big, the client will need a small number of segments for a certain video, which means we need more overhead time for one segment. This will make the client omit some bandwidth variation spikes and decrease the instability.

For our method, the startup delay is the lowest by adopting the *fast start mechanism*. The influence of the segment duration for HTTP/2.0 and DASH2M methods is very small because they only need one or two push request(s) to start playing the video content with the variation of the segment duration. For the HTTP/1.1 method, the different segment duration means different requests number to start playing the video content, so its startup delay is more sensitive than segment duration. No doubt that the startup delay for our method is the lowest all the time as adopting our *fast start mechanism*.

E. Performance Under Square Bandwidth

In this part, the experimental results under the case that the available bandwidth goes through some positive or negative spikes that last for few seconds are compared in Fig.9, where the segment duration is 1sec and the RTT is 100ms.

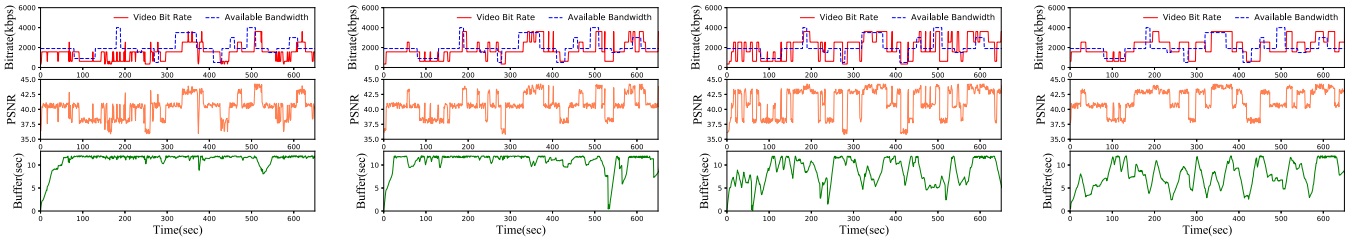

Fig. 9. Performance under square bandwidth with 100ms RTT and 1sec segment duration environment. (a) HTTP/1.1 (b) HTTP/2.0. (c) DASH2M. (d) QK-Push.

When happening short-term variations, which are common in practice, especially in wireless networks, a good rate adaption method is able to compensate for such spikes using its buffer, without causing short-term bitrate switchings. As shown in Fig.9, when the available bandwidth changes (such as 540sec to 560sec), HTTP/1.1 method will often change the segments' bitrate level. However, the HTTP/2.0 method will not switch the quality as the HTTP/1.1 method, because the HTTP/2 *k-push* mechanism will skip some bandwidth variations. Compared to these bandwidth-based methods, DASH2M method will always chase a goal buffer occupancy, so it will often switch the quality. On the contrary, our method takes the smoothness of video bitrate in the multi-objective optimization decision process into account. We can observe that the short-term spikes are well absorbed via buffer accumulation/consumption, without causing short-term bitrate fluctuations. Thus, in our QK-Push method, the instability metric always stay at a low level.

When happening long-term variations, all schemes switch the requested video bitrate so as to prevent the buffer from overflow/underflow. However, when the bandwidth decreases severely from 1.9Mbps to 0.5Mbps, in DASH2M method and bandwidth-based methods, different levels of buffer depleting happen. This is because that when the bandwidth is stable, a large push length is selected to reduce requests in bandwidth-based methods, which will lead to dramatical buffer dropping for unpredictable severe bandwidth deterioration. In DASH2M method, to improve playback quality, the maximal video bitrate is selected under constraints that the estimated buffer is no less than given target level. However, network bandwidth can be highly dynamic which is hardly predicted accurately, thus buffer depleting happens for unpredictable severe bandwidth deterioration. In contrast, the buffer occupancy with our method is well controlled via quickly switching down requested video bitrate. This is because that in our method, when buffer occupancy falls below threshold $q_{\min}$ where the risk of buffer depleting is high, the running push session is terminated immediately and a new push cycle is launched with new adaptive segments to prevent the buffer from underflow. Thus our method will not lead to stalling.

In Table.III, the performance of various methods under short-term bandwidth variations is compared in terms of four critical QoE metrics, including average video bitrate, average PSNR, bitrate instability, and stalling ratio. We also can find

TABLE III
PERFORMANCE UNDER SQUARE BANDWIDTH

Scheme	Average PSNR (dB)	Average video bitrate (kbps)	Average Instability	Stalling ratio (%)	eMOS
HTTP/1.1	40.11	1445.80	0.0534	0.094	2.83
HTTP/2.0	40.73	1739.79	0.0323	0.161	3.09
DASH2M	41.20	2011.29	0.0511	0.816	2.98
QK-Push	41.67	2114.49	0.0197	0	3.62

that the HTTP/2-based methods will achieve higher average video bitrate and PSNR by the HTTP/2 *Server Push* feature. The instability for DASH2M is the highest, as it pursues a target buffer. And the HTTP/1.1 method also has high instability because it is a bandwidth-based method, which is easily influenced by the bandwidth variation. The stalling ratio for HTTP/1.1 method is small, as the selected video bitrate is never allowed to be higher than the available bandwidth. For the HTTP/2.0 method, it adopts *k-push* and it will slowly react to bandwidth variation, so it will have a high stalling ratio when the available bandwidth is fluctuant. To pursue a target buffer level, DASH2M will always request segments, whose bitrate mismatches the available bandwidth, which will lead to big instability. As a result, DASH2M will easily stall when it faces drastic fluctuation in available bandwidth. However, our method will not cause stall as we adopt a probabilistic buffer model and a *push cancel mechanism*. Finally, our QK-Push achieves the highest eMOS.

F. Performance Under Real Internet Trace

In this section, we first compare all the methods under a real Internet trace (about 670sec) depicted in Fig.10, where both the long-term shifts and short-term fluctuations of bandwidth can be observed. All Internet trace is collected in PlanetLab, with one node located in Hong Kong, China (plab1.cs.ust.hk) and another node in Beijing, China (p11.pku.edu.cn) for 4 hours from 7:00 pm to 11:00 pm.

The results depicted in Fig.10 demonstrate our proposed method well adapts to the varying network conditions. The results show that our proposed method can effectively absorb short-term spikes using buffered video, without causing short-term bitrate switchings. While for long-term bandwidth changes, our proposed method is able to switch to an appropriate bitrate without causing buffer overflow/underflow or playback stall. However, in bandwidth-based methods, due to

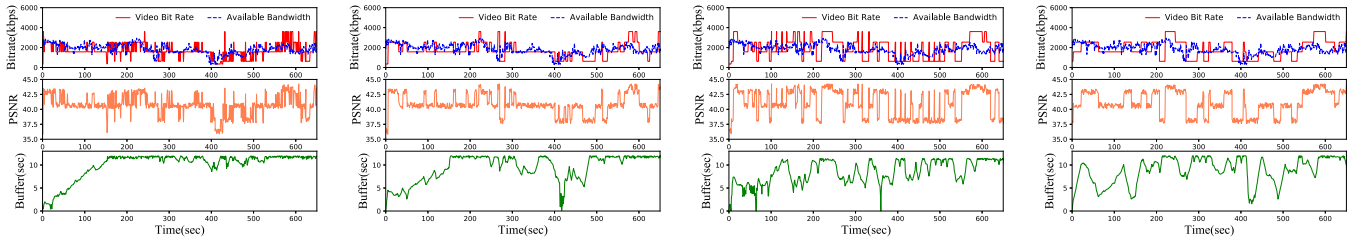

Fig. 10. Performance under real Internet trace with 100ms RTT and 1sec segment duration environment. (a) HTTP/1.1. (b) HTTP/2.0. (c) DASH2M. (d) QK-Push.

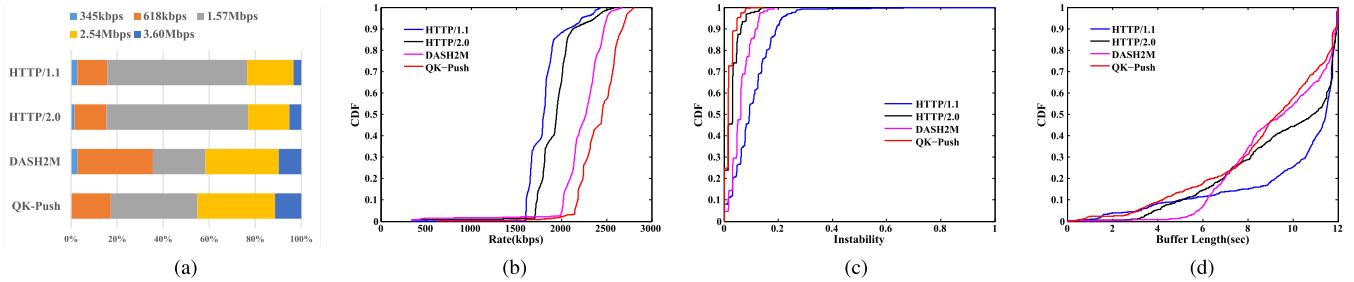

Fig. 11. Performance under real Internet trace with 100ms RTT and 1sec segment duration environment. (a) Proportion of video bitrate level. (b) CDF of average video bitrate. (c) CDF of instability. (d) CDF of buffer occupancy.

TABLE IV
PERFORMANCE UNDER REAL INTERNET TRACE

Scheme	Average PSNR (dB)	Average video bitrate (kbps)	Average Instability	Stalling ratio (%)	eMOS
HTTP/1.1	40.72	1676.14	0.0990	0.181	3.27
HTTP/2.0	40.82	1707.51	0.0282	0.243	3.37
DASH2M	40.94	1812.12	0.0586	0.954	2.96
QK-Push	41.29	1876.85	0.0184	0	3.58

its conservative bandwidth-based adaption logic, the video bitrate fluctuates frequently and the buffer occupancy always stays at high level. In DASH2M, the bitrate smoothness is sacrificed so as to provide high playback quality while stabilizing buffer occupancy. However, the sacrificing of video bitrate smoothness to stabilize the buffer occupancy usually unbefitting in maintaining QoE. From the viewer's point of view, visual quality degradation due to bitrate fluctuations are more perceivable than buffer occupancy oscillations.

The comparison of quality metrics is shown in Table.IV, which shows that our method provides a continuous video playback with the most smooth bitrate, highest quality level and eMOS. And the HTTP/2-based methods will achieve higher average video bitrate and PSNR by the HTTP/2 *Server Push* feature. We find that HTTP/1.1 holds the highest instability because the real internet trace is time-varying. For DASH2M, it will always pursue a target buffer, so it will have high instability. HTTP/2.0 method can overcome some bandwidth spikes by *k-push*. However, by considering the smoothness, our QK-Push will hold the lowest instability. Stalling events will not occur as our probabilistic buffer model and *push cancel mechanism*. As the selected video bitrate is never allowed to be higher than the available bandwidth, the stalling ratio for HTTP/1.1 method is small. HTTP/2.0 method will slowly react to bandwidth variation as this method will push k segments which hold the same bitrate, so it will have a high stalling ratio compared with the HTTP/1.1 method. DASH2M method has the highest stall ratio. Because, DASH2M will always request segments which mismatch the available bandwidth to pursue a target buffer level, which will lead to high instability when the available bandwidth fluctuates.

Then, we test our proposed scheme in the long Internet trace (approximately 8500sec). To make a fair comparison, we then sequentially conduct 3 group of experiments. In each group, different schemes are tested sequentially by running 650sec. The results of different methods are summarized in Fig.11.

Results show that QK-Push method always provides smooth video bitrate with the high-quality level (3.60Mbps, about 11%, 2.54Mbps, about 34%, 1.57Mbps, about 38%, as shown in Fig.11 (a)) since our method takes those critical factors affecting QoE into account comprehensively and thoroughly. However, other three schemes only consider part of critical QoE factors and thus fail to achieve the best performance. Besides, though with *k-push*, the average video bitrate in bandwidth-based methods are always lower for its conservative bandwidth-based rates selection logic, such as the proportion of HTTP/1.1 and HTTP/2.0 are mainly concentrated on the 1.57Mbps (HTTP/1.1, about 61%, HTTP/2.0, about 62%) and 2.54Mbps (HTTP/1.1, about 20%, HTTP/2.0, about 18%).

In Fig.11 (b), we calculate the Cumulative Distribution Function (CDF) of average video bitrate for each segment. We can find that our QK-Push method will hold the highest

average video bitrate level by our QoE-driven adaptive *k-push* scheme, and the HTTP/1.1 holds the lowest average video bitrate as it never allows to request segment which video bitrate is higher than the available bandwidth. The HTTP/2.0 and DASH2M methods have a higher average video bitrate as the adoption of the *Server Push* feature.

We also calculate the CDF of instability for each segment in Fig.11 (c), the instability for our QK-Push is the lowest and the HTTP/1.1 method achieves the highest instability as the real Internet trace is time-varying. However, by adopting the *Server Push* feature, the HTTP/2.0 method achieves low instability to some extent. For DASH2M, it also has high instability to maintain the buffer at a target level. By considering the smoothness, our QK-Push still holds the lowest instability.

Fig.11 (d) shows that DASH2M always keeps the buffer near the target buffer, and HTTP/1.1 with HTTP/2.0 keep the buffer at a high level as they are the bandwidth-based method and they will request segments which video bitrate never higher than the available bandwidth all the time. On the contrary, our QK-Push method switches the buffer level between $q_{\min}$ and $q_{\max}$, which avoid buffer overflow/underflow.

VIII. Conclusion

In this paper, we proposed a QoE-driven rate adaptation approach of dynamic *k-push* in HTTP/2 live streaming. To ensure continuous playback, a probabilistic buffer control model was firstly designed to keep buffer occupancy staying between dual-threshold to avoid buffer underflow/overflow. Then, we designed three QoE objective functions and maximized them via comprehensively considering critical factors of QoE for an HTTP/2 live video streaming session. To balance the needs for video quality, bitrate smoothness and request overhead, we cast the above multi-objective optimization problem as a *Pareto problem* via a *Nash bargaining solution*. A *discrete space Lagrangian* method was designed to produce the segments in each push cycle. Furthermore, we also designed a *fast start mechanism* and a *push cancel mechanism*, which provided low startup delay and flexibility for our scheme. Finally, we implemented a real HTTP/2 QK-Push prototype with *dash.js* and modified MPD. To evaluate the performances, the extensive live streaming experiments were carried out over controlled network test-bed and real Internet trace. The results demonstrated good efficiency of the QK-Push algorithm.

References

[1] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2011, pp. 1–8

[2] V. Swaminathan and S. Wei, "Low latency live video streaming using HTTP chunked encoding," in *Proc. IEEE Int. Workshop Multimedia Signal Process.*, 2011, pp. 1–6

[3] V. Swaminathan and S. Wei, "Low latency live video streaming over HTTP 2.0," in *Proc. ACM Netw. Operating Syst. Support Digit. Audio Video Workshop*, 2014, p. 37.

[4] M. Belshe, M. Thomson, and R. Peon, *Hypertext Transfer Protocol Version 2 (HTTP/2)*, document RFC 7540, 2015.

[5] S. Wei and V. Swaminathan, "Cost effective video streaming using server push over HTTP 2.0," in *Proc. IEEE Int. Workshop Multimedia Signal Process.*, Sep. 2014, pp. 1–5

[6] R. Huysegems *et al.*, "HTTP/2-based methods to improve the live experience of adaptive streaming," in *Proc. ACM Int. Conf. Multimedia*, 2015, pp. 541–550

[7] D. V. Nguyen, H. T. Le, P. N. Nam, A. T. Pham, and T. C. Thang, "Adaptation method for video streaming over HTTP/2," *IEICE Commun. Express*, vol. 5, no. 3, pp. 69–73, 2016.

[8] V. N. Minh, P. H. Phong, and P. N. Nam, "A probability-based adaption method for server-pushed streaming over HTTP 2.0," in *Proc. IEEE 6th Int. Conf. Commun. Electron.*, Jul. 2016, pp. 25–29

[9] T. Vu, H. T. Le, N. P. Ngoc, and T. C. Thang, "Adaptive mobile streaming over HTTP/2 with gradual quality transitions," in *Proc. IEEE Global Conf. Consum. Electron.*, Oct. 2016, pp. 1–2

[10] M. Xiao, V. Swaminathan, S. Wei, and S. Chen, "DASH2M: Exploring HTTP/2 for Internet streaming to mobile devices," in *Proc. ACM Multimedia Conf.*, 2016, pp. 22–31

[11] S. Wei, V. Swaminathan, and M. Xiao, "Power efficient mobile video streaming using HTTP/2 server push," in *Proc. IEEE 17th Int. Workshop Multimedia Signal Process. (MMSP)*, Oct. 2015, pp. 1–6

[12] M. Xiao, V. Swaminathan, S. Wei, and S. Chen, "Evaluating and improving push based video streaming with HTTP/2," in *Proc. Int. Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2016, p. 3.

[13] T. Tsujikawa. (2015). *LibngHTTP2_Asio: High Level HTTP/2 C++ Library* [Online]. Available: https://nghttp2.org/documentation/libnghttp2_asio.html

[14] D. I. Forum. (2012), *dash.js* [Online]. Available: http://github.com/DASH-Industry-Forum/dash.js

[15] M. Claeys, S. Latre, J. Famaey, and F. De Turck, "Design and evaluation of a self-learning HTTP adaptive video streaming client," *IEEE Commun. Lett.*, vol. 18, no. 4, pp. 716–719, Apr. 2014.

[16] A. Sobhani, A. Yassine, and S. Shirmohammadi, "A video bitrate adaptation and prediction mechanism for http adaptive streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 2, p. 18, 2017.

[17] M. Levkov, "Video encoding and transcoding recommendations for HTTP dynamic streaming on the Adobe Flash platform," Adobe Syst. Inc, White Paper, 2010. [Online]. Available: http://download.macromedia.com/flashmediaserver/http_encoding_recommendations.pdf

[18] R. Pantos and W. May. (2016). *HTTP Live Streaming*. http://tools.ietf.org/html/draftpantos-http-live-streaming-11

[19] T. Stockhammer, "Dynamic adaptive streaming over HTTP: Standards and design principles," in *Proc. ACM Conf. Multimedia Syst.*, 2011, pp. 133–144

[20] L. Toni, R. Aparicio-Pardo, K. Pires, G. Simon, A. Blanc, and P. Frossard, "Optimal selection of adaptive streaming representations," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 2s, 2015, Art. no. 43.

[21] C. Zhou, C.-W. Lin, and Z. Guo, "mDASH: A Markov decision-based rate adaptation approach for dynamic HTTP streaming," *IEEE Trans. Multimedia*, vol. 18, no. 4, pp. 738–751, Apr. 2016.

[22] B. Rainer, D. Posch, and H. Hellwagner, "Investigating the performance of pull-based dynamic adaptive streaming in NDN," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2130–2140, Aug. 2016.

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

[24] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 187–198, 2015.

[25] M. Watson, "HTTP adaptive streaming in practice," in *Proc. ACM Multimedia Syst. Conf. (MMSys)*, San Jose, CA, USA, Feb. 2011. [Online]. Available: http://www.sigmm.org/archive/MMsys/mmsys11/MMSys11.pdf

[26] B. Zhou, J. Wang, Z. Zou, and J. Wen, "Bandwidth estimation and rate adaptation in HTTP streaming," in *Proc. Int. Conf. Comput., Netw. Commun.*, 2012, pp. 734–738

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

[28] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proc. ACM SIGMM Conf. Multimedia Syst. (Mmsys)*, San Jose, CA, USA, Feb. 2011, pp. 145–156.

[29] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz, "Adaptation algorithm for adaptive streaming over HTTP," in *Proc. Packet Video Workshop*, 2012, pp. 173–178

[30] H. T. Le, D. V. Nguyen, N. P. Ngoc, A. T. Pham, and T. C. Thang, "Buffer-based bitrate adaptation for adaptive HTTP streaming," in *Proc. Int. Conf. Adv. Technol. Commun.*, 2014, pp. 33–38

[31] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[32] D. Stohr, A. Frömmgen, A. Rizk, M. Zink, R. Steinmetz, and W. Effelsberg, "Where are the sweet spots?: A systematic approach to reproducible DASH player comparisons," in *Proc. ACM Multimedia (MM)*, 2017, pp. 1113–1121.
[33] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 325–338.
[34] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner, "Dynamic adaptive streaming over HTTP/2.0," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2013, pp. 1–6
[35] S. Petrangeli, V. Swaminathan, F. De Turckand, and M. Hosseini, "Improving virtual reality streaming using HTTP/2," in *Proc. ACM Multimedia Syst. Conf.*, 2017, pp. 225–228
[36] S. Petrangeli, V. Swaminathan, M. Hosseini, and F. De Turck, "An HTTP/2-based adaptive streaming framework for 360° virtual reality videos," in *Proc. ACM Multimedia (MM)*, 2017, pp. 306–314.
[37] M. Xiao, C. Zhou, V. Swaminathan, Y. Liu, and S. Chen, "Bas-360: Exploring spatial and temporal adaptability in 360-degree videos over HTTP/2," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2018. [Online]. Available: http://infocom2018.ieee-infocom.org/program/main-technical-program
[38] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala, "Quality selection for dynamic adaptive streaming over HTTP with scalable video coding," in *Proc. ACM SIGMM Conf. Multimedia Syst. (Mmsys)*, Chapel Hill, NC, USA, Feb. 2012, pp. 149–154
[39] T. Hoßfeld, M. Seufert, M. Hirth, T. Zinner, P. Tran-Gia, and R. Schatz, "Quantification of YouTube QoE via Crowdsourcing," in *Proc. IEEE Int. Symp. Multimedia*, Dec. 2011, pp. 494–499.
[40] O. Oyman and S. Singh, "Quality of experience for HTTP adaptive streaming services," *IEEE Commun. Mag.*, vol. 50, no. 4, pp. 20–27, Apr. 2012.
[41] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "QDASH: A QoE-aware DASH system," in *Proc. Multimedia Syst. Conf.*, 2012, pp. 11–22.
[42] S. Tavakoli, J. Gutiérrez, and N. Garcia, "Subjective quality study of adaptive streaming of monoscopic and stereoscopic video," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 4, pp. 684–692, Apr. 2014.
[43] T.-Y. Huang, R. Johari, and N. Mckeown, "Downton abbey without the hiccups: Buffer-based rate adaptation for HTTP video streaming," in *Proc. ACM SIGCOMM Workshop Future Hum.-Centric Multimedia Netw.*, 2013, pp. 9–14
[44] N. Cranley, P. Perry, and L. Murphy, *User Perception of Adapting Video Quality*. New York, NY, USA: Academic, 2006.
[45] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori, "DASH fast start using HTTP/2," in *Proc. ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2015, pp. 25–30
[46] T. Zinner, S. Geissler, F. Helmschrott, and V. Burger, "Comparison of the initial delay for video playout start for different HTTP-based transport protocols," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1027–1030
[47] R. K. P. Mok, E. W. W. Chan, X. Luo, and R. K. C. Chang, "Inferring the QoE of HTTP video streaming from user-viewing activities," in *Proc. ACM SIGCOMM Workshop Meas. Stack*, 2011, pp. 31–36
[48] N. Cranley, P. Perry, and L. Murphy, "User perception of adapting video quality," *Int. J. Hum.-Comput. Stud.*, vol. 64, no. 8, pp. 637–647, 2006.
[49] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive," *IEEE/ACM Trans. Netw.*, vol. 22, no. 1, pp. 326–340, Feb. 2014.
[50] J. Kilpi and I. Norros, "Testing the Gaussian approximation of aggregate traffic," in *Proc. 2nd ACM SIGCOMM Workshop Internet Meas.*, 2002, pp. 49–61.
[51] M. Jain and C. Dovrolis, "End-to-end estimation of the available bandwidth variation range," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, 2005, pp. 265–276.
[52] L. Zadeh, "Optimality and non-scalar-valued performance criteria," *IEEE Trans. Autom. Control*, vol. AC-8, no. 1, pp. 59–60, Jan. 1963.
[53] K. Binmore, A. Rubinstein, and A. Wolinsky, "The Nash bargaining solution in economic modelling," *RAND J. Econ.*, vol. 17, no. 2, pp. 176–188, 1986.
[54] B. W. Wah and Z. Wu, "The theory of discrete Lagrange multipliers for nonlinear discrete optimization," in *Proc. Int. Conf. Princ. Pract. Constraint Program.*, 1999, pp. 28–42.
[55] GPAC. (2013). *Dashcast*. [Online]. Available: https://gpac.wp.imt.fr/dashcast

Zhimin Xu received the bachelor's degree in network engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2016. He is currently pursuing the master's degree with Institute of Computer Science and Technology, Peking University, Beijing.

He also was an outstanding undergraduate in Beijing, in 2016. His research interests include dynamic adaptive HTTP streaming over HTTP (DASH) and virtual reality.

He is the Runner-Up of the IEEE International Conference on Multimedia and Expo DASH-IF Grand Challenge Award on Dynamic Adaptive Streaming over HTTP (DASH) in 2017.

Xinggong Zhang received the Ph.D. degree from the Department of Computer Science, Peking University, Beijing, China, in 2011.

He has been an Associate Professor with the Institute of Computer Science and Technology, Peking University, since 2012. His research interests are in modeling, control, and optimization of multimedia networks, video communications, information-centric network, and dynamic adaptive HTTP streaming over HTTP (DASH).

He was a Senior Researcher with the Founder R&D Center, Peking University, from 1998 to 2012, and a Visiting Scholar with Polytechnic Institute of New York University, New York, USA, under the supervision of Prof. Y. Wang and Y. Liu, from 2010 to 2011.

Zongming Guo (M'09) received the bachelor's, master's, and Ph.D. degrees from the Department of Computer Science, Peking University, Beijing, China, in 1987, 1990, and 1994, respectively.

He has been a Professor with the Institute of Computer Science and Technology, Peking University, since 2002. His research interests lie in multimedia streaming, image/video compression, image and video retrieval, watermarking, IPTV, and mobile multimedia.

Dr. Guo is a Senior Member of the Society of Motion Picture and Television Engineers for China and a Senior Member of the China Computer Federation.