DASH2M: Exploring HTTP/2 for Internet Streaming to Mobile Devices

Mengbai Xiao[1,2] Viswanathan Swaminathan[2] Sheng Wei[2,3] Songqing Chen[1]

[1]Dept. of CS
George Mason University
{mxiao3, sqchen}@gmu.edu

[2]Adobe Research
Adobe Systems Inc.
vishy@adobe.com

[3]Dept. of CSE
University of Nebraska-Lincoln
shengwei@unl.edu

ABSTRACT

Today HTTP/1.1 is the most popular vehicle for delivering Internet content, including streaming video. Standardized in 2015 with a few new features, HTTP/2 is gradually replacing HTTP 1.1 to improve user experience. Yet, how HTTP/2 can help improve the video streaming delivery has not been thoroughly investigated. In this work, we set to investigate how to utilize the new features offered by HTTP/2 for video streaming over the Internet, focusing on the streaming delivery to mobile devices as, today, more and more users watch video on their mobile devices. For this purpose, we design DASH2M, Dynamic Adaptive Streaming over HTTP/2 to Mobile Devices. DASH2M deliberately schedules the streaming content delivery by comprehensively considering the user's Quality of Experience (QoE), the dynamics of the network resources, and the power efficiency on the mobile devices. Experiments based on an implemented prototype show that DASH2M can outperform prior strategies for users' QoE while minimizing the battery power consumption on mobile devices.

Keywords

HTTP Streaming; HTTP/2; server push; video delivery;

1. INTRODUCTION

Recent years have witnessed the steady increase of video traffic on the Internet. According to Cisco, consumer Internet video traffic will be 80% of all consumer Internet traffic in 2019, up from 64% in 2014 [1].

In tandem, mobile devices are gaining increasing popularity, thanks to the technology advancement and the ever-decreasing prices. As a result, Internet streaming clients are also experiencing a steady shift from the traditional wired desktops to the wireless mobile devices. Today, mobile video traffic accounts for more than half (55%) of all mobile data traffic [1].

HTTP has been the most popular delivery vehicle for Internet video traffic, given its considerable market penetration and browser support. Video content providers, such as YouTube [9] and Netflix [7], mostly deliver their videos via HTTP. In HTTP based video streaming services, the video data are usually encoded into various bit rate levels, and are further segmented in terms of playback length, such as 1 second. These segments reside on HTTP servers and are delivered via HTTP responses. HTTP streaming and the ISO/MPEG standard is referred as to Dynamic Adaptive Streaming over HTTP (DASH) [28]. DASH is currently widely used on the Internet for delivering video streaming to both wired desktop users and wireless mobile users [25, 26, 18].

Today the most widely used HTTP protocol is HTTP/1.1 [3]. DASH based on HTTP/1.1 can only switch the video quality at the video segment boundary. This demands accurate bandwidth prediction or a long playback buffer to deal with network jitter for achieving best streaming performance. Furthermore, HTTP/1.1 also imposes some unnecessary overhead. For example, a client needs to send a request for every video segment. This leads to increased delay as it takes at least a round-trip time (RTT) before the next segment is delivered.

The limitations of HTTP/1.1 not only impact Internet streaming delivery, but also Internet content delivery in general. They have motivated the development of HTTP/2, pioneered by Google SPDY [22]. As the successor of HTTP/1.1, HTTP/2 has been standardized in RFC 7540 [4] in February 2015. The new HTTP protocol inserts an interpreted layer, where the traditional HTTP requests and responses are dissembled into *frames*, above the transport layer. A typical HTTP session (a request and a response) is represented as a *stream* in this layer. In addition to the frames carrying the HTTP payload, control frames are used to implement new functions, such as *stream termination* and *server push*. More discussions of these new features are in Section 2.

Due to the improved web access performance, HTTP/2 has started to replace HTTP/1.1 and is expected to eventually fully replace its predecessor. This provides an unprecedented opportunity for utilizing HTTP/2 to further improve the Internet video streaming performance. For example, HTTP/2 enables the server to speculatively push HTTP responses to the client without receiving the corresponding requests. In this way, a server can push additional segments back when responding to the client's request for one segment. This potentially can eliminate the time overhead (one RTT) for every segment. Furthermore, HTTP/2 provides a termination mechanism. Thus, it can enable the client to conduct rate adaptation in the middle of a segment, offering better streaming experience to the client.

A few pioneering studies have investigated HTTP/2 for Internet streaming delivery [33, 31, 32, 16, 13]. For example, Mueller et al. [24] found that Secure Socket Layer (SSL) encryption, which is mandatory in SPDY, incurred noticeable overhead. Wei et al. studied the potential of the push mechanism in reducing the live latency [32], eliminating unnecessary requests [31], and saving power in cellular network [33]. In [16], the authors focused on improving the live experience of HTTP streaming by exploiting the HTTP/2

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2964313

features. It adopts *full-push* while additional messages are used to control the video quality adaptation. Cherif et al. [13] also investigated the push mechanism for fast start in the DASH-compliant video streaming.

However, properly leveraging HTTP/2 for streaming delivery also poses some challenges. For example, while utilizing the push mechanism [16, 33, 31, 32, 13], it is critical to determine the push number (the number of segments to be pushed). A large number of segments to be pushed can greatly improve the streaming throughput. However the pushed contents are wasted if the user stops watching the video prematurely. Thus, a fixed number of segments for push, as in [33, 32, 31], risk network resource waste. In addition, setting the server push with the same bit rate until a switch request is sent from the client [16] is not a timely response for adapting to the frequent network fluctuations. Therefore, properly delivering video on HTTP/2 is not straightforward, and an inappropriate use can lead to network resource waste and degraded user experience.

The problem becomes more sophisticated if the client is a mobile device. Today over 40% of cellular traffic is reported as video streaming [14]. Mobile devices are inherently energy-constrained because of the limited battery capacity. As a result, in addition to the concerns that are often critical for streaming services on desktop computers, such as Quality-of-Experience (QoE), power efficiency is of high priority as well.

To address these challenges, in this work, we explore HTTP/2 to improve DASH for streaming to mobile devices and propose DASH2M (Dynamic Adaptive Streaming over HTTP/2 to Mobile Devices). DASH2M aims to optimize the mobile user's streaming experience while minimizing the resource consumption. Instead of a fixed number of segments to be pushed, in DASH2M, the number of segments for pushing is determined dynamically based on the predicted available network resources and the impact of the user's early termination. Compared to prior exploration, DASH2M also enables graceful quality degradation to deal with network fluctuations in the middle of a push cycle. Furthermore, once the prediction deviates from the reality, the HTTP/2 termination is utilized to start a new push cycle.

We design and implement a prototype of DASH2M. Experimental results show that DASH2M can maximize the streaming throughput, user's QoE, as well as the battery power efficiency.

While details are presented in the paper, some highlights of our contributions include the following:

- By exploring HTTP/2 new features, such as server push and stream termination, DASH2M can greatly improve the Internet streaming delivery to mobile users.
- DASH2M carefully schedules the segment delivery by comprehensively considering the QoE required by the user, the network fluctuations, the potential premature termination, and the mobile user's battery consumption.
- The evaluation results show that DASH2M can maximize the streaming throughput, optimize the user's QoE, and improve the power efficiency for mobile devices.

The rest of the paper is organized as follows. Section 2 discusses background and related work. Section 3 presents the design of DASH2M. Section 4 describes the prototype implementation. The results of evaluation is presented in Section 5, and Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

In this section, we briefly introduce the most relevant features in HTTP/2. This is followed by a discussion on HTTP based video streaming over the Internet.

2.1 HTTP/2

HTTP/2 [4] originates from SPDY [22] developed by Google. Cardaci et al. [12] showed that the SPDY protocol slightly outperformed HTTP/1.1 over high latency satellite links. To the applications above the HTTP layer, HTTP/2 shares the same Application Programming Interfaces (APIs) as HTTP/1.1. The new protocol inserts an additional interpreted layer between the application layer and the transmission layer, which breaks the traditional HTTP requests and responses into *frames*. HTTP messages are associated to their own *streams*, which represent individual HTTP requests or responses. By manipulating the frames as well as the streams, HTTP/2 attains some prominent features like stream termination, server push, etc.

Server push: In HTTP/2, a server is able to speculatively push a response to the client prior to receiving a corresponding request. To make the client aware of what all are arriving, the server needs to send frames named *PUSH_PROMISE* in an active stream. The client can then hold the requests with same URLs if the pushed contents are still unavailable, or respond immediately with the pushed data from the cache. The server push is usually applied to push associated resources of an HTML page to effectively reduce page loading time.

Stream termination: Stream termination is achieved by sending a specific *RST_STREAM* frame, either from the sender or the receiver. When an endpoint receives a *RST_STREAM* frame, the active stream delivering that frame is then closed. Though this is an effective solution to cancel any undesired transmission, about half of a RTT is still required.

2.2 HTTP Streaming

One of the most significant characteristics of video streaming over HTTP is the capability of selecting suitable quality level based on various network conditions. Commonly the HTTP based streaming protocol, such as MPEG-DASH, only provides the mechanism supporting the segment-based designation of specific quality level, and the responsibility of choosing the proper bit rate of next segments is left to the client side (video player).

Many prior studies focused on the quality selection algorithms. QDASH [23] integrated a proxy-like bandwidth measurement component to accurately and promptly derive the network bandwidth. The authors also suggested a gradual quality switching algorithm to improve the subjective user-perceived quality. FESTIVE [17] identified three prominent metrics, namely efficiency, fairness, and stability, in HTTP streaming systems. A suite of techniques are developed to help make beneficial trade-offs among these metrics. Huang et al. [15] suggested that it is difficult to accurately estimate the underlying bandwidth above the HTTP layer according to the surveys done on popular video streaming services. In [11], the authors showed the instability problem when there are competing flows. The root cause is due to the ON-OFF activity pattern on the client-side, and a server-based traffic shaper is implemented to eliminate the quality oscillations. Zou et al. [34] investigated that accurate bandwidth prediction at short time scales of a cellular network is possible, and they proposed a scheme based on integer linear programming to maximize the quality of the downloaded segments.

With the emerging mobile platforms, power efficiency is also becoming a major concern for HTTP streaming. GreenTube [19] found the special power profile of cellular network interfaces, which

is the special *TAIL* state in Radio Resource Control (RRC). It maintained a smart cache for reshaping the network traffic and helping the network interface greedily stay in the sleep state. Other studies [30, 33, 20] shared the same methodology but explored different contexts. These existing studies indicate that a prominent challenge is how to balance the trade-off between the magnitude in one download and the probability that the downloaded data are dropped.

3. DESIGN OF DASH2M

Ideally, while watching a video streamed over Internet, a user expects (1) continuous playback (subject to the buffer size constraint), and (2) smooth rate adaptations to the network fluctuations (graceful and timely bit rate switching according to the available network bandwidth). If the receiving device is a mobile device, the user additionally expects (3) a minimum consumption of the battery power, which depends on (a) how the streaming data is delivered, and (b) how much data is received but not watched due to the client's premature termination. To achieve these goals, the media player needs to have an accurate bandwidth prediction.

In this section, first we will present the design considerations of DASH2M followed by the detailed design.

3.1 Overview of DASH2M Design

To optimize the users' streaming experience, DASH2M addresses the above issues as follows.

[Data Delivery vs. Power Consumption] Our previous work [33] showed that the server push has the capability of reshaping network traffic, which is important to network interface power efficiency on mobile devices. Aggregating network transmissions, which can be easily accomplished via server push, helps the network interface (WiFi or 3G/4G) enter sleep mode more efficiently. While pushing all segments in one time saves the most energy, it may also lead to over-pushed contents. On mobile devices, these downloaded but not watched segments mean significant waste of battery power consumption (in addition to the bandwidth waste, which may also mean monetary loss for some mobile users.). Consequently, DASH2M aims to balance the push number for achieving power efficiency on mobile devices.

[Rate Adaptation vs. Network Fluctuations] With the push mechanism introduced in HTTP/2, some prior studies have suggested to eliminate the request overhead via only sending a leading request for a set of segments, which we call a *push cycle*. This is particularly helpful to live streaming that demands a low live latency. For example, k-push [33, 32, 31] suggests to push additional k segments in response to a leading HTTP request. Hence k-push is only capable of switching to a new quality after every k segments. Alternatively, full-push [16] keeps pushing segments at a constant bit rate level until special requests are sent to the server for switching current video quality. While potentially good for a low latency if the segment size is very small, neither k-push nor full-push can support graceful rate switching in the middle of a push cycle to quickly respond to network fluctuations. That is, in a push cycle, only one quality level (i.e., bit rate) can be used. Instead, DASH2M aims to enable fine quality control by allowing bit rate switching even in a push cycle, swiftly responding to the network fluctuations.

[Bandwidth Estimation vs. Prediction Inaccuracy] To enable rate adaptation, DASH2M requires a bandwidth prediction method. In DASH2M, this is done based on local measurement information to estimate the available network resources. Meanwhile, as the available throughput of a stream is also dependent on the server's outgoing bandwidth, the server information can improve the accuracy of bandwidth prediction. Thus, as an option, the server's information is also used to help estimate the future bandwidth whenever possible. Nevertheless, the bandwidth prediction can deviate from the reality, particularly when the prediction period is long. Therefore, DASH2M also utilizes the stream termination feature to cancel all upcoming pushed segments in a timely manner.

To accomplish these goals, DASH2M consists of the following four major components:

- Push Number Determination per Cycle (subsection 3.2): it quantitatively decides how many segments should be pushed in a push cycle.
- Bit Rate Selection per Segment (subsection 3.3): it determines the bit rate of each involved segment in a push cycle according to the bandwidth prediction.
- Bandwidth Prediction (subsection 3.4): it predicts the bandwidth availability in the next push cycle based on local information and optional server information.
- Push Cycle Termination (subsection 3.5): it is a complement method for correcting the inaccuracy of bandwidth prediction.

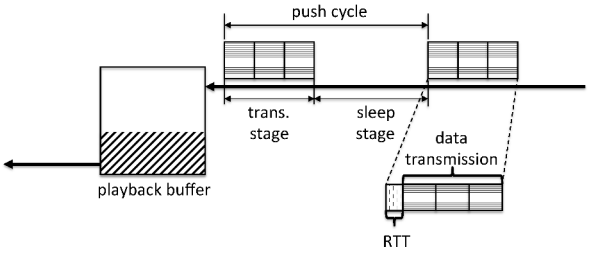

Figure 1: A DASH2M session with server push

DASH2M considers the design of these components based on a push cycle, illustrated by Figure 1. That is, the network activities of a DASH2M session consist of HTTP sessions interleaved with probable idle durations. According to the server push technique, several segment transmissions are bonded together and only one HTTP request overhead is incurred at the beginning of the transmission. As shown in Figure 1, a *playback buffer* stores video data received and relays the data to the display at a constant rate. The buffer accepts data from the network at a varying speed. The network activities are composed of *push cycles*, which can be further divided into *transmission stages* and *sleep stages*. To this end, DASH2M is to launch a push cycle with a push number K, denoting the number of segments involved, and the bit rate arrangement of these segments, represented by

$$\mathbf{b} = (b_1, \ldots, b_K).$$

Next, we present how DASH2M decides the push number K and the bit rates $\mathbf{b}$ in a cycle.

3.2 Push Number Determination per Cycle

Intuitively, a larger push number K can lead to less consumed battery energy. But the amount of pre-fetched segments may unnecessarily drain the battery energy due to potential user abandonment behaviors [27]. Furthermore, with a larger K, the bandwidth prediction period for this push cycle is also longer, which potentially introduces more prediction inaccuracy. Ultimately, K is also constrained by the playback buffer size. Therefore, K cannot be too large and is bound by K_{max}. At a high level, a streaming session always starts with a small K (e.g., 2) for the low startup latency. During the playback, if the predicted bandwidth increases,

Table 1: Variables of A Push Cycle

Symbol	Meaning
K	the segment number in the push cycle
b_i	the bit rate of the i-th segment
D	the segment duration
t_0	the start timestamp of the push cycle
$B(t)$	network throughput of time t
L_0	the buffer length at t_0
L_c	the critical buffer length
S_0	the buffer size at t_0
S_{max}	the maximum buffer size

K will increase accordingly, but not beyond the upper bound. If the predicted bandwidth decreases, K will decrease accordingly.

As K is closely related to the energy consumption, DASH2M dynamically determines K that can minimize the potential energy consumption on the mobile device. To characterize the energy efficiency, we first calculate the overall energy consumption of a push cycle. We characterize the time distribution, which includes T_{cyc} (the push cycle duration), T_{trans} (the transmission stage duration) and T_{sleep} (the sleep stage duration), and combine them with the power profile of network interfaces to calculate the energy consumption. The time distribution of the push cycle, given a K, is

$$\begin{cases} T_{cyc} = L_0 + K \cdot D - L_c \\ T_{trans} = RTT + \frac{\sum_{i=1}^{K} b_i \cdot D}{E[B_i]} \\ T_{sleep} = T_{cycle} - T_{trans} \end{cases}.$$

$E[B_i]$ represents the bandwidth expectation until the i-th segments are transmitted, and Table 1 lists the parameters we use. It is worth noting that during T_{sleep} the network interface may consume different levels of power. We will explain the details next.

With the corresponding power profile, which depends on device types, network types, network operators and etc., we can derive the overall energy consumption of a push cycle for a smartphone as follows:

$$Energy(K) = T_{trans} \cdot p_a + C \cdot p_t + (T_{cyc} - T_{trans} - C) \cdot P_i,$$

when using a 3G/4G network [19], where p_a, p_t and p_i are the power consumption when the network interface is in ACTIVE, TAIL and IDLE states, respectively. C is the duration of the TAIL state. For cellular network interfaces, usually the intermediate TAIL state with a constant duration is desired for promptly returning to the ACTIVE state. The power profiles of other network interfaces (e.g., WiFi) can be presented similarly with slight modifications.

Then the watching energy efficiency can be represented as $\frac{Energy(K)}{K_w}$, where K_w is the watched segment number in this push cycle before abandonment. We define $P\{t \geq T\}$ as the probability a user watches this video for at least T seconds. This probability can be obtained from a general distribution, such as the one suggested in [27], which is characterized by the session duration. We define $\mathscr{E}(\cdot)$ as the watching energy efficiency, and then we can calculate its expectation in a push cycle, for a specific K, as

$$E[\mathscr{E}(K)] = \sum_{i=1}^{K-1} \frac{P\{t_0 + (i-1)D \leq t < t_0 + iD\} \cdot Energy(K)}{i \cdot P\{t \geq t_0\}} + \frac{P\{t \geq t_0 + KD\} \cdot Energy(K)}{K \cdot P\{i \geq t_0\}}. \quad (1)$$

For every push number candidate, we can calculate a corresponding watching energy efficiency expectation from Eq. 1. By comparing these results, a best K leading to the lowest watching energy efficiency is selected as the pushed segment number in the next push cycle. So we have

$$\arg\max_K E[\mathscr{E}(K)] \quad \text{for} \quad K = 1, \ldots, K_{max},$$

where K_{max} is the upper bound of K. As mentioned before, K_{max} is constrained by a fixed duration, after which the bandwidth prediction is believed unreliable. In addition K_{max} also helps to limit the space of candidate Ks. A constant K^C_{max} is helpful in reducing complexity of searching optimal K at the beginning stage of a video session, where the probability of users stopping is high. The prior study by Shafiq et al. [27] shows that terrible network dynamics increase the user abandonment rate. So we scale down K_{max} if the quality level of the last segment in the last push cycle is not the highest one. We have

$$K_{max} = \max\left(\frac{K^C_{max}}{2^{|\mathbf{b}| - Index(b^*)}}, 1\right),$$

where b^* is last bit rate level, and $Index(\cdot)$ represents the index of target bit rate. K_{max} is also constrained by how many segments the remaining streaming session is composed of. So we have the maximum K as

$$\min(K_{max}, \lceil \frac{t_e - t_0}{D} \rceil),$$

where t_0 and t_e are the timestamps of now and the end of the video session, respectively.

Once the number of segments involved in next push cycle is determined, an efficient bit rate arrangement for these segments can be determined, as presented in the next subsection.

3.3 Bit Rate Selection per Segment

The goal of bit rate selection for each segment in a push cycle is to maximize the quality of the video for the user, along with several constraints fulfilled for the user's QoE requirements. A set of ideal bit rates should be the highest ones that the network can sustain. Meanwhile, with such bit rate selection, users should be able to watch the video without noticing stalling playback and abrupt quality switches.

As a result, given a K, and a bandwidth prediction function that predicts $B(t)$ (subsection 3.4), our bit rate selection problem is formulated as an *integer linear programming*. If the segment duration is consistent across the video session, we have the objective function as

$$\text{maximize:} \quad \sum_{i=1}^{K} b_i.$$

In practice, the bit rate b_i is usually selected from a discrete set.

During a streaming session, to maximize the QoE experienced by a user, we need to minimize the re-buffering events, abrupt quality degradation. In addition, the memory a mobile device provides for a streaming session is not unlimited [21]. Therefore the above objective function is subject to several constraints:

Continuous playback: To avoid playback stalls, the time used to transmit the next segment should be less than current buffer length. In a DASH session based on HTTP/1.1, we have

$$T_{req} + T_{resp} < L_0,$$

where T_{req} and T_{resp} are the transmission times of the request and the response, respectively. In a push cycle powered by HTTP/2, only one request is required for multiple responses. If the time to transmit the request payload is considered trivial and the buffer length is extended by a segment duration whenever a segment has been downloaded, the constraint now becomes

$$RTT + \frac{\sum_{i=1}^{n} b_i \cdot D}{E[B_i]} < L_0 + (n-1)D \quad \text{for} \quad n = 1, \ldots, K.$$

Smooth playback: Existing research [23] shows that gradual quality adaptation is more favored in improving subjective perception experience. Therefore we smooth the video quality adaptation in two dimensions, namely quality duration and quality variance. In a quality degradation process, if the duration of an individual video quality is too short, users can still perceive an abrupt quality degradation even when every intermediate bit rate is traversed. So every segment should be in a group of continuous segments at the same bit rate, and the group size, denoted as m, is sufficiently large before moving on to next higher or lower quality group.

To formalize such QoE requirements, a bit rate arrangement in a push cycle can be expressed a vector of J two-tuples, $\langle \hat{b}, n \rangle_1, \ldots, \langle \hat{b}, n \rangle_J$, where $J \leq K$. So the quality duration constraint can be expressed as

$$\begin{cases} n_1 + n^* \geq m & n^* < m \\ n_1 \geq m & n^* \geq m \\ n_j \geq m & 1 < j < J \end{cases},$$

where n^* and $\hat{b}^*$ are the group size and the bit rate inherited from the last group of last push cycle. To represent the quality variance constraint, we have

$$\begin{cases} \hat{b}_1 = \hat{b}^* & n^* < m \\ \hat{b}_1 \in \{N(\hat{b}^*), \hat{b}^*\} & n^* \geq m \\ \hat{b}_j \in \{N(\hat{b}_{j-1}), \hat{b}_{j-1}\} & j > 1 \end{cases},$$

where $N(\cdot)$ represents the neighboring bit rate levels.

Maximum buffer size: Usually the buffer size of a media player on a mobile device is bounded [21]. Every time a segment is downloaded and is put into the buffer, the total buffer size should not exceed the watermark upper bound. We denote the buffer size as S. Then the downloaded segments are subject to

$$S_0 - S_p + \sum_{i=1}^{n} b_i \cdot D < S_{max} \quad \text{for} \quad n = 1, \ldots, K,$$

where S_p is the data amount sent to the decoder.

By solving this integer linear programming problem, we can get the bit rate selection result for each segment in the cycle.

3.4 Bandwidth Prediction

To effectively utilize the network resources according to the bit rate selection algorithm, DASH2M demands a reliable and accurate bandwidth prediction. In DASH2M, the future bandwidth availability is predicted by two means, a short-term estimation based on local measurement and a long-term prediction by analyzing the population distribution of all users watching the video from the server. The latter is optional because the availability of a server's assistance is not guaranteed.

Local measurement: Previous study [29] shows that the simple prediction based on historical information has the best prediction accuracy. So we use the measured bandwidth information of previous segments to predict the future bandwidth. Specifically, We extract the measured information of last push cycles, and use the average value as the future bandwidth.

However, not all the measurement points are reliable. Only two measurement points, the one measured where the first segment is transmitted, and the one measured at the end of the whole push cycle are selected for a push cycle. The reason for not choosing other points is due to potential inaccurate information associated with them. For example, a segment may already be in the cache (in a web browser context), and it only takes around ten milliseconds to complete this operation.

If we assume that there is a historical time point only the newer information after that can be used, then there is a maximum effective measurement duration "D_{max}", ending at now. We then have a set of data points available: $B_m(t_1^*), \ldots, B_m(t_u^*)$, where $t_1^* > \ldots > t_u^*$ and $t_u^* > D_{max}$. Correspondingly, we have a weighted function $w(t)$ that $\int_0^{D_{max}} w(t)dt = 1$. Hence, we have

$$B = \sum_{i=1}^{u} B_m(t_i^*) \int_{t_0 - t_{i-1}^*}^{t_0 - t_i^*} w(t_0 - t_i^*)dt,$$

where $t_0^* = t_0$. The weighted function can be defined as

$$w(t) = \frac{2}{D_{max}^2} t.$$

(Optional) Server support: Note that the throughput of a streaming session is also dependent on the server's network bandwidth allocation. A population distribution of all users watching the video along time, combined with the network capacity of the server, is beneficial in predicting the trend of available bandwidth variation in the long term. The bandwidth function provided by the server is

$$B_s(t) = \frac{B^*}{A + \int_{t_0}^{t_0+t} (a(x) - b(x))dx},$$

where B^* is the overall outgoing bandwidth capacity of the server. A is the user number at time t_0. $a(x)$ and $b(x)$ are the user arrival rate and departure rate, respectively.

When the server support is available, we can combine the local measurement and the remote bandwidth information for more accurate prediction. We choose the lower value for a given t as the predicted bandwidth.

3.5 Push Cycle Termination

For a streaming session, a stable state where the video bit rate approximates the available bandwidth means that a relatively small playback buffer is adequate on the client player. So we always expect that the next scheduling operation starts as long as the buffer length is smaller than a critical value L_c, for example, two segments.

However, accurate bandwidth prediction is not always guaranteed. A bandwidth mismatch can be discovered by periodically comparing the measured bandwidth values to the predicted ones. If the real bandwidth is higher than the prediction, the expected video quality and power efficiency are guaranteed. Thus the risk of degrading current streaming session comes from the bandwidth decrease. When a bandwidth mismatch is found, a longer buffer length is helpful in scaling down the video quality gracefully. So a relaunch of the transmission stage to reschedule video quality is preferred when a large bandwidth mismatch is discovered. The bandwidth mismatch can be quantified as

$$Diff(t) = \int_{t_0}^{t} B_m(t)dt - \int_{t_0}^{t} B(t)dt,$$

where $B_m(t)$ is the measured bandwidth function. $\int_{t_0}^{t} B_m(t)dt$ shows the actual consumed network resources, and the second component represents the predicted consumed network resources. We can expect a push cycle to relaunch once $Diff(t)$ reaches a threshold.

To relaunch, DASH2M utilizes HTTP/2 stream termination. It works as follows. After the client sends a request with a push directive, it is expected to receive *PUSH_PROMISE* frames notifying the client what segments are being pushed. Corresponding URLs and stream IDs can be extracted from these *PUSH_PROMISE* frames

and preserved at the client side. These pushed segments will be returned to the application level once the client receives all the content in the promised streams. If a bandwidth mismatch is found, the client can cancel the promised but not complete streams by sending *RST_STREAM* frames associated with the stream IDs.

Procedure 1 *DASH2M segment scheduling in a push cycle*

Input: $B_l(t)$, $B_s(t)$, $Energy[\cdot]$, RTT, m^*, b^*, $P\{\cdot\}$

Determine K_{max}
$E[\mathscr{E}]_{min} \leftarrow \infty$
$B(t) \leftarrow$ synthesizing $B_l(t)$ and $B_s(t)$

for K in $1 \ldots K_{max}$ **do**
  Calculate **b** by integer linear programming
  Calculate $E[\mathscr{E}]$
  if $E[\mathscr{E}] < E[\mathscr{E}]_{min}$ **then**
    $K_{min} \leftarrow K$
    $\mathbf{b}_{min} \leftarrow \mathbf{b}$
    $E[\mathscr{E}]_{min} \leftarrow E[\mathscr{E}]$
  end if
end for

$b^* \leftarrow b[K]$
$m^{**} \leftarrow 1$
for K in $(K_{min}-1) \ldots 1$ **do**
  if $b[K] = b[K_{min}]$ **then**
    $m^{**} \leftarrow m^{**} + 1$
  else
    break
  end if
end for

if $m^{**} = K_{min}$ **then**
  $m^* \leftarrow m^* + m^{**}$
else
  $m^* \leftarrow m^{**}$
end if
return $(K_{min}, \mathbf{b}_{min})$

As shown in Procedure 1, the client first determines the range of push number candidates. Then for each possible K value, a set of bit rates are selected via linear programming. With such designs, the algorithm sketches the segment scheduling in a push cycle for DASH2M.

4. IMPLEMENTATION

To evaluate the performance of DASH2M, we have implemented a prototype. We present the implementation details in this section.

4.1 DASH2M Server Implementation

We build our DASH2M Server on the top of the Jetty Project [5], a Java based HTTP server that has implemented features of HTTP/2. The major functions implemented on the server side include the following.

Streaming Push: The core module of DASH2M is implemented as a *Filter* class. A HTTP header named *PushSegments* is used to indicate a new push cycle. All URLs of the remaining segments in this push cycle are passed in as the value of this field. Once this field is received as part of a request, 1) the server will then build a new stream to send the corresponding response, 2) *PUSH_PROMISE* frames of the pushed segments will also be sent over this stream, 3) the server will also open a stream within a newly launched thread for each pushed segment, 4) the server determines the dependencies of these pushed segments based on when they should be played. These segments are sequentially pushed without affecting the normal playback.

Stream Termination: While the HTTP/2 interfaces that manipulate frames are not exposed to an application level program, we implement a workaround to simulate the HTTP/2 stream termination feature as our client is implemented in javascript. To relaunch a push cycle, we attach a special HTTP header, named *ResetStreams* to request that. If the server receives such a request, before it launches a new push cycle, the server sends the *RST_FRAME* frames to all active streams to close them. The client can expect the failures of all other ongoing sessions when sending the stream termination directive.

4.2 DASH2M Client Implementation

We implement the video player based on the open source project *dash.js* [2], which is a DASH-compliant video player in JavaScript. The video player is packaged as a web-app and deployed on an HTTP server. Our player has three modules, which are the *push-cycle configuration*, the *bandwidth monitoring*, and the *bandwidth prediction* modules as described below.

push-cycle configuration: This module determines both the number and qualities of segments involved in the next push cycle. This component is normally triggered when the playback buffer reaches a critical level, and it is abnormally triggered by the *bandwidth monitoring* module when a bandwidth mismatch is found. This module takes the output of the *bandwidth prediction* module and follows Procedure 1 to generate the configuration of the next push cycle. Then, the player encodes this result in the header of the first request and sends that request to the server for launching a push cycle.

bandwidth monitoring: This module is triggered by the *onProgress* callback of the standard XMLHttpRequest API of the web browser. It is periodically invoked when the player receives the response content. It estimates the instantaneous bandwidth by checking how many bytes have been loaded, and further compares the result to the predicted value generated by the *bandwidth prediction* module. As long as a significant mismatch is found, it updates the prediction function and calls the *push-cycle configuration* module to relaunch a push cycle. The sent request carries the directive to terminate all ongoing streams on the server.

bandwidth prediction: This module outputs the predicted bandwidth. It keeps accepting local measurement results from the *bandwidth monitoring* module and the remote server (optionally) for updating its output. It follows the description in subsection 3.4 to estimate the bandwidth.

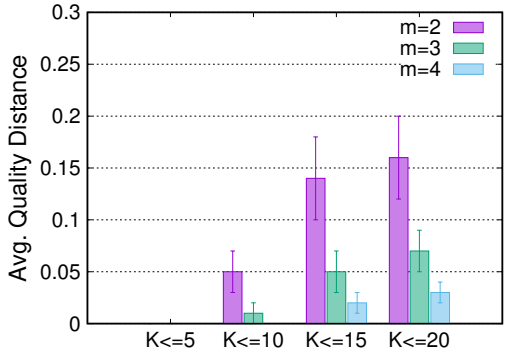

Figure 2: The average quality distance from the time-bound greedy solutions to the optimal solutions

4.3 Scheduling Algorithm Implementation

The core of DASH2M lies in the segment scheduling at the client side as described by Procedure 1. It uses linear programming and is implemented in the Push-cycle Configuration module. When the push number is large, the time used to search the optimal result is impractical because of the huge solution space. So we optimize the linear programming implementation with two heuristics. First, we impose a time constraint on the linear programming implementation. We think a few hundreds of milliseconds is reasonable while not impairing the streaming throughput. Thus, the result is locally optimal since only a part of the solution space is explored. Second, to improve the performance of linear programming, we take advantage of the objective function that is to seek the maximum sum of the bit rate sum. The search always starts from the highest bit rate level when constructing solutions by deepest first search. The greedy algorithm greatly helps find feasible solutions in the limited time with high probability.

5. EVALUATION

To evaluate DASH2M, we run experiments to compare with the standard HTTP/1.1 and the k-push scheme. We actively manipulate the network conditions to observe how the different schemes react to the network fluctuations.

5.1 Experiment Setup

We firstly conduct a series of experiments in a control environment for characterizing the basic features of our scheme. The HTTP server resides on a Linux machine with a 64-bit Intel Pentium CPU 2.8 GHz dual core, 6 GB memory, 2×32 KB L1 caches, 2×256 KB L2 caches and shared 3 MB L3 cache. The installed operating system is Ubuntu 12.04 with Linux kernel 3.13.0-66-generic.

The Jetty server supporting DASH2M and k-push is launched as a normal HTTP server, serving all requests to port 8444 in HTTP/2. It also supports HTTP/1.1 DASH based streaming. The available resources on the HTTP server includes the video player and the video segments, which are packaged by the tool *MP4Box* [6]. The Chrome browser is used to acquire and run the video player. The network shaping rules are imposed on this interface targeting the IP packets that have 8444 as either *dport* or *sport*. Such a setting is to eliminate the impact of other uncertain factors so that we can get a more accurate assessment of DASH2M functions.

5.2 Linear Programming Evaluation

We firstly evaluate the performance of our linear programming implementation. In the experiment, the video bit rate levels are set as 51 kbps, 195 kbps, 515 kbps and 771 kbps, respectively. The segment duration is 2 seconds and the RTT is 50 milliseconds. The buffer length is 10 seconds and we always start with $b^* = 195$ kbps and $n^* = 1$. We search the solutions for $K \leq 20$. We compare our time-bound greedy searching algorithm, which exits after 100 milliseconds, with the optimal algorithm. The bandwidth variation in the future 40 seconds is predictable and the value varies every 10 seconds as follows. The bandwidth candidates are selected from $\{60, 200, 600, 800\}$ kbps randomly so there are 256 combinations to traverse. We repeat the experiment with different m, which is $2, 3, 4$, to observe whether the result differentiates with the different scales of legal solution space.

We define the *quality distance* as the bit rate level difference from the scheduled bit rate to the optimal one. For example, if we schedule a segment with the bit rate of 51 kbps while the optimal one is 515 kbps, the quality distance is 2. We use the average quality distance for specific K to reflect the quality degradation of a solution from the optimal quality arrangement. If we can not get a solution for a specific K, the lowest bit rate level is selected for all segments. We exhaustively search the optimal solutions from the solution space of 4^{20}, and the average size of the solution space is 52425, 9033, and 2867, respectively, corresponding to the increasing m. The time to perform the search is on average 15.58, 2.51, and 0.8 seconds. For the time bound greedy algorithm, the average legal solution space is bound by the limited search times, which are 336, 341, and 343, respectively. We report the average quality distance statistics in Figure 2. The y-axis is the average quality distance with 95% confidence interval, and the x-axis represents different K ranges. The bars in a cluster shows how the quality distance varies when the legal solution space, i.e. m, changes. We can see that both the m and K increments lead to a higher average quality distance, which is reasonable since the m increment means a larger legal solution space and the K increment means a larger solution space. Ultimately, across all parameter sets, the video quality does not obviously degrade due to our time-bound greedy algorithm while the average quality distances are always less than 0.2.

5.3 Rate Adaptation Evaluation

We evaluate how different schemes react to the network dynamics. In particular, we want to assess whether DASH2M can bring in graceful rate adaptation. The video we used in this experiment is two-minute long, and is encoded into four different bit rate levels, which are 51 kbps, 195 kbps, 515 kbps, and 771 kbps, respectively. The segment duration is 2 seconds. We test different push numbers 2, 5, 10, 20 with the k-push schemes. In addition, we also repeat the experiments for regular DASH streaming based on HTTP/1.1, denoted as *DASH* in the figures.

In the regular DASH scheme, the client estimates current bandwidth according to the duration for receiving the last segment, and chooses the maximum bit rate level that are smaller than the measured bandwidth as the quality of next segment to request. The k-push schemes follow the similar rule but estimate the bandwidth by measuring across the entire last push cycle. Meanwhile the network variations are preset in DASH2M, in order to eliminate the impact of bandwidth prediction. We set the watch probability to 1 all the time since the experiment is conducted until the end of video. In this case, DASH2M prefers a large K for the power efficiency. The quality duration m is selected as 4 to smooth the quality variations in the video session.

In all schemes, the critical buffer is set as 20 seconds, which means a new push cycle is launched as long as there are no other active push cycles and the buffer length is below the critical value. We manually introduce the bandwidth drops to result in the bandwidth sequence of {1600, 800, 600, 400, 800, 600, 400} kbps. The bandwidth varies every 15 seconds. The RTT is set as 20 ms all the time, and all schemes start from the lowest bit rate.

Figure 3 shows the request result of a playback session. The x-axis is the time when the requests are sent and the y-axis is bit rates of the requested segments. The bandwidth changes are also plotted as a solid line in the figure. *x-push* presents the k-push scheme with the push number set to x. As we can observe from the figure, the regular DASH scheme performs similar to the 2-push scheme and they can react the bandwidth variation in a timely manner. DASH2M scales up the requested bit rate with a little delay at the beginning because the bit rate switch is gradual in DASH2M. When the bandwidth drops at the 30th and the 90th second, DASH2M firstly exploits the current buffer data to maintain the maximum video quality for a while, and then gracefully decreases the requested bit rate. When the requested bit rate is low at around the 60th second, DASH2M reduces the push cycle and it promptly increases the requested bit rate since the higher bandwidth is discovered. For the other k-push schemes, 5-push and

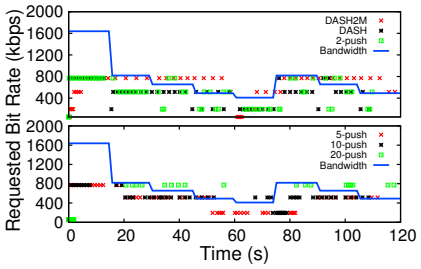

Figure 3: Requested segment bit rate

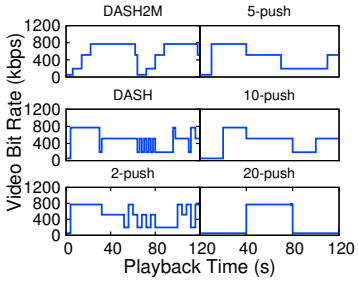

Figure 4: Perceived video quality variations

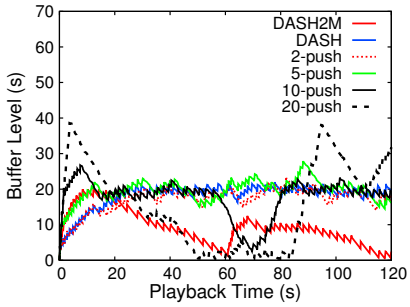

Figure 5: Buffer length variations

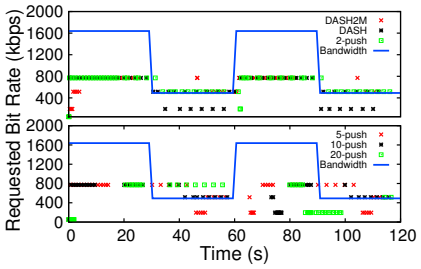

Figure 6: Requested segment bit rate

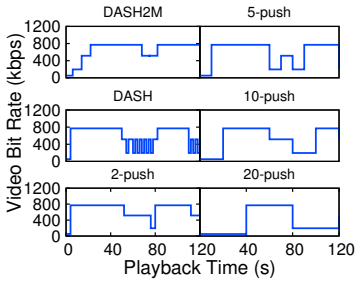

Figure 7: Perceived video quality variations

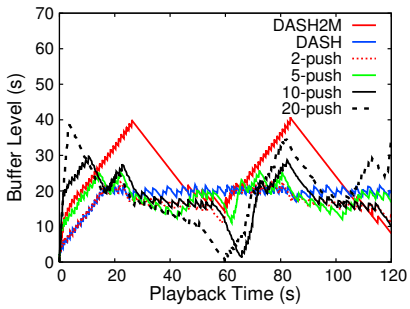

Figure 8: Buffer length variations

10-push fit the bandwidth variation well because their push cycles (10 and 20 seconds) approach the bandwidth variation period. 20-push fails to accommodate the network variations, which are more frequent than its push cycle.

Figure 4 shows how the user perceived quality changes for these schemes. In this figure, the x-axis is the playback time in seconds and the y-axis is the playback video bit rate in kbps. We can see from the figure that DASH2M introduces graceful quality degradation and improvement in response to the network dynamics. While for all other schemes, there are a lot of fluctuations and abrupt quality changes on the client side.

We further calculate the average quality of segments that played on the client. The regular DASH based on HTTP/1.1 has an average bit rate of 485 kbps. In the 2-, 5-, 10- and 20-push schemes, the client received the video at an average bit rates of 489 kbps, 440 kbps, 434 kbps and 293 kbps, respectively. While in DASH2M, the client received the highest average quality across all the schemes, which is 572 kbps.

Figure 5 further shows the buffer variation of all schemes in the above experiment. In this figure, the x-axis is the playback time in seconds, and the y-axis represents the buffer length in seconds. From the figure, we can observe that in the regular DASH scheme and the 2-push and 5-push schemes, the buffer can be maintained at a stable level, indicating that they can react to the network variations in a timely manner. There are large fluctuation when the push number increases. We can see that the 10-push and the 20-push significantly consume the buffer when there is an unpredictable network variation. In the worst case, 20-push triggers a re-buffering at around the 60th second when the bandwidth is the lowest.

On the other hand, with the accurate bandwidth information, Figure 5 shows that DASH2M can always maintain the playback buffer at a low level even when there are rate adaptations, since DASH2M always aims to serve the client with the highest possible bit rates to maximize the video quality.

5.4 Push Termination Evaluation

We have shown that DASH2M outperforms the push schemes with a fixed push number K with graceful quality changes in response to network dynamics. Next we test to see whether the stream termination can also assist DASH2M if the actual bandwidth deviates from the predicted one. That is, when prediction is not accurate. We use the configuration of last experiment except the network variations are set as the sequence of {1600, 600, 1600, 600} kbps. The bandwidth drops occur every 30 seconds.

Figure 6 shows the requested segment bit rates distribution plotted against time. Two deviated points can be found for DASH2M when the bandwidth drops to a low value. This is the time when DASH2M has monitored a significant bandwidth mismatch, and DASH2M immediately sends a directive for terminating all active push streams to the server. DASH2M further decided to switch to a lower quality next to the current bit rate level. The other schemes perform similarly as in the previous experiment that a large K leads to the failure of adapting to the bandwidth fluctuations.

Figure 7 further shows the received video quality at the client. As we can see from the figure, the results are similar to what we have observed in the last experiment. DASH2M still introduces the highest average perceived quality, which is 647 kbps. The regular DASH scheme based on HTTP/1.1 streams the video at an average rate of 586 kbps. The k-push schemes have the average rates of 644 kbps, 561 kbps, 512 kbps and 357 kbps, respectively, following the increasing push number from 2 to 20.

Figure 8 shows the corresponding buffer level variations. Compared to the last experiment, DASH2M accumulates the buffer in a faster manner, which is similar to the k-push schemes with a high push number (e.g., 10 or 20) when the network resources are abundant. However when a bandwidth mismatch is found, DASH2M terminates the segment pushing, which helps the client promptly switch to an appropriate bit rate without consuming the entire buffer. While for 10-push and 20-push, there are re-buffering events due to the drainage of the playback buffer.

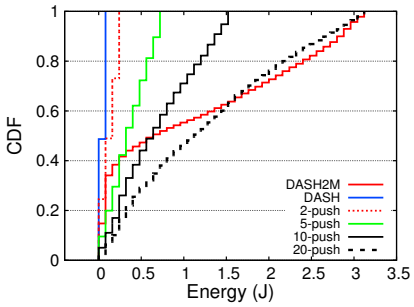

Figure 9: Cumulative Distribution Function (CDF) of energy wasted by over-pushed segments when the ratio of bandwidth to video quality is 10 : 1

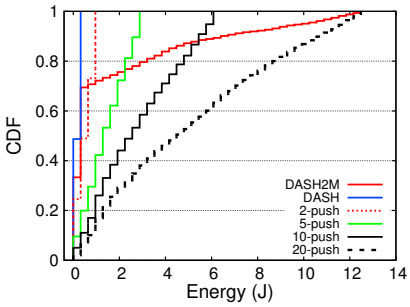

Figure 10: Cumulative Distribution Function (CDF) of energy wasted by over-pushed segments when the ratio of bandwidth to video quality is 10 : 4

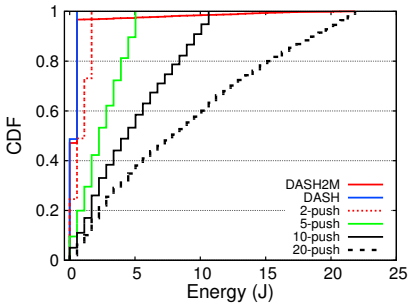

Figure 11: Cumulative Distribution Function (CDF) of energy wasted by over-pushed segments when the ratio of bandwidth to video quality is 10 : 7

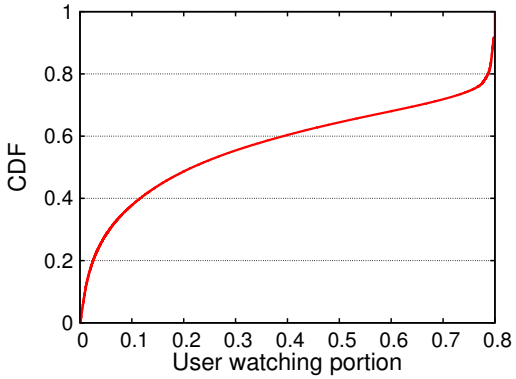

Figure 12: Summarized user watching portion

5.5 Power Efficiency

We also conduct a simulation based on a large scale of HLS trace to evaluate the power efficiency of DASH2M. Our purpose is to find out how many segments are over-pushed and how much energy is wasted. This trace is measured on the mobile devices, which ranges from 07/15/2015 to 08/31/2015, and there are $\sim$ 12 million records from the second largest mobile streaming service provider Vuclip [8]. To study the over-push result, we implement a simulator in perl following the same algorithm described in Procedure 1. In the simulator, we set the segment duration as 2 seconds. In order to retrieve a general distribution of the watching portion, we split the whole data set into two sub-traces with the equal size. One of them is summarized as a normalized distribution, which is represented in Figure 12. This distribution is further built into the simulator. The other sub-trace is fed to the simulator as the input. For evaluating the energy consumption of a push cycle in cellular network, several parameters are required. We set the timeout value between ACTIVE and TAIL as 5 seconds and the timeout value between TAIL and SLEEP as 12 seconds, which are the typical values from AT&T [10]. The average power consumed when transmitting data is set as 800 mW and the TAIL state consumes 400 mW on average, which is measured on ASUS ZenPad 10.

We vary the ratio of the video quality to the network bandwidth for observing how DASH performs under different network utilization. The time used to transmit a video segment is set as 10%, 40% and 70% of the segment duration. We present the simulation results in Figure 9, 10 and 11, respectively. The x-axis is the energy wasted by over-pushed segments and the y-axis is the cumulative distribution function. The k-push schemes and the regular DASH scheme perform almost the same whatever the bandwidth utilization is. The regular DASH scheme rarely downloads the segments that users do not watch, and the segments over-pushed by 20-push waste the most energy, whose value increases with the transmission time. DASH2M aggressively increases the push number when the network resources are abundant, because the transmission time is short and the over-push operations will not waste too much energy. However if the network resources are limited, DASH2M then carefully chooses a smaller push number and no more energy is wasted. Therefore DASH2M is aware of the power profile of current network interface and adopts an efficient transmission solution.

6. CONCLUSION AND FUTURE WORK

As the major delivery mechanism for Internet streaming delivery, HTTP is starting the transition from HTTP/1.1 to HTTP/2, which offers a set of new schemes for improving Internet content delivery. In this work, we systematically explore how to best utilize HTTP/2 to optimize the Internet streaming delivery to mobile devices. For this purpose, we have designed DASH2M, Dynamic and Adaptive Streaming over HTTP/2. DASH2M deliberately considers the network fluctuations and the resource consumption on the mobile devices to optimize the user's streaming experience. To evaluate DASH2M, we have implemented a prototype, and experiments have been conducted in a controlled local LAN environment. The experimental results show that DASH2M outperforms prior strategies by greatly enhancing the user's experience while preserving the energy consumption on the mobile device.

With HTTP/2 playing an increasingly critical role in HTTP streaming, the server-initiated push is a promising technique to effectively improve the streaming experience. While some previous studies have suggested continuous push and shorter segment duration for live streaming, we plan to investigate their effectiveness in practice. A designated short segment duration imposes a small playback buffer to a live stream, which is vulnerable to network fluctuations in Internet. In the future, we expect to practically improve the live streaming experience even with a complicated network conditions.

7. ACKNOWLEDGMENTS

We appreciate constructive comments from anonymous referees. The work is partially supported by NSF under grants CNS-1117300 and CNS-1524462.

8. REFERENCES

[1] Consumer internet traffic report. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[2] dash.js. https://github.com/Dash-Industry-Forum/dash.js/wiki.

[3] Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2068.

[4] Hypertext Transfer Protocol Version 2 (HTTP/2). https://tools.ietf.org/html/rfc7540.

[5] Jetty. http://www.eclipse.org/jetty/.

[6] MP4Box. https://gpac.wp.mines-telecom.fr/mp4box/.

[7] Netflix. https://www.netflix.com/.

[8] Vuclip. http://www.vuclip.com/index.html.

[9] YouTube. https://www.youtube.com/.

[10] A. Gerber et al. A call for more energy-efficient apps, available online. http://www.research.att.com.

[11] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 19–24. ACM, 2013.

[12] A. Cardaci, L. Caviglione, A. Gotta, and N. Tonellotto. Performance evaluation of SPDY over high latency satellite channel. In *Personal Satellite Services*, pages 123–134. Springer, 2013.

[13] W. Cherif, Y. Fablet, E. Nassor, J. Taquet, and Y. Fujimori. DASH fast start using HTTP/2. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 25–30. ACM, 2015.

[14] J. Erman, A. Gerber, K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the top video: the gorilla in cellular networks. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 127–136. ACM, 2011.

[15] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 225–238. ACM, 2012.

[16] R. Huysegems, T. Bostoen, P. Rondao Alface, J. van der Hooft, S. Petrangeli, T. Wauters, and F. De Turck. HTTP/2-based methods to improve the live experience of adaptive streaming. In *Proceedings of the 23rd International ACM Conference on Multimedia Conference*, pages 541–550. ACM, 2015.

[17] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in HTTP-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 97–108. ACM, 2012.

[18] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Quality-adaptive prefetching for interactive branched video using HTTP-based adaptive streaming. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 317–326, New York, NY, USA, 2014. ACM.

[19] X. Li, M. Dong, Z. Ma, and F. C. Fernandes. Greentube: power optimization for mobile videostreaming via dynamic cache management. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 279–288. ACM, 2012.

[20] Y. Liu, F. Li, L. Guo, Y. Guo, and S. Chen. Bluestreaming: towards power-efficient internet p2p streaming to mobile devices. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 193–202. ACM, 2011.

[21] Y. Liu, F. Li, L. Guo, B. Shen, and S. Chen. A comparative study of Android and iOS for accessing internet streaming services. In *Passive and Active Measurement*, pages 104–114. Springer, 2013.

[22] M. Belshe et al. SPDY Protcol. https://tools.ietf.org/html/daft-ietf-httpbis-http2-00.

[23] R. K. Mok, X. Luo, E. W. Chan, and R. K. Chang. QDASH: a QoE-aware DASH system. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 11–22. ACM, 2012.

[24] C. Mueller, S. Lederer, C. Timmerer, and H. Hellwagner. Dynamic adaptive streaming over HTTP/2.0. In *2013 IEEE International Conference on Multimedia and Expo*, pages 1–6. IEEE, 2013.

[25] B. Rainer and C. Timmerer. Self-organized inter-destination multimedia synchronization for adaptive media streaming. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pages 327–336, New York, NY, USA, 2014. ACM.

[26] M. Ryu and U. Ramachandran. Flashstream: A multi-tiered storage architecture for adaptive HTTP streaming. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 313–322, New York, NY, USA, 2013. ACM.

[27] M. Z. Shafiq, J. Erman, L. Ji, A. X. Liu, J. Pang, and J. Wang. Understanding the impact of network dynamics on mobile video user engagement. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 367–379. ACM, 2014.

[28] T. Stockhammer. Dynamic adaptive streaming over HTTP–: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144. ACM, 2011.

[29] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic HTTP streaming. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 109–120. ACM, 2012.

[30] G. Tian and Y. Liu. On adaptive HTTP streaming to mobile devices. In *2013 20th International Packet Video Workshop*, pages 1–8. IEEE, 2013.

[31] S. Wei and V. Swaminathan. Cost effective video streaming using server push over HTTP 2.0. In *2014 IEEE 16th International Workshop on Multimedia Signal Processing*, pages 1–5. IEEE, 2014.

[32] S. Wei and V. Swaminathan. Low latency live video streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 37. ACM, 2014.

[33] S. Wei, V. Swaminathan, and M. Xiao. Power efficient mobile video streaming using HTTP/2 server push. In *2015 IEEE 17th International Workshop on Multimedia Signal Processing*, pages 1–6. IEEE, 2015.

[34] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha. Can accurate predictions improve video streaming in cellular networks? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 57–62. ACM, 2015.