Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption

Abhijit Mondal
IIT Kharagpur, 721302 INDIA
abhimondal@iitkgp.ac.in

Satadal Sengupta
IIT Kharagpur, 721302 INDIA
satadal.sengupta@iitkgp.ac.in

Bachu Rikith Reddy
IIIT Guwahati, 781001 INDIA
rikith.legend@gmail.com

M.J.V. Koundinya
IIIT Guwahati, 781001 INDIA
jwalaiiitg@gmail.com

Chander Govindarajan
IIT Kharagpur, 721302 INDIA
chandergovind@gmail.com

Pradipta De
GSU, GA 30460, USA
pde@georgiasouthern.edu

Niloy Ganguly
IIT Kharagpur, 721302 INDIA
niloy@cse.iitkgp.ernet.in

Sandip Chakraborty
IIT Kharagpur, 721302 INDIA
sandipc@cse.iitkgp.ernet.in

ABSTRACT

YouTube has emerged as the largest player among video streaming services, serving QoE-optimized content for users using DASH. Research studies on various aspects of YouTube, especially its streaming service, abound in the literature. However, these works study YouTube streaming from the periphery, and report results based on their understanding of general DASH recommendations. In this study, we explore in depth YouTube's implementation of the DASH client. We identify important parameters in YouTube's rate adaptation algorithm, and study their roles. In a departure from existing literature, we observe that YouTube opportunistically adapts segment length, in addition to quality level, in response to bandwidth fluctuations. We report that this scheme results in a much lower average data wastage ratio ($0.82x10^{-6}$), than reported earlier. We also propose an analytical model, augmented with a machine learning based classifier (with average accuracy of 85.75%), to predict data consumption for a playback session in advance.

ACM Reference format:
Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, M.J.V. Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. 2017. Candid with YouTube: Adaptive Streaming Behavior and Implications on Data Consumption. In *Proceedings of ACM NOSSDAV conference, Taipei, Taiwan, June 2017 (NOSSDAV '17),* 6 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Mobile traffic has exhibited colossal growth over the past half-decade (18-fold over the last 5 years[1]), with video contributing to 60% of the total usage in 2016. Google's YouTube, already a part of the common Internet parlance, has emerged as the largest player in the mobile video market, accounting for 40–70% of total video traffic across most mobile networks[2]. Not surprisingly, YouTube has garnered significant interest in the research community over the years, furnishing studies which explore various aspects of the service – a large majority of which focus on its video playback mechanism. However, the interest in YouTube's video streaming behavior is far from satiated – a phenomenon largely propelled by YouTube's practice of incessant technical evolution.

Existing literature and limitations: Consequentially, the research community has strived to keep pace with YouTube's technical evolution, churning out studies focused on YouTube's streaming behavior during different time periods. Even recently, several works [8–10] have studied YouTube's DASH behavior to analyze the trade-off between quality of experience and wastage of downloaded data. However, such studies have largely looked at YouTube's streaming behavior through the tinted glass of their understanding of the DASH recommendations. The studies have, in essence, focused on the impact of variation in input parameters (such as bandwidth via throttling), on the streaming output (video bit-rate adaptation), treating YouTube's implementation of DASH as a black-box.

Approach in this work: In a departure from existing literature, we, in this paper, adopt a focused approach to understand the internals of YouTube's bit-rate and quality adaptation algorithm. We endeavor to explore YouTube's implementation of the DASH recommendations by studying the interplay between various parameters of YouTube adaptive video streaming. In terms of methodology, we capture a sizeable volume of YouTube traffic traces (~ 500 videos) in the form of HTTP archives (HAR), which give us access to the contents of HTTP request and response message headers; from these headers, we first identify the parameters used by YouTube in their rate adaptation algorithm, followed by controlled experiments to understand their individual roles.

Experimental observations: Our experiments reinforce the earlier reported observation that video quality adaptation is based on buffer size distribution at the YouTube client. However, we also

[1] http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html

[2] https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-november-2016.pdf

NOSSDAV '17, Taipei, Taiwan

DOI: 10.1145/nnnnnnn.nnnnnnn

observe that when encountered with a drop in network bandwidth, YouTube makes an effort to adaptively change segment lengths of the downloaded video chunks, before downgrading video quality – this observation, to the best of our knowledge, has not been reported in any prior work. In fact, we find that YouTube employs an opportunistic approach of joint video quality and streaming rate adaptation, which is similar to the elastic behavior characteristic observed in TCP traffic (§4).

Data wastage during YouTube streaming: Downloaded data can end up being wasted in adaptive bit-rate streaming when, due to a sudden improvement in network conditions, a higher quality video segment is downloaded for viewing, even when a lower quality segment for the same playback duration already exists – the lower quality segment is rendered unproductive. In light of our experimental observations, we ask the following research question: "How does segment length adaptation affect data wastage in YouTube adaptive streaming?" The question is particularly interesting, since many existing works (including [10] recently) have studied data wastage in DASH implementations, including that of YouTube. We experimentally determine the data wastage ratio involved in YouTube adaptive streaming to be around $0.82x10^{-6}$ on an average, which is significantly lesser than values reported by earlier studies.We reason that such overestimations in earlier works stemmed from their incorrect assumption that the segment size remains constant, which led to gross approximations in their data wastage computations.

Model to predict data consumption: Furthermore, we realize that prediction of data consumption (both productive and wasted) even before a video has actually played, can serve as an important parameter for more intelligent streaming in challenging scenarios. Suppose an user is traveling through a zone of irregular connectivity, thereby resulting in fluctuating bandwidth. It can be noted that existing mechanisms, such as [14] and the references therein, can predict the bandwidth fluctuation pattern a priori under many mobility patterns such as predicted urban mobility. The user would ideally wish to watch videos for as long as possible, without sacrificing her quality of experience too much (streaming in lowest available resolution is too extreme for her). Assuming the YouTube client has prior information of these challenges, and can predict data consumption, it can decide upon the most balanced quality level to start streaming in, so that the data download is minimized. To such ends, we propose an analytical model, augmented with a machine learning based classifier, using which one can estimate data consumption even before actually playing a YouTube video, if channel conditions and the initial video quality level are known.

Contributions: In summary, our contributions in this work are:

(1) We illustrate a methodology to study YouTube's adaptive streaming behavior in-depth (§3) – we identify and closely study the interplay among important parameters enabling this streaming algorithm (§4).
(2) Our experiments reveal that YouTube adapts the *segment length* parameter before attempting to adapt video resolution – a phenomenon not reported in the literature (§4.2).
(3) We observe that segment length adaptation leads to much lower values of data wastage on average, than reported by prior studies.

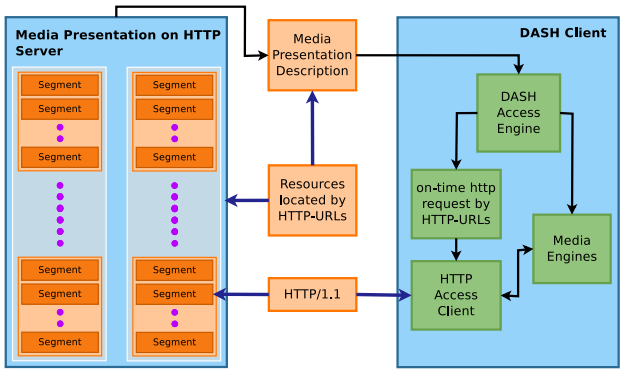

Figure 1: DASH Architecture – On the left side, the server-side media storage is shown, where content is divided into small segments of alternative bit-rates. On the right side, the DASH client architecture is shown; the *DASH Access Engine* monitors network bandwidth at the client and accordingly decides which segment to request from the server. (Image Source: https://www.w3.org/2011/09/webtv/slides/W3C-Workshop.pdf)

(4) We propose an analytical model, augmented with a machine learning based classifier, which enables prediction of data consumption for an initial playback video quality when it is possible to estimate the network conditions a priori using existing mechanism like [14] (§5).

2 BACKGROUND AND PRELIMINARIES

In this section, we provide the reader with a detailed background on YouTube's video streaming strategy.

Evolution of YouTube's playback mechanism: Since its inception in 2005, viewing videos on YouTube required the Adobe Flash plug-in to be installed on the user's browser[3]. In an attempt to reduce third party dependency, and to take advantage of the HTML5 standard which allowed embedded multimedia, YouTube launched an experimental version of the site in January 2010. The service, which encompassed only a section of available videos, was extended to users who opted-in for the trial[4] and were on a browser which supported HTML5 video using WebM or H.264 formats. YouTube experimented with *Dynamic Adaptive Streaming over HTTP (DASH)* around 2013[5] to elevate quality of experience of viewers, before adopting it as the default playback mechanism on January 27, 2015[6].

DASH specifications: Dynamic Adaptive Streaming over HTTP (DASH), also referred to as MPEG-DASH, is an adaptive bit-rate solution for video streaming, which enables client-operated video delivery over HTTP. DASH is implemented by breaking down the video content into small segments, each worth a short duration of playback time. For every segment of playback time, alternative versions at various bit-rates are available at the server. The client typically requests for the highest quality segment possible under current network conditions, such that it is received (downloaded)

[3] http://news.bbc.co.uk/2/hi/8287239.stm
[4] https://www.youtube.com/html5
[5] https://www.youtube.com/watch?v=UkIDSMG9ffU
[6] https://arstechnica.com/gadgets/2015/01/youtube-declares-html5-video-ready-for-primetime-makes-it-default/

in time for playback, without causing stalling or re-buffering. However, DASH is not a protocol – it only specifies an architecture (Fig. 1) to enable adaptive video streaming over HTTP. Every video streaming service (e.g., YouTube, Netflix, etc.) is free to define its own implementation of the DASH modules. In this work, our aim is to study in depth YouTube's implementation of the *DASH Access Engine* module (as seen in Fig. 1).

Encoding technique: YouTube uses the VP9 codec[7], which is a free video codec developed by Google to serve YouTube video. The codec specifies that video files encoded using it shall consist of two types of frames – (1) key-frame, and (2) intra-frame; key-frames contain complete frame information, while every subsequent intra-frame contains incremental information relative to the last seen key-frame. Such specifications mandate that a VP9 decoder can start decoding only at key-frames, an observation we utilize to model data consumption more accurately in §5.

Related Works: Existing studies on YouTube streaming and quality of experience (QoE) can be grouped into two broad classes. The first class of works explore traffic patterns and video QoE properties of YouTube [3, 5–7, 11, 12]. These papers mostly study YouTube behavior at the periphery, which although provides a summary of performance metrics, but fails to say much about the internals of YouTube's video streaming protocol. The second class of studies, however, explore adaptive streaming characteristics of YouTube. In [4], the authors investigate YouTube's data delivery system from the end user view, and illustrate evidence of massive wastage of downloaded data, since viewers often do not watch entire videos – the study, however, was performed at a time when YouTube used progressive download as the streaming mechanism, and is therefore stale. [6] is probably the first work to evaluate YouTube's performance since its adoption of adaptive streaming – the authors claim that YouTube gains 83%-95% in terms of bandwidth by switching from progressive download to DASH. Some recent works [8–10] study YouTube's DASH behavior to analyze the trade-off between quality and data wastage – however, as already pointed out in §1, their approximations lead to gross overestimation. they perform controlled experiments by varying the underlying link bandwidth, and compute wastage.

3 EXPERIMENTAL SETUP

In this study, experiments are conducted with two broad objectives: (1) Observing YouTube's adaptive streaming behavior under different network conditions, and (2) Extracting protocol level parameters from granular video playback data under controlled network conditions, and deducing their individual roles.

Experimental setup for observing streaming behavior: In the preliminary experiment, we play a YouTube video (video title: *"The Division Walkthrough Gameplay Part 1 – The Virus (PS4 Xbox One)"*, video ID: *b80ShWk_Aro*, URL: https://www.youtube.com/watch?v=b80ShWk_Aro, video duration: 34 min. 22 sec.) by varying the link bandwidth using a dynamic throttling mechanism. We refer to this video as our *sample video* throughout the rest of the paper. Link throttling is enabled using the Unix library

Table 1: Information regarding YouTube videos used in the experiments

Video Size	Number of Videos	Total Playback Duration
< 10 mins	195	19h 44m
10 – 30 mins	104	26h 07m
30 – 60 mins	122	94h 33m
> 60 mins	30	37h 15m

`NetFilterQueue`[8], which is a user-space library with an API to handle packets queued by the kernel packet filter. Based on this library, we develop a traffic shaper to control link bandwidth. However, we continuously monitor and ensure that the backbone network has sufficient bandwidth so that the overall link bandwidth is controlled only by the throttling procedure. During the video playback, we capture video data packets using the Unix tool `tcpdump`; from these packet traces, we extract the amount of video data transferred from the YouTube server to the client, with respect to time. Additionally, we note the resolution in which the video is rendered, w.r.t. time.

Experimental setup to identify parameters and their interplay: Since DASH works on top of HTTP, the video playback information and the requested video quality are embedded in HTTP request/response headers. As part of the *developer tools* for Mozilla Firefox, a *network monitor* is available, where the browser dumps information regarding all requests made by the current page – HTTP-request, HTTP-response, request/response time, link speed, etc. The entire information can be exported to **H**TTP **AR**chive (HAR) files, which are in the JSON[9] file format. We develop an automated tool called *AutoHARExporter* using `Selenium`[10] to capture a HAR from the browser with the help of `har_export_trigger` (version 0.5.0-beta) Firefox plugin. This tool automatically opens a Firefox browser, loads a YouTube video (in the format https://youtube.com/watch?v=⟨videoid⟩), waits for the video to finish, and finally saves the HAR and other information to the disk. During video playback, we also control network bandwidth (using the throttling mechanism described earlier) by progressively increasing and then dropping it from a given set of bandwidth levels ranging from 200 Kbps to 2400 Kbps, in a step of 200 Kbps. Each level of bandwidth is kept fixed for 200 seconds. We repeat the experiment for ∼ 500 videos (detailed statistics shown in Tab. 1), the list of which is collected in advance by crawling the YouTube website .

4 IDENTIFYING PARAMETERS INVOLVED IN YOUTUBE ADAPTIVE STREAMING, AND THEIR INTERPLAY

4.1 Parameters involved in YouTube adaptive streaming algorithm

Inspection of the HAR traces obtained using our experimental setup indicates that YouTube uses video playback requests to grab media data from the server. URLs for these video playback requests contain 35 parameters (and their values): pl, dur, $expire$, $sver$, gir, $pcm2cms$, $mime$, $itag$, $signature$, $ipbits$, $source$, $keepalive$, mt, mv, ms, mm, mn, key, $clen$, $requiressl$, lmt, $initcwndbps$, id, upn, $sparams$, $fexp$, ip, cpn, alr, $ratebypass$, c, $cver$, $range$, rn, and $rbuf$. We observe

[7] https://youtube-eng.googleblog.com/2015/04/vp9-faster-better-buffer-free-youtube.html

[8] http://www.netfilter.org/projects/libnetfilter_queue/

[9] http://www.json.org/

[10] http://www.seleniumhq.org/

Table 2: Values of lmt for $itag$ over time and the converted lmt values using epoch converter

itag	lmt	lmt value through epoch converter
242	1454101693286140	2016-01-29 21:08:13.286140
243	1454101777404229	2016-01-29 21:09:37.404229
244	1454101749889990	2016-01-29 21:09:09.889990
250	1454089995528810	2016-01-29 17:53:15.528810
251	1454089993768128	2016-01-29 17:53:13.768128
278	1454101689089978	2016-01-29 21:08:09.089978

Table 3: Evolution of $range$ across rn and the corresponding $itag$, payload is in bytes

rn	itag	range	Payload	rn	itag	range	Payload
0	251	0-65535	65536	6	251	259193-509255	250063
1	244	0-234840	234841	7	244	856823-2396798	1539976
2	244	234841-451095	216255	8	251	509256-739523	230268
3	251	65536-131071	65536	9	251	739524-1113523	3740001
4	244	451096-856822	405727	10	244	2396799-4296833	1900035
5	251	131072-259192	128121	11	251	1113524-1480791	367268

the behavior of these parameters across all the request/response messages that we collected, under various scenarios, as presented below. Note that in the ensuing analysis, a *video session* is defined as a complete rendering of the video on YouTube player.

Parameters constant across multiple videos over multiple sessions: Such parameters do not take part in video adaptation procedure, and only forward some static information, such as the device and OS related information. The parameters alr, c, gir, $iptbits$, $keepalive$, key, mm, mn, ms, mv, $pcm2cms$, pl, $playretry$, $ratebypass$, $requiressl$, $source$ and $sver$ fall under this category.

Parameters constant for the same video across multiple sessions: These parameters are essentially video specific parameters. The parameters cpn, $cver$, ei, $expire$, $fexp$, id, $initcwndbps$, ip, mt, $rbuf$, rn, $signature$, $sparams$, upn and $range$ fit this description.

Parameters changing within a single video session: Out of such parameters, the ones which change only in response to variation in the link bandwidth, possibly take part in the video adaptation process. We figure out that $clen$, dur, $itag$, lmt, $mime$, $rbuf$, rn, $signature$, and $range$ are such parameters. However, on closer inspection, we find the parameters $mime$ and $signature$ relate to audio and video channels. Further the parameter dur denotes video duration, and we observe that it changes only in the order of microseconds, which is due to the change in video encoding technique. Consequentially, we select the rest of the parameters, i.e. $clen$, $itag$, lmt, $range$, $rbuf$, and rn, for more detailed analysis.

***itag*:** YouTube samples every video under different video quality levels based on its resolution, bit rate and encoding techniques used for sampling, and assigns a numeric level to every quality, which is the $itag$ value [1]. Since $itag$ indicates the video quality level, we explore how the other 5 parameters respond to changes in the $itag$ values. We observe that $rbuf$, rn and $range$ change even for a single $itag$; whereas, parameters like $clen$ change overall, but remain constant for a single $itag$ value, indicating that it may have a direct relationship with $itag$.

***lmt*:** Our analysis reveals that the numeric values expressed by this parameter resemble Unix timestamps, although the length is 6 digits longer. The values, converted using *epoch converter*[11], are shown in Tab. 2 for our *sample video* (defined in §3). We find that the date matches with the video upload date, while the time changes slightly. We repeat this experiment for many other randomly chosen YouTube videos, and conclude that lmt defines the time when the chunk was created at the YouTube server. However, it does not participate in video bit-rate adaptation; since YouTube downloads data from multiple servers [6], lmt is possibly used to avoid playing outdated video chunks.

[11] http://www.epochconverter.com/

***rn*:** Our experiments unveil that rn is non-decreasing for a video session. By observing the sequence of HTTP requests sent by the YouTube client to YouTube server, we conclude that rn is the request number to uniquely identify a DASH video playback request.

***rbuf*:** We plot evolution of $rbuf$ for 4 videos and the corresponding $itag$ values in Fig. 2 with varying link bandwidth. In the figure, the lines denote results for the different videos, while color codes are used to distinguish between $itag$ values. Each point in the graph corresponds to a video playback request sent from the YouTube client. We observe that $rbuf$ increases as the YouTube client fetches more data from the server, while it depletes if there is no request from the client to the server. Further, whenever $rbuf$ starts decreasing, the client starts fetching data for a different $itag$ value (as we observe near 1600 secs). Also, as bandwidth increases, $rbuf$ keeps on increasing until it reaches a threshold, and then remains constant. Conversely, as bandwidth drops, $rbuf$ either remains constant or diminishes. The observations obviate that $rbuf$ denotes the *receive buffer* at the client.

***range*:** Tab. 3 presents sample values of $range$ for a particular video. The $range$ values consist of two integers separated by '-'; the first integer is always smaller than the second one – therefore, the two integers indicate $start$ and end of a $range$ value. We also observe that the sizes of the response payloads are given by $(end - start + 1)$. We conclude that $range$ is the byte range parameter in HTTP request header. As a proof of concept, we also perform the following experiment: YouTube provides an API called `get_video_info` through the URL http://www.youtube.com/get_video_info?video_id=⟨video_ID⟩ – the API provides information regarding $itag$ values used in a video, as well as URLs to download a complete video (with unrestricted access), or video chunks (using $range$) for a particular $itag$. Using the API, we download the video chunks of different videos for different available $itag$ values. Then for these videos, we also download different video chunks using the `wget` utility, by changing the values of the $range$ parameter. We observe that MD5 checksums of the downloaded video chunks for a $itag$ value are equal to the MD5 checksum of the byte range of the original video of that particular $itag$. This reinforces our conclusion that $range$ indicates HTTP byte range; YouTube client adaptively changes this parameter to increase or decrease the video chunk size to be fetched, based on network conditions.

***clen*:** In order to interpret the significance of $clen$, we start a browser with developer mode enabled and start monitoring network activities. We render a video in YouTube client and collect the video playback request URLs. We then change the value of the $range$ parameter with 4 different cases where $range$ is equal to (a) $[0, clen]$, (b) $[0, (clen - x)]$ where $0 < x < clen$ is some positive random number, (c) $[0, (clen + x)]$, and (d) with the range parameter omitted. Chunks are fetched from these URLs using `wget` command

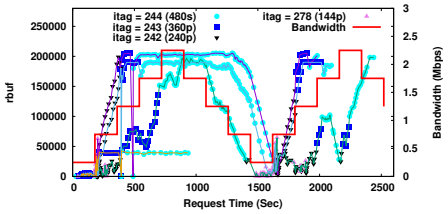

Figure 2: $rbuf$ evolution

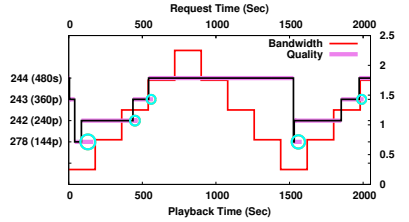

Figure 3: $itag$ evolution

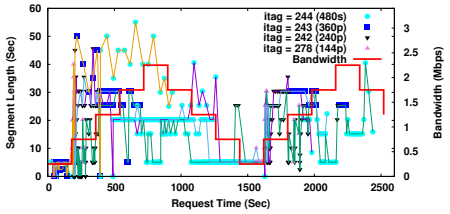

Figure 4: Segment length adaptation

and the lengths and MD5 checksums of the received data segments are checked. We observe that for a particular *itag*, (i) the length and MD5 checksum of the received data are exactly same for the cases when *range* is omitted or *range* is either $[0, clen]$ or $[0, (clen + x)]$ – in such cases, the length of the data is *clen*. However, the length of the data is $clen - x$ when the range value is $[0, (clen - x)]$. We observe similar behavior across all videos that we randomly chose. From these observations, we conclude that *clen* is the length of the video chunk for a particular *itag* value. The YouTube server stores a video chunk with *clen* amount of data for a particular *itag*, and therefore the length of received data never shoots beyond *clen* even if the byte range requested through *range* is higher.

Summary: *range* and *itag* are parameters responsible for adaptiveness in YouTube video streaming, while $rbuf$ is the client-side indicator of channel quality. Although the maximum length of a video segment is defined by *clen* for a particular *itag*, YouTube has the option to download lesser data per request, which is specified by the *range* parameter.

4.2 Insights into YouTube's bitrate adaptation algorithm

Opportunistic (quality) upscale vs. conservative downscale: We concentrate on the nature of YouTube video playback requests whenever it switches from one video quality to another. We first convert the *range* parameter to equivalent video segment length in terms of video playback time – this can be done by looking into the video file header (using the Python package `python-ebml`) that provides a mapping between the byte range and playback time. Fig. 3 plots the video segments (in terms of video playback time, as shown in x-axis) and the corresponding *itag* values for which the client makes a request. We make an interesting observation – whenever the video quality improves, there is an overlap between the segments of lower quality and higher quality (shown using circles in the figure); however, there is no such overlap when the video quality drops. Therefore, we can conclude that the YouTube video adaptation algorithm works in a way where it takes an opportunistic approach for downloading higher quality video segments when the link quality improves, but takes a conservative approach when the link quality drops. In the opportunistic approach, it downloads the video chunks of both the video qualities in parallel, whenever it decides to switch from the lower quality to the higher quality – this often leads to data wastage, as mentioned earlier in §1. However, in the conservative approach, it directly sends the request for lower quality video when the link quality drops.

Segment length adaptation: As mentioned earlier, YouTube adapts both the video quality and the streaming data rate whenever network conditions change. In order to explore the behavior of streaming data rate adaptation, we analyze how the requested video segment length (specified by the *range* parameter) per video playback request, changes with change in link bandwidth. We convert the byte range mentioned in the video playback request to the equivalent video playback time, and find out the video segment length in terms of playback time. The sample results from 4 videos are shown in Fig. 4 where x-axis denotes the video playback request time, and y-axis denotes the segment length in seconds. We use different color codes to indicate the *itag* values associated with corresponding playback requests. The figures give very interesting insights into YouTube video streaming behavior – whenever the link bandwidth increases, YouTube first increases the segment length of lower quality video and buffers maximum amount of video data. It then switches to the higher quality video but with smaller segment lengths. At this point, we observe an overlap between the segments of two different video qualities, as discussed earlier with Fig. 3. Then it progressively increases the segment length and repeats the procedure for the next higher quality level video if the link quality improves further (measured through the increase rate of $rbuf$). However, when the link quality drops, in a similar way, YouTube first starts requesting for same quality video chunks of smaller segments, and drops the segment length. If it still observes a drop in $rbuf$ after reducing the segment length in the playback requests, then it switches to request for the next lower quality level video chunks of smaller segments. Segment length is increased only if $rbuf$ becomes stable. In this manner, YouTube jointly adapts the video quality as well as the segment length (indirectly, the streaming data rate). This observation has not been reported in existing literature, to the best of our knowledge.

Implication on data wastage: Segment length adaptation may have far-reaching implications in terms of advantages for YouTube streaming, one of which is minimal data wastage. Since segment length increases gradually from a low value to higher values when bandwidth improves, overlaps between the segments of a lower quality and the next higher quality are largely diminished. This implies that data wastage values come down drastically (as opposed to a scenario with no segment length adaptation) – in our experiments, we compute the average wastage ratio, defined as $\frac{data\_downloaded - data\_played}{data\_played}$, to be $0.82x10^{-6}$. This is in sharp contrast with previously reported values.

5 PREDICTIVE MODEL FOR DATA CONSUMPTION DURING STREAMING

Let i, $1 \leq i \leq n$ represent an $itag$ value; where n is the total number of $itag$ values used by YouTube. Let λ_i represent the frame rate for $itag$ value i; from our observations, frame rate per $itag$ is constant. Let us also consider an indicator variable $p(t)$, where $p(t) = 1$ denotes that at a particular instance of time (t), a key-frame has arrived. Let α_i be the size of a key-frame (in bits) for $itag$ value i, and β_i be the size of an intra-frame (in bits) at $itag$ value i. Therefore, the amount of data arrived (in bits) in a single frame for $itag$ i, say μ_i, is given by:

$$\mu_i = [p(t).\alpha_i + \{1 - p(t)\}.\beta_i] \tag{1}$$

Since frame rate is λ_i, the amount of data (in bits) downloaded per unit time for $itag$ i, is given by:

$$\delta_i = \mu_i.\lambda_i \tag{2}$$

Let us consider an infinitesinally small time instance dt, during the YouTube streaming process. The amount of data downloaded per instance of time would be: $\delta_i.dt$, for a single $itag$ i. However, data may be downloaded for different $itag$ values at the same time instance, which is where the data wastage stems from. Let us define another indicator variable $q_i(t)$, where $q_i(t) = 1$ indicates that data corresponding to $itag$ i has been downloaded at time t. In such a scenario, the total amount of data (in bits) downloaded in playback time duration τ, is given by:

$$\Delta = \int_0^\tau \left\{ \sum_{i=1}^{n} q_i(t).\delta_i \right\} dt \tag{3}$$

Δ is the measure of *data consumption* due to YouTube playback. Let us also define $f(t)$, which is the maximum $itag$ value at time t, for which data is available (since data may be available for multiple $itag$ values simultaneously):

$$f(t) = max\{i\}\ \forall i\ :\ q_i(t) = 1 \tag{4}$$

Therefore, the data (in bits) actually played by the YouTube player in time τ, is given by:

$$\rho = \int_0^\tau q_f(t).\delta_f\ dt \tag{5}$$

ρ is therefore the measure of *productive data* downloaded during YouTube streaming. Consequently, data wastage ratio (say ω), is defined as:

$$\omega = \frac{\Delta - \rho}{\rho} \tag{6}$$

The key to determining Δ, ρ, and thereby ω, is to estimate the values of q_i for all i, at every time instance t.

Classification Problem: For every $itag$ value i, we consider $\frac{1}{\lambda_i}$ (inverse of frame rate) as the sampling interval, since 1 frame arrives every such interval. We hypothesize that the value of $q_i(\hat{t})$ (where $\hat{t}$ indicates a sampled time instance) depends on the following factors – (1) $\beta(\hat{t}-1)$ (bandwidth at previous instance), (2) $\beta(\hat{t})$ (bandwidth at current instance), (3) $i(\hat{t}-1)$ ($itag$ at previous instance), and (4) $p(\hat{t})$ (presence of key-frame at current instance). We identify that estimating $q_i(\hat{t})$ is a binary classification problem, with every $q_i(\hat{t})$ assuming a value of either 0 (absence) or 1 (presence) of data of $itag$ i at time instance $\hat{t}$.

Model Accuracy: A machine learning based classifier is employed for this purpose – we use Weka [13], a machine learning tool, and select the Random Forest classification [2] technique, with 100 iterations ($I = 100$) and unlimited depth ($K = 0$) as hyperparameters. In the classification technique, we try to predict the value of $q_i(\hat{t})$ using the four parameters mentioned above as classification features. The results (***average precision*** = 0.86***, average recall*** = 0.85***, and average accuracy*** = 85.75%) across all $itag$s indicate high classification prowess, which validates our hypothesis regarding parameters affecting $q_i(\hat{t})$. Using this model, a user can predict the amount of data consumption for YouTube video streaming.

6 CONCLUSION

In this work, we studied the internal working of YouTube's bitrate adaptation algorithm, by identifying important parameters and exploring their roles. We observed that YouTube adapts segment length in addition to quality level, a behavior not been reported earlier. As an implication, we observed that data wastage for a playback session is significantly lower than estimated previously. We further provided an analytical model, augmented with a machine learning based classifier, to predict data consumption in adavance for a video playback session. As an immediate future direction, we would like to explore other important implications of segment length adaption for YouTube.

REFERENCES

[1] 2016. Format and Resolution of YouTube Videos (last accessed: July 2016). (2016). http://www.genyoutube.net/formats-resolution-youtube-videos.html
[2] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[3] X. Che, B. Ip, and L. Lin. 2015. A Survey of Current YouTube Video Characteristics. *IEEE Multimedia* 22, 2 (Apr 2015), 56–63.
[4] Alessandro Finamore, Marco Mellia, Maurizio M Munafò, Ruben Torres, and Sanjay G Rao. 2011. YouTube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM IMC*. 345–360.
[5] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM IMC*. 15–28.
[6] Dilip Kumar Krishnappa, Divyashri Bhat, and Michael Zink. 2013. DASHing YouTube: An analysis of using DASH in YouTube video service. In *Proceedings of IEEE 38th LCN*. 407–415.
[7] J. J. Ramos-munoz, J. Prados-Garzon, P. Ameigeiras, J. Navarro-Ortiz, and J. M. Lopez-soler. 2014. Characteristics of mobile youtube traffic. *IEEE Wireless Communications* 21, 1 (February 2014), 18–25.
[8] Michael Seufert, Florian Wamser, Pedro Casas, Ralf Irmer, Phuoc Tran-Gia, and Raimund Schatz. 2015. YouTube QoE on mobile devices: Subjective analysis of classical vs. adaptive video streaming. In *Proceedings of IEEE IWCMC*. 43–48.
[9] Christian Sieber, Andreas Blenk, Max Hinteregger, and Wolfgang Kellerer. 2015. The cost of aggressive HTTP adaptive streaming: Quantifying YouTube's redundant traffic. In *Proceedings of IFIP/IEEE IM*. 1261–1267.
[10] Christian Sieber, Poul Heegaard, Tobias Hoßfeld, and Wolfgang Kellerer. 2016. Sacrificing efficiency for quality of experience: YouTube's redundant traffic behavior. In *Proceedings of IFIP Networking*. 503–511.
[11] Florian Wamser, Pedro Casas, Michael Seufert, Christian Moldovan, Phuoc Tran-Gia, and Tobias Hossfeld. 2016. Modeling the YouTube stack: From packets to quality of experience. *Computer Networks* (2016).
[12] Florian Wamser, Michael Seufert, Pedro Casas, Ralf Irmer, Phuoc Tran-Gia, and Raimund Schatz. 2015. Poster: Understanding YouTube QoE in Cellular Networks with YoMoApp: A QoE Monitoring Tool for YouTube Mobile. In *Proceedings of the 21st MobiCom*. 263–265.
[13] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. 2016. *Data Mining: Practical Machine Learning Tools and Techniques* (4 ed.). Morgan Kaufmann, Burlington, MA.
[14] Xuan Kelvin Zou, Jeffrey Erman, Vijay Gopalakrishnan, Emir Halepovic, Rittwik Jana, Xin Jin, Jennifer Rexford, and Rakesh K Sinha. 2015. Can accurate predictions improve video streaming in cellular networks?. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. ACM, 57–62.