Dissecting VOD Services for Cellular: Performance, Root Causes and Best Practices

Shichang Xu
University of Michigan

Z. Morley Mao
University of Michigan

Subhabrata Sen
AT&T Labs – Research

Yunhan Jia
University of Michigan

ABSTRACT

HTTP Adaptive Streaming (HAS) has emerged as the predominant technique for transmitting video over cellular for most content providers today. While mobile video streaming is extremely popular, delivering good streaming experience over cellular networks is technically very challenging, and involves complex interacting factors. We conduct a detailed measurement study of a wide cross-section of popular streaming video-on-demand (VOD) services to develop a holistic understanding of these services' design and performance. We identify performance issues and develop effective practical best practice solutions to mitigate these challenges. By extending the understanding of how different, potentially interacting components of service design impact performance, our findings can help developers build streaming services with better performance.

CCS CONCEPTS

• **Networks** → **Mobile networks**; *Network measurement*;

KEYWORDS

Cellular, Adaptive Streaming, VOD, Video Streaming

ACM Reference Format:
Shichang Xu, Z. Morley Mao, Subhabrata Sen, and Yunhan Jia. 2017. Dissecting VOD Services for Cellular: Performance, Root Causes and Best Practices. In *Proceedings of IMC '17*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3131365.3131386

1 INTRODUCTION

Mobile video streaming has become increasingly popular in recent years. It now dominates cellular traffic, accounting for 60% of all mobile data traffic and is predicted to grow to 78% by 2021 [10]. However, delivering good QoE over cellular networks is technically challenging. A recent Internet-scale study indicates that 26% of smartphone users face video streaming QoE problems daily [12].

HTTP Adaptive Streaming (HAS) (see § 2.1) has been adopted for streaming video over cellular by most services including Amazon, Hulu and Netflix. It enables apps to adapt the streaming quality to changing network conditions. To build an HAS service, app developers have to determine a wide range of critical components spanning from the server to the client such as encoding scheme, adaptation logic, buffer management and network delivery scheme. The design involves (i) considering various service-specific business and technical factors, e.g. nature of content, device type, service type and customers' network performance, and (ii) making complex decisions and tradeoffs along multiple dimensions including efficiency, quality, and cost, and across layers (application, network) and different entities. It is thus challenging to achieve designs with good QoE properties, especially given the variable network conditions in cellular networks.

It is important to develop support for developers to navigate this complex design space. Understanding the performance and QoE implications of their design decisions helps developers make more informed and improved designs. Towards this goal, in this work, we conduct a detailed measurement study of 12 popular streaming video-on-demand (VOD) services to develop a holistic understanding of their respective designs and associated performances.

1.1 Contributions

Methodology. The closed, proprietary nature of commercial services makes it very challenging to gain deep visibility into their designs. Approaches like code disassembly suffer from limitations such as code obfuscation. Other approaches that either leverage app-specific features such as URL patterns [19, 28] or rely on deep modifications to the apps [18, 27] cannot be generally applied.

To address these challenges, we develop a general methodology that leverages common properties of commodity VOD apps to derive valuable insights into the proprietary VOD services without access to the source code (§ 2.2). Based on the observation that most popular VOD services adopt well-known HAS protocols, i.e., HTTP Live Streaming (HLS), SmoothStreaming (SS) and Dynamic Adaptive Streaming over HTTP (DASH) [1, 15, 45], we analyze the network traffic and extract useful information regarding the content download process, including timing, quality and size of video chunks downloaded. In addition, detailed analysis of the displayed User Interface (UI) elements for these apps reveals that they use common methods to inform users about the playback, including playback progress and stall events. We therefore develop techniques to extract this information. Correlating the network and UI, our approach is able to effectively extract critical video QoE metrics such as video quality, stall duration, initial delay and number of track switches. In addition, we can infer the apps' internal buffer state which is critical to gain insights into their behavior.

To derive insights into critical aspects of service design such as the adaptation logic, we craft targeted black-box experiments to

IMC '17, November 1–3, 2017, London, UK

ACM ISBN 978-1-4503-5118-8/17/11…$15.00
https://doi.org/10.1145/3131365.3131386

stress-test the apps by emulating various network conditions and manipulating the communication between the client and server (e.g., by altering the manifest file). By analyzing the reaction of the apps, we are able to glean critical properties of their design.

In this paper, we focus primarily on VOD services on the Android platform. However, the measurement methodologies we outline are generally applicable to other platforms (e.g., iOS) and services such as live streaming as they use the same standards (e.g., iOS AV Foundation uses HLS) and substantially similar approaches.

QoE issues and best practices. This paper shows i) the different points in the design space adopted by popular services, ii) the different performance tradeoffs they entail. By examining the absolute and relative performances across different points in the design space, developers are able to get more insights into the implications of design decisions they make, and hopefully make more informed design decisions.

Our measurements cover both individual components across the end-to-end delivery path of HAS and their interactions. This is key to developing insights for better designs across components to realize an overall enhanced QoE. In contrast, different entities involved in the streaming system such as the content provider, ISP and app developers have traditionally possessed only partial views and optimized specific factors somewhat independently, based mainly on their limited views. This can sometimes lead to suboptimal performance as end-to-end QoE is ultimately determined by the interplay across all the different factors. Towards filling this gap, this cross-sectional study across different services develops unique insights by revealing QoE implications of different points in the design space, shedding light on industry best practices by comparing across different services and identifying outlier behaviors.

In this study, we observe interesting behaviors that span a wide range of design decisions and further identify a number of QoE-impacting issues and derive best practices for improvement. We summarize some of the most interesting findings as follows.

- To improve quality, some apps perform Segment Replacement (SR)- replacing a downloaded *segment*[1] with a fresh download for the same position in the video at a potentially different quality. We uncover inefficiencies with existing SR schemes that result in substantial additional data usage, identify root causes, and propose practical SR schemes that achieve better tradeoffs between QoE and data usage (§ 4.1).
- Some services use *Variable Bitrate (VBR) encoding*[1]. However, when determining the next segment to download, they do not account for the substantial size differences across different segments in a *track*[1], which can be a factor of 2 or more. This can lead to suboptimal video QoE. We propose that apps should expose such segment information to the adaptation logic and adopt an actual bitrate aware track selection algorithm (§ 4.2).
- Players typically wait until a minimum number of seconds (i.e., startup buffer duration) of video is fetched before initiating playback. We observe that some apps constantly stall at the beginning of playback when network bandwidth is relatively low, even with observed startup buffer values as other apps which don't exhibit this issue. Our evaluation suggests the need for an additional constraint on when playback should begin – a minimum threshold on the number of segments downloaded (§ 4.3).
- Inadequate synchronization between multiple TCP connections and audio/video downloads can lead to QoE impairments (stalls) for some apps. This highlights the need for better coordination between the parallel download processes for better QoE (§ 3.2).
- A suboptimal buffer-based download strategy waits until the buffer is close to empty, before it restarts downloading. The corresponding app suffered more frequent stalls compared to the others with higher resuming thresholds. Increasing this *resuming threshold* would keep the buffer more occupied and be a practical way to reduce the chances of stalls and provide the client extra headroom to adapt to transient network variability (§ 3.3.2).

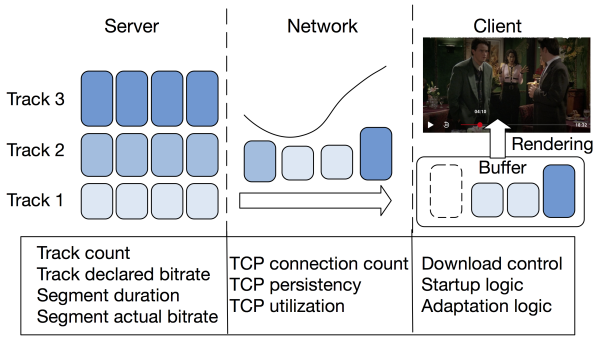

Figure 1: Relevant design factors in HAS service.

2 BACKGROUND AND METHODOLOGY

We provide some background on HTTP Adaptive Streaming and describe our methodology to extract QoE information from the popular VOD services.

2.1 HTTP Adaptive Streaming Overview

Video streaming over the best-effort Internet is challenging, due to variability in available network bandwidth. To address such problems and provide satisfactory QoE, HTTP Adaptive Streaming (HAS) has been proposed to adapt the video bitrate based on network conditions.

In HAS, videos are encoded into multiple *tracks*. Each track describes the same media content, but with a different quality level. The tracks are broken down into multiple shorter *segments* and the client can switch between tracks on a per-segment basis. Media meta-information including the available tracks, segment durations and URIs is described in a metafile called *manifest* or *playlist*.

The manifest specifies a bitrate for each track (referred to as *declared bitrate*) as an estimation of the network bandwidth required to stream the track. Note that this value can be different from the actual bandwidth needed for downloading individual segments especially in the case of *Variable Bitrate (VBR) encoding*. How to set this declared bitrate is left to the specific service, and a common practice is to use a value in the neighborhood of the peak bitrate of the track. In addition to the declared bitrate, some HAS implementations also provide more fine-grained information about segment sizes, such as average *actual segment bitrate*. For services with VBR encoding, as the actual bitrate of segments in the same track can

[1]These terminologies are defined in §2.1

have significant differences, such fine-grained bitrate information can potentially help players better estimate the required network bandwidth to download each track and make more informed decision on track selection. We shall further look into this in § 4.2.

At the beginning of a session, the player downloads the manifest from the server, and uses the HTTP/HTTPS protocol to fetch media segments from the server. Each segment needs to be downloaded completely before being played. To absorb network variance and minimize stall events, the player usually maintains a buffer and tries to fetch segments ahead of playback time. During streaming, the client-side adaptation logic (often proprietary) determines what track to fetch next based on a variety of factors, such as the estimated available network bandwidth and playback buffer occupancy.

There exist a number of different implementations of the above high-level HAS design, involving different file format and protocols. HTTP Live Streaming (HLS) [45], Dynamic Adaptive Streaming over HTTP (DASH) [1] and Smooth Streaming [15] are the most well known of these.

Regardless of implementation details, a wide range of factors spanning the server, the network and the client and across the transport and application layers can be customized based on the system designers' considerations around different tradeoffs to optimize streaming performance. For instance, the client can adopt different track selection algorithms to balance video quality and stalls. We summarize the relevant factors in Figure 1.

We explore a wide range of popular mobile VOD services, including Amazon Video, DIRECTV, FOX NOW, Hulu, HBO GO, HBO NOW, MAX GO, Netflix, NBC Sports, Showtime Anytime and XFINITY TV. In this paper, we focus on 12 of these[2] covering a wide diversity of points in the design space, and study them in depth. These services individually have millions of app store downloads, and collectively span a wide range of content types including movies, TV shows and sports videos.

2.2 Methodology overview

Understanding the design choices and characterizing the QoE of these proprietary video streaming services are challenging, as they do not readily expose such information. To address the challenge, we develop a general methodology to extract information from the traffic and app UI events. To capture important properties of the adaptation logic designs, we further enhance our methodology with carefully crafted black-box testing to stress test the players.

Figure 2 shows an overview of the methodology. The proxy between the server and the user device emulates various network conditions (§ 2.6) and extracts video segment information from the traffic flow (§ 2.3). The on-device UI monitor monitors critical UI components (§ 2.4), such as the seekbar in the VOD apps that advances with the playback to inform users the playback progress and allow users to move to a new position in the video.

We combine information from the traffic analyzer and UI monitor to characterize QoE. While developing objective measures of overall user QoE for video streaming is still an active research area, it is commonly acknowledged that QoE is highly correlated to a few metrics listed below.

[2]One of the services adopts both DASH and SmoothStreaming. As they have very different design on both server and client side, we treat them as two different services.

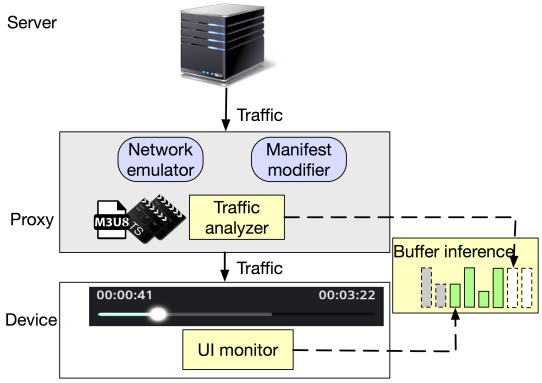

Figure 2: Methodology overview.

- **Video quality.** One commonly used metric to characterize video quality is average video bitrate, i.e. the average declared bitrate of segments shown on the screen. A low video bitrate indicates poor video quality, leading to poor user experience. However, the average bitrate by itself is not sufficient to accurately reflect user experience. As we discuss in more detail in § 4.1.3, user experience is more impacted by the playback of low quality, low bitrate tracks. It is therefore important to reduce the duration of streaming such tracks. To account for this, another metric is the percentage of playtime when low quality tracks are streamed.
- **Video track switches.** Frequent track switches impair user experience. One metric to characterize this is the frequency of switches. In addition, users are more sensitive to switches between non-consecutive tracks.
- **Stall duration.** This is the total duration of stall events during a session. A longer stall duration means higher interruptions for users and leads to poorer user experience.
- **Startup delay.** The startup delay measures the duration from the time when the users click the "*play*" button to the time when the first frame of video is rendered on the screen and the video starts to play. A low startup delay is preferred.

Each metric by itself provides only a limited viewpoint and all of them need to be considered together to characterize overall QoE. In our methodology, the *Traffic Analyzer* obtains detailed segment information, such as bitrate and duration etc, and therefore can be used to characterize video quality and track switches. The *UI Monitor* on the device tracks the playback progress from the player's UI, and is able to characterize the stall duration and initial delay. Furthermore, combining the information from both the traffic analyzer and the UI monitor, we can infer the player buffer occupancy across time (§ 2.5), which critically allows us to reason about, identify and unveil underlying causes of many QoE issues.

To understand complex designs such as the adaptation logic, the proxy uses the *Network Emulator* and *Manifest Modifier* to conduct black-box testing. The network emulator performs traffic shaping to emulate various network conditions. By carefully designing the bandwidth profile, we are able to force players to react and understand their design. In some cases, we use the manifest modifier to modify the manifest from the server and observe players' behavior to understand how client side players utilize information from servers. For example, in § 4.2 with the manifest modification,

we are able to explore whether players take actual track bitrate information into consideration when performing track selection.

In the following, we provide details of components used in the measurement methodology.

2.3 Traffic analyzer

We develop the network traffic analyzer to perform man-in-the-middle analysis on the proxy and extract manifest and segment information from flows between the server and client.

We observe that all the studied apps adopted one or more among the three popular HAS techniques, i.e. HLS, DASH, and SmoothStreaming. We denote the four services that use DASH as *D1* to *D4*, another six that use HLS as *H1* to *H6*, the two services that use SmoothStreaming as *S1* and *S2*.

We specifically developed the traffic analyzer to be generally applied for all VOD services that adopt the three popular standard HAS techniques. The traffic analyzer parses the manifest based on the specification of the HAS protocols, and builds the mapping between HTTP requests and segments. Since the three streaming protocol implementations have some different properties, the traffic analyzer extracts QoE information with different methodologies based on the protocol each service adopts. We shall mainly describe how the traffic analyzer works with the two most popular protocol implementations HLS and DASH.

HLS vs. DASH HTTP Live Streaming (HLS) [45] is a media streaming protocol proposed by Apple Inc. In HLS, a media presentation is described by a *Master Playlist*, which specifies the resolution, bitrate and the URL of corresponding *Media Playlist* of each track. The URL and duration of media segments are specified in the *Media Playlist*. Each media segment in HLS is a separate media file[3]. At the beginning of playback, the client downloads the *Master Playlist* to obtain information about each track. After it decides to download segments from a certain track, it downloads the corresponding *Media Playlist* and gets the URI of each segment.

Compared with HLS, the Dynamic Adaptive Streaming over HTTP (DASH) [1] is an international standard specifying formats to deliver media content using HTTP. Media content in DASH is described by the *Media Presentation Description (MPD)*, which specifies each track's declared bitrate, segment duration and URI etc. Each media segment can be a separate media file or a sub-range of a larger file. The byte-range and duration of segments may be directly described in the MPD. The MPD can also put such information in the *Segment Index Box (sidx)* of each track and specify the URI of sidx. The sidx contains meta information about the track and is usually placed at the beginning of the media file.

To accommodate the differences across the HAS protocol and service variations, the traffic analyzer works as follows. It gets the bitrate of each track from the Master Playlist for HLS, and then extracts the URI and duration of each segment from it. For DASH, it gets the bitrate of each track from the MPD, and generates the mapping of byte ranges to segment information using different data sources for different apps. *D2*, *D3* and *D4* put such information into the sidx of each track, while *D1* directly encodes it in the MPD. *D3* encrypts the MPD file in application layer before sending it through the network. However, the sidx is not encrypted and we can still get segment durations and sizes.

[3]From version 4, HLS also supports using a sub-range of a resource as a media segment. But none of our studied services use this feature.

2.4 UI monitor

The UI monitor aims at exposing QoE metrics that can be obtained from the app UI on the client. Based on our exploration of all the VOD apps in our study, we identify the seekbar to be a commonly used UI element that indicates the playing progress, i.e. the position of displayed frames in the video in time.

We investigate how to robustly capture the seekbar information. As the UI appearance of the seekbar has a significant difference across different apps, we do not resort to image process techniques. Instead, we use the Xposed framework [13], an Android framework which enables hooking Android system calls without modifying apps, to log system calls from the apps to update the seekbar.

We find that despite the significant difference in visual appearance, the usage of the seekbar is similar across the services. During playback, the players update the status of the seekbar periodically using the Android API *ProgressBar.setProgress*. Thus, we obtain information about playback progress and stall events from the API calls. The update may occur even when the seekbar is hidden on the screen. This methodology can be generally applied to apps that use the Android seekbar component regardless of the UI layout and visual appearance.

For the all apps we studied, the progress bar was updated at least every 1s and we can therefore get the current playing progress at at least 1s granularity.

2.5 Buffer inference

The client playback buffer status, including the occupancy and the information regarding segments in the buffer, is crucial for characterizing the player's behavior. We infer the buffer occupancy by combining information from the downloading process and the playback process, collected by the traffic analyzer and UI monitor respectively: at any time, the difference between the downloading progress and playing progress should be the buffer occupancy, and the details, such as the bitrate, and duration of the segments remaining in the buffer, can be extracted from the network traffic.

2.6 Network emulator

We use the Linux tool *tc* to control the available network bandwidth to the device across time to emulate various network conditions.

To understand designs such as the adaptation logic, we apply carefully designed network bandwidth profiles. For instance, to understand how players adapt to network bandwidth degradation, we design a bandwidth profile where the bandwidth stays high for a while and then suddenly drops to a low value. In addition, to identify QoE issues and develop best practices for cellular scenarios, it is important to compare the QoE of the different services in the context of real cellular networks. To enable repeatable experimentations and provide apples-to-apples comparisons between different services, we also replay multiple bandwidth traces from real cellular networks over WiFi in the lab for evaluating the services.

To collect real world bandwidth traces, we download a large file over the cellular network and record the throughput every second. We collect 14 bandwidth traces from real cellular network in various scenarios covering different movement patterns, signal strength

and locations. We sort them based on their average bandwidth and denote them from Profile 1 to Profile 14 (see Figure 3).

We run each of the services with the 14 collected cellular bandwidth traces. Each experiment lasts for 10min and is repeated for several runs to eliminate temporary QoE issues caused by the external environment, e.g. transient server load.

3 SERVICE CHARACTERIZATION

The interactions between different components of each VOD service across multiple protocol layers on both the client and server side together ultimately determine the QoE. Using our methodology from §2, for each service, we identify critical design choices around three key components: the server, the transport layer protocols, and the client, and investigate their QoE implications. We summarize the various designs in Table 1.

Our measurements reveal a number of interesting QoE-impacting issues caused by the various design choices (Table 2). We shall present the design factors related to these issues in this section and dive deeper into 3 most interesting problems in §4.

3.1 Server design

At the server-side, the media is encoded into multiple tracks with different bitrates, with each track broken down into multiple segments, each corresponding to a few seconds worth of video. Understanding these server-side settings is important as they have critical impact on the adaptation process and therefore the QoE.

For each service, we analyze the first 9 videos on the landing page which span different categories. We find that for all studied services, for the 9 videos in the same service, the settings are either identical or very similar. We select one of these videos as a representative sample to further illustrate the design for each service.

Separate audio track. The server can either encode separate audio tracks or multiplex video and audio content in the same track. Using separate audio tracks decouples video and audio content, and gives a service more flexibility to accommodate different audio variants for the same video content, e.g. to use a different language or a different audio sample rate. We analyze a service's manifest to understand whether the service encodes separate audio tracks. We find that all the studied services that use HLS do not have separate audio tracks, while all services that use DASH or SmoothStreaming encode separate audio tracks.

Track bitrate setting. Track settings such as track count (number of tracks), the properties of the highest and lowest tracks, and the spacing (bitrate difference) between consecutive tracks all impact HAS adaptation and therefore the QoE. We obtain the track declared bitrate from the manifest of each service[4].

The highest track represents the highest quality that a service provides. We find across the services the highest track has diverse bitrates from 2 Mbps to 5.5 Mbps. Note that the declared bitrate is not the only factor that determines video quality, as it also depends on other factors such as encoding efficiency.

The bitrate of the lowest track impacts the players' ability to sustain seamless playback under poor network conditions. Apple recommends that the lowest track should be below 192 kbps for cellular network [16]. However, the lowest track of 3 services is higher than 500 kbps and significantly increases the possibility of having stalls with slow network connection. For example, our evaluations show with the two lowest bandwidth profiles, *H5* always stalls for more than 10 s, while apps with lower bit-rate bottom tracks such as *D2* and *D3* do not have stalls under the same network conditions. *Because stalls severely impact QoE, we suggest setting the bitrate of the bottom track to be reasonably low for mobile networks.*

Tracks inbetween the highest and lowest track need to be selected with proper inter-track spacing. If adjacent tracks are set too far apart, the client may often fall into situations where the available bandwidth can support streaming a higher quality track, but the player is constrained to fetch a much lower quality, due to the lack of choices. If adjacent tracks are set too close to each other, the video quality improves very little by switching to the next higher track and the higher track count unnecessarily increases server-encoding and storage overheads. Apple recommends adjacent bitrate to a factor of 1.5 to 2 apart [16]. All services we study are consistent with this guideline.

CBR/VBR Encoding. Services can use two types of video encoding scheme, i.e. Constant Bitrate (CBR) encoding which encodes all segments into similar bitrates, and Variable Bitrate (VBR) encoding which can encode segments with different bitrates based on scene complexity [9].

We examine the distribution of bitrates across segments from the same track to determine the encoding. We get segment duration information from the manifest. To get segment sizes, for services using DASH, we directly get segment sizes from the byte range information provided by the manifest and sidx. For services using HLS and SmoothStreaming, we get the media URLs from the manifest file and use *curl* [11] to send HTTP HEAD requests to get the media size. We find that 3 services use CBR, while the others use VBR with significant different actual segment bitrates in a single track. For example, the peak actual bitrate of *D1* is twice the average actual bitrate.

With VBR encoding, using a single declared bitrate to represent the required bandwidth is challenging. We look into how services set the declared bitrate. For the highest track of each service, we examine the distribution of actual segment bitrates normalized by the declared bitrate. As shown in Figure 5, *S1* and *S2* set the declared bitrate around the average actual bitrate, while other services set the declared bitrate around the peak actual bitrate. We shall explore further in § 4.2 the associated QoE implications.

Segment duration. The setting of segment duration involves complex tradeoffs [3]. A short segment duration enables the client to make track selection decision in finer time granularity and adapt better to network bandwidth fluctuations, as segments are the smallest unit to switch during bitrate adaptation. On the other side, a long segment duration can help improve encoding efficiency and reduce the server load, as the number of requests required to download the same duration of video content reduces. We find significant differences in the segment duration across the different services, ranging from 2s to as long as 10s (see Table 1). We leave a deeper analysis on characterizing the tradeoffs to future work. In addition, as we find later in § 4.3, other factors such as startup buffer duration need to be set based on the segment duration to ensure good QoE.

[4] This approach did not work for *D3* as the manifest is encrypted at the application layer and cannot be decrypted. Instead, we use the peak value of the actual segment bitrates (which can be obtained by parsing the sidx) as the declared bitrate since other DASH services such *D1* and *D2* follow such practice

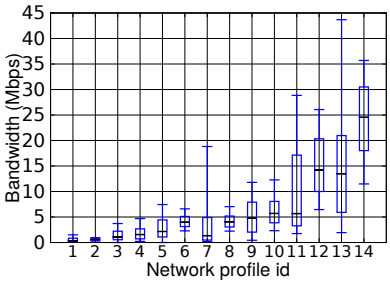

Figure 3: Collected cellular network bandwidth profiles.

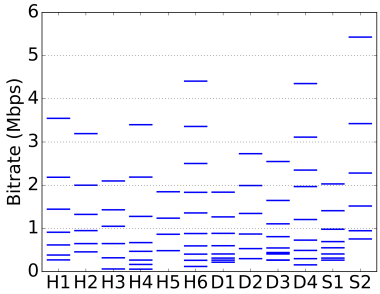

Figure 4: Declared bitrates of tracks for different services

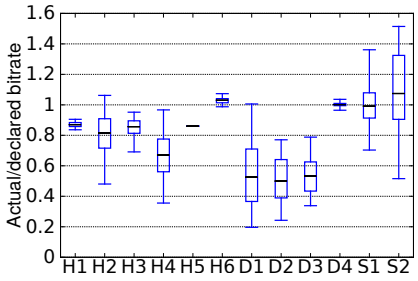

Figure 5: The distribution of actual bitrate normalized by declared bitrate

Designs		H1	H2	H3	H4	H5	H6	D1	D2	D3	D4	S1	S2
Server	Segment duration (s)	4	2	9	9	6	10	5*	5	2	6	2	3*
	Separate audio track	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y
Transport layer	Max #TCP	1	1	1	1	1	1	6	2	3	3	2	2
	Persistent TCP	Y	N	N	Y	N	Y	Y	Y	Y	Y	Y	Y
Startup	Startup buffer (s)	8	8	9	9	12	10	15	5	8	6	16	6
	Startup bitrate (Mbps)	0.63	1.33	1.05	0.47	1.85	0.88	0.41	0.30	0.40	0.67	1.35	0.76
Download control	Pausing threshold (s)	95	90	40	155	30	80	182	30	120	34	180	30
	Resuming threshold (s)	85	84	30	135	20	70	178	25	90	15	175	4
With constant bw	Stability	Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
	Aggressiveness	N	N	N	N	N	N	Y	N	Y	N	Y	N
With varying bw	Decrease buffer (s)	-	40	-	-	-	-	-	-	30	-	50	-

* The audio segment duration of *D1* and *S2* is 2s.

Table 1: Design choices

Design factors	Problem	QoE impact	Affected service
Track setting	The bitrate of lowest track is set high.	Frequent stalls	*H2, H5, S1*
Encoding scheme	Adaptation algorithms do not consider actual segment bitrate.	Low video quality	*D2*
TCP utilization	Audio and video content downloading progress is out of sync when using multiple TCP connections.	Unexpected stalls	*D1*
TCP persistence	Players use non-persistent TCP connections.	Low video quality	*H2, H3, H5*
Download control	Players do not resume downloading segments until the buffer is almost empty.	Frequent stalls	*S2*
Startup logic	Players start playback when only one segment is downloaded.	Stall at the beginning	*H3, H4, H6, D2, D4*
Adaptation logic	The bitrate selection does not stabilize with constant bandwidth.	Extensive track switches	*D1*
	Players ramp down selected track with high buffer occupancy.	Low video quality	*H1, H4, H6, D1*
	Players can replace segments in the buffer with ones of worse quality.	Waste data and low video quality	*H1, H4*

Table 2: Identified QoE-impacting issues

3.2 Transport layer design

In HAS, players use the HTTP/HTTPS protocol to retrieve segments from the server. However, how the underlying transport layer protocols are utilized to deliver the media content depends on the service implementation. All the VOD services in this study use TCP as the transport layer protocol.

TCP connection count and persistence. As illustrated in Table 1, all studied apps that adopt HLS use a single TCP connection to download segments. 3 of these apps use non-persistent TCP connections and establish a new TCP connection for each download. This requires TCP handshakes between the client and server for each segment and TCP needs to go through the slow start phase for each connection, degrading achievable throughput and increasing the potential of suboptimal QoE. *We suggest apps use persistent TCP connections to download segments.* All apps that adopt DASH and SmoothStreaming use multiple TCP connections due to separated audio and video tracks. All these connections are persistent.

TCP connection utilization. Utilizing multiple TCP connections to download segments in parallel brings new challenges. Some apps such as *D1* use each connection to fetch a different segment. Since concurrent downloads share network resources, increasing the concurrency can slow down the download of individual segments. This can be problematic in some situations (especially when either the buffer or bandwidth is low) by delaying the arrival of a segment with a very close playback time, increasing the potential for stalls. Different from these apps, *D3* only downloads one segment at a time. It splits each video segment into multiple subsegment and schedules them on different connections. To achieve good QoE, the splitting point shall be carefully selected based on per connection bandwidth to ensure all sub-segments arrive in similar time, as the whole segment needs to be downloaded before it can be played. The above highlights that developing a good strategy to make efficient utilization of multiple TCP connections requires considerations of complex interactions between the transport layer and application layer behavior. We leave further exploration to future work.

When audio and video tracks are separate, the streaming of audio and video segments are done separately. Since both are required to

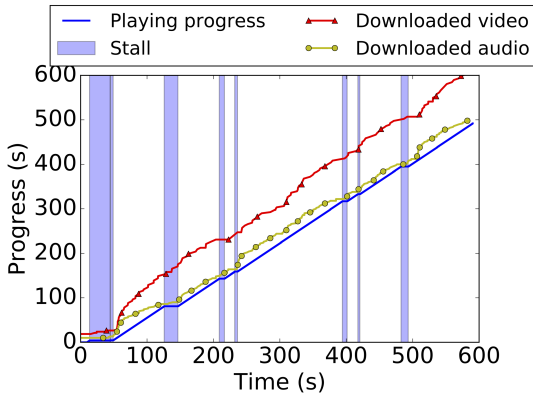

Figure 6: The downloading progress of video and audio content of *D1* is out of sync, causing unexpected stalls.

play any portion of the video, there should be adequate synchronization across the two download processes to ensure that both contents are available by the designated playback time of the segment. Our evaluations reveal that uneven downloads for audio and video lead to clear QoE impairments for some apps. For example, we find *D1* uses multiple TCP connections to download audio and video content in parallel, but its download progresses for audio and video content can have significant differences, especially when the network bandwidth is low. For the two network profiles with the lowest average bandwidth, the average difference between video and audio downloading progress is 69.9 s and 52.5 s respectively. In the example shown in Figure 6, buffered video content is always more than audio content. When stalls occur, the buffer still contains around 100 s of video content. In this case, the stalls could have been avoided, without using any additional network resources, by just reusing some of the bandwidth for fetching more audio and a little bit less video. *We suggest ensuring better and tighter synchronization between audio and video downloads.*

3.3 Client-side design

The client player is a core component that impacts QoE by performing intelligent adaptation to varying network conditions. In this subsection we stress test the different players using the 14 bandwidth profiles collected from various scenarios. By comparing the behavior across different services under identical network conditions, we are able to identify interesting client behaviors and pinpoint potential QoE problems. More specifically, we use black-box testing to study how players behave at startup, i.e. the startup logic, when they load the next segment, i.e. the download control policy and what segment they load, i.e. the adaptation logic.

3.3.1 Startup logic. We characterize two properties in the startup phase, startup buffer duration and startup track.

Startup buffer duration. At the beginning of a session, clients need to download a few segments before starting playback. We denote the minimal buffer occupancy (in terms of number of seconds' worth of content) required before playback is initiated as the startup buffer duration.

Setting the startup buffer duration involves tradeoffs as a larger value can increase the initial delay experienced by the user (as it takes a longer time to download more of the video), but too small a value may lead to stalls soon after the playback. To understand how popular services configure the startup buffer, we run a series of experiments for each service. In each experiment we instrument the proxy to reject all segment requests after the first n segments. We gradually increase n and find the minimal n required for the player to start playback. The duration of these segments is the startup buffer duration. As shown in Table 1, most apps set similar startup duration around 10s.

Startup track. The selection of the first segment impacts users' first impression of the video quality. However, at the beginning the player does not have information about network conditions (eg., historical download bandwidths), making it challenging to determine the appropriate first segment.

We examine the startup track of different players in practice. We find each app consistently selects the same track level across different runs. The startup bitrates across apps have high diversity. 4 apps start with a bitrate lower than 500 kbps, while another 4 apps set the startup bitrate higher than 1 Mbps. We shall further explore the QoE impact of startup buffer duration and startup track in sec 4.3.

3.3.2 Download control. One important decision the client makes is determining when to download the next segment. A naive strategy is to keep fetching segments continuously, greedily building up the buffer to avoid stall events. However, this can be suboptimal as (1) it increases wasted data when users abort the session and (2) it may miss the opportunity to get a higher quality segment if network condition improves in the future. We observe that, even under stable network conditions, all the apps exhibit periodic on-off download patterns. Combining with our buffer emulation, we find an app always pauses downloading when the buffer occupancy increases to a *pausing threshold*, and resumes downloading when the occupancy drops below another lower *resuming threshold*.

We set the network bandwidth to 10 Mbps, which is sufficient for the services to their respective highest tracks. We find 5 apps set the pausing threshold to be around 30 s, while other apps set it to be several minutes (Table 1). With a high pausing threshold, the player can maintain a high buffer occupancy to avoid future stall events. However, it may lead to more data wastage when users abort the playback. The different settings among services reflect different points in the decision space around this tradeoff.

The difference between the pausing and resuming threshold determines the network interface idle duration, and therefore affects network energy consumption. 8 apps set the two thresholds to be within 10 s of each other. As this is shorter than LTE RRC demotion timer [41], the cellular radio interface will stay in high energy mode during this entire pause in the download, leading to high energy consumption. *We suggest setting the difference of the two thresholds larger than LTE RRC demotion timer in order to save device energy.*

If either the pausing threshold or the resuming threshold is set too low, the player's ability to accommodate network variability will be greatly limited, leading to frequent stalls. We find that *S2* sets the pausing threshold to be only 4s and has a higher probability of incurring stalls than other services under similar network conditions. As the example in Figure 7, at 25 s, the buffer occupancy of *S2* reaches to the pausing threshold and the player pauses downloading for around 30 s. When the player resumes downloading

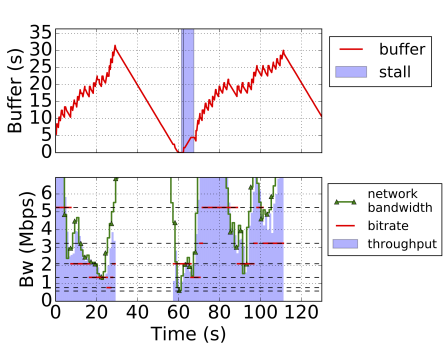

Figure 7: *S2* sets the resuming buffer to only 4s, leading to stalls.

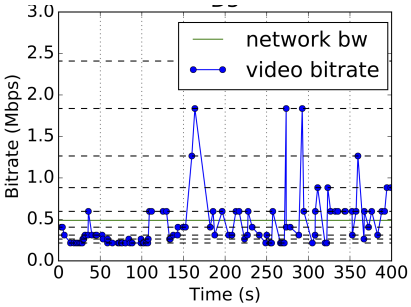

Figure 8: *D1* selected track is not stable even with constant bandwidth

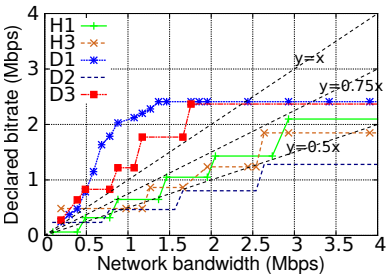

Figure 9: Selected declared bitrate given a constant bandwidth

segments, the buffer occupancy is only 4s and drains quickly due to temporary poor network condition. As stalls significantly degrade user experience, *we suggest setting both thresholds reasonably high to avoid stalls.* The exact value will depend on factors like the specific adaptation algorithm and is beyond the scope of this paper.

Next, we study the client adaptation logic. A good adaption logic should provide high average bitrate and reduce stall events and unnecessary track switches.

3.3.3 Track selection under stable network bandwidth. For each app, we run a series of experiments within each of which we emulate a specific stable network bandwidth for 10 min and examine the resulting track selection in the steady state. A good adaption logic should achieve an average bitrate similar to the network bandwidth without stalls and frequent track switches.

Stability. We find that the selected track of *D1* does not stabilize even with constant network bandwidth. As shown in Figure 8, the network bandwidth is constantly 500 kbps. However, *D1* frequently switches between different tracks and tries to improve the average actual bitrate to be close to network bandwidth. However, frequent switches, especially switches between non-consecutive tracks, can impair user experience. In contrast, the other apps all converge to a single track (different for each app) after the initial startup phase. *We suggest the adaptation logic avoid unnecessary track switches.*

Aggressiveness. We find that the track that different apps converge to under the same stable bandwidth condition has significant difference across different services. We term services that converge to a track with declared bandwidth closer to available bandwidth as more aggressive. We show a few examples in Figure 9. We find 3 apps are more aggressive and select tracks with bitrate no less than the available network bandwidth. The reason why they are able to stream tracks with a bitrate higher than available network bandwidth without stalls is that they use VBR encoding and the actual segment bitrate is much lower than the declared bitrate. The other apps are relatively conservative and select tracks with declared bitrates no more than 75% of the available bandwidth. In particular, *D2* even select tracks with declared bitrates no more than 50% of available bandwidth.

3.3.4 Track adaptation with varying network bandwidths. To understand the adaptation to varying network condition, we run each app with a simple "step function" bandwidth profile, i.e. the network bandwidth first stays stable at one value and suddenly changes to another value. We test different combinations of the initial and final bandwidth steps, and when the step occurs. The behavior across the different apps is summarized in Table 1.

Reaction to bandwidth increase. When bandwidth increases, all apps start to switch to a track with higher bitrate after a few segments. In addition, we find some apps revisit earlier track switching decisions and redownload existing segments in the buffer in an attempt to improve video quality. We further analyze this in § 4.1.

Reaction to bandwidth decrease. When bandwidth decreases, apps eventually switch to a track with a lower bitrate.

A higher buffer pausing threshold enables more buffer buildup, which can help apps better absorb bandwidth changing events and defer the decision to select a lower track without the danger of stalls. However, among the 7 apps that have a large buffer pausing threshold (larger than 60 s), 4 apps always immediately switch to a low track when a bandwidth degradation is detected, even when the buffer occupancy is high, leading to suboptimal QoE. In contrast, the other 3 apps set thresholds on buffer occupancy above which they do not switch to a lower track even if the available bandwidth reduces. *We suggest the adaptation logic takes buffer occupancy into consideration and utilizes the buffer to absorb network fluctuations.*

In summary, our measurements show popular VOD services make a number of different design choices and it is important to perform such cross-section study to better understand the current practices and their QoE implications.

4 QOE ISSUES: DEEP DIVE

Some QoE impacting issues involve complex interactions between different factors. In this section, we explore in depth some key issues impacting the services we study, and use targeted black-box experiments to deduce their root causes. In addition, we further examine whether similar problems exist for ExoPlayer, an open source media player used by more than 10,000 apps [4] including YouTube [2], BBC [5], WhatsApp [6] and Periscope [8] etc. Exoplayer therefore provides us a unique view of the underlying design decisions in a state-of-the-art HAS player being increasingly used as the base for many commercial systems. The insights and mitigation strategies we develop from this exploration can be broadly beneficial to the community for improving the QoE of VOD services.

4.1 Segment Replacement (SR)

Existing adaptation algorithms [27, 31, 33, 52] try to make intelligent decisions about track selection to achieve the best video quality while avoiding stall events. However, due to the fluctuation of network bandwidth in the mobile network, it is nearly impossible for the adaption logic to always make the perfect decision on selecting the most suitable bitrate in terms of the tradeoff between quality and smoothness. We observe that to mitigate the problem, when the network condition turns out to be better than predicted, some players will discard low quality segments that are in the buffer but have not yet been played, and redownload these segments using a higher quality track to improve user perceived video quality. We denote this behavior of discarding video segments in the buffer and redownloading them with potentially different quality as *Segment replacement* (SR).

While SR could potentially improve video quality, it does involve some complex tradeoffs. As segments in the buffer are discarded and redownloaded, the additional downloads increase network data usage. In addition, SR uses up network bandwidth which could potentially have been used instead to download future segments and may lead to quality degradation in the future. Existing works [36, 40, 48] find Youtube can perform extensive SR in a non-cellular setting. However, how common SR is used across popular services and the associated cost-benefit tradeoff for cellular networks is not well understood. We characterize this tradeoff for popular services, identify underlying causes of inefficiencies, and propose improvements.

4.1.1 Usage and QoE impact of SR for popular VOD apps. To understand the usage of SR by popular VOD apps, we run them with the 14 collected network bandwidth profiles. We analyze the track and index (the position of the segment within the video track) of downloaded segments. As segments with the same index represent the same content, when multiple segments with the same index are observed in the traffic, we confirm that the player performs SR. Among the players we study, we find *H1* and *H4* perform SR.

We conduct what-if analysis to characterize the extent of video quality improvement and additional data usage caused by SR. When SR occurs, among the segments with the same index, only the last downloaded segment is preserved in the buffer and all previous downloads are discarded. We confirm this using the buffer information in the logcat of *H1*. We emulate the case with no SR by keeping only the first downloaded segment for each index in the trace and use it as a baseline comparison. Our analysis shows that SR as currently implemented by *H4*, does not work well. The findings are summarized as follows. *H1* shows similar trends.

- SR as currently implemented can significantly increase data usage. With 5 of the bandwidth profiles, the data consumption increases by more than 75%. The median data usage increase is 25.66%.
- For most bandwidth profiles, the video quality improves marginally. The median improvement in average bitrate across the 14 profiles is 3.66%.
- Interestingly, we find SR can even degrade video quality. For one profile, SR decreases the average bitrate by 4.09% and the duration for which tracks higher than 1 Mbps are streamed reduces by 3.08%.

The video quality degradation we observed with SR is surprising, as one would expect SR to only replace lower-bitrate segments with higher-bitrate ones and therefore improve the average bitrate. Diving deeper, for each experimental run, we emulate the client buffer over time. When a new segment is downloaded, if the buffer already contains a segment with the same index, we replace the previously buffered segment with the newly downloaded one and compare their quality. A somewhat counter-intuitive finding is that the redownloaded segments are not always of higher quality. Across the 14 bandwidth profiles, for all SR occurrences, on average respectively 21.31% and 6.50% of redownloaded segments were of lower quality or same quality as the replaced segment. These types of replacements are intuitively undesirable, as they use up network resources, but do not improve quality.

To understand why *H4* redownloads segments with lower or equal quality, we analyze when and how *H4* performs SR. We make the following observations.

- **How SR is performed.** We find that after *H4* redownloads a segment *seg*, it always redownloads all segments that are in the buffer with indexes higher than *seg*. In other words, it performs SR for multiple segments proactively and does not just replace a segment in the middle of the buffer. In all SR occurrences across the 14 profiles, the 90^{th} percentile of the number of contiguously replaced segments was 6 segments.
- **When SR is triggered.** Whenever *H4* switches to a higher track, it always starts replacing some segments in the buffer. For all runs with the 14 bandwidth profiles, each time SR occurs, we examine the quality of the first replaced segment among the contiguous replaced ones. We find in 22.5% of SR cases, even the first redownloaded segment had lower or equal quality compared with the one already in the buffer. This implies that *H4* may not properly consider the video quality of buffered segments when performing SR.

We show an example of *H4* performing SR in Figure 10. At 150 s, *H4* switches from Track 3 to Track 4, which triggers SR. Instead of downloading the segment corresponding to 580 s' of content, it goes back to redownload the segment corresponding to 500 s' of content. In fact, that segment was already downloaded at 85 s with a higher quality from Track 8. As the new downloaded segment is from Track 4, this indicates SR with lower quality. Even worse, *H4* keeps redownloading all buffered segments after that. This even causes a stall at 165 s, which otherwise could have been avoided.

Deducing the root causes of such suboptimal SR design from commercial players such as *H4* is challenging due to their proprietary nature. To gain a deeper understanding into the underlying considerations behind SR policies, we next examine the SR design of the popular open-source ExoPlayer and its QoE implications.

4.1.2 SR analysis with ExoPlayer. We find that ExoPlayer version 1 uses SR and suffers from some similar issues as *H4*, i.e. it can also redownload segments with lower or equal quality. To understand this, we first need to understand Exoplayer's adaptation logic. Before loading each segment the track selection algorithm selects the track based on available network bandwidth and buffer occupancy. When it decides to select a higher track X than the last selected one Y, it initiates SR if the buffer occupancy is above a threshold value. It identifies the segment with the smallest playback

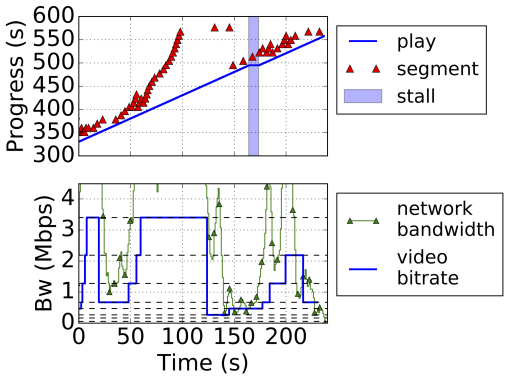

Figure 10: *H4* starts SR as long as it switches to a higher track and does not consider the track of segments in the buffer.

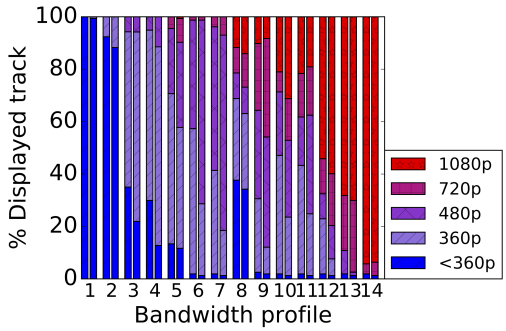

Figure 11: The displayed track percentage with/without SR. Each pair of bars are with the same network condition: left is without SR; right is with SR.

index in the buffer that is from a track lower than the track Y that ExoPlayer is about to select for the upcoming download. Beginning with that segment, it discards all segments with a higher index from the buffer. While this strategy guarantees that the first discarded segment is replaced with higher quality one, the same does not hold for the following segments being replaced.

The root cause of these SR-related issues is that the player does not (i) make replacement decision for each segment individually and (ii) limit SR to only replace segments with higher quality. To answer the question *why players including H4 and ExoPlayer do not do this*, we study the ExoPlayer code and discover that it does not provide APIs to discard a single segment in the middle of the buffer. Further investigation shows that this is caused by the underlying data structure design. For efficient memory management, ExoPlayer uses a double-ended queue to store segments ordered by the playback index. Network activities put new segments on one end, while the video renderer consumes segments on the other end, which ensures that the memory can be efficiently recycled. Discarding a segment in the middle is not supported, and thus to perform SR, the player has to discard and redownload all segments with higher indexes than the first chosen one.

We find that the underlying data structure and SR logic remain the same in the latest Exoplayer version 2, but that SR is currently deactivated and marked for future activation. To understand the reasons behind ExoPlayer's approach to SR, we contacted its designers. They communicated that they were concerned about the additional complexity and less efficient memory allocation associated with allowing a single segment in the middle to be discarded, and uncertainty about the benefits of SR. They were also concerned that allowing discard for a single segment introduces some dependency between the track selection algorithm and other modules such as buffering policy.

4.1.3 SR Best practices and improvement evaluation. The response from ExoPlayer developers motivates us to look into how useful SR is when designed properly and whether it is worthwhile to implement it. Intuitively a proper SR logic should have the following properties.

- The logic considers replacing a segment a time. Each segment is replaced individually.
- Segments can only be replaced by higher quality segments.
- When buffer occupancy drops below a threshold, the player should stop performing more replacements and resume fetching future segments to avoid the danger of stalls.

Changing the Exoplayer memory management implementation to enable discarding individual segments is a non-trivial endeavor. Instead, for our evaluations, we modify the Exoplayer track selection logic to work with HTTP caching to achieve the same end-results. As an example, when segments are discarded from the buffer, their track information is recorded. Later if the track selection logic determines to redownloaded them with quality no higher than the discarded ones, we change the track selection to select the track of the discarded segment so that they can be recovered directly from the local cache on the device without sending traffic to the network. From the network perspective, this would have the same effect as not discarding the segment.

To evaluate the QoE impact of the improved SR algorithm, we play a publicly available DASH stream [7] using the 14 collected real world bandwidth profiles. We find that, across the profiles, the median and 90^{th} percentile improvements in average bitrate are 11.6% and 20.9% respectively.

Subjective QoE studies(eg., [35]) show that the video bitrate is not linearly proportional to user QoE. Rather, increasing the bitrate when bitrate is low will cause a much sharper increase in user experience. But when bitrate is already high, further increasing the bitrate does not lead to significant additional QoE improvements. In other words, it is more important to reduce the duration of time that really low quality tracks are streamed. Thus we further break down the track distribution of displayed segments without and with SR. As shown in Figure 11, when network bandwidth shows significant fluctuation and players have chances to switch between tracks, a properly designed SR strategy can greatly reduce the duration of streaming low tracks. For bandwidth profiles 3 and profile 4, the duration of streaming tracks lower than 360p reduces by 32.0% and 54.1% respectively. For profile 7 to profile 12, the duration of streaming tracks worse than 480p reduces significantly, reduction ranging from 30.6% to 64.0%.

SR increases video bitrate at the cost of increasing network data usage. For ExoPlayer with our improved SR algorithm, the median data usage increase across 14 profiles is 19.9%. For 5 profiles, the

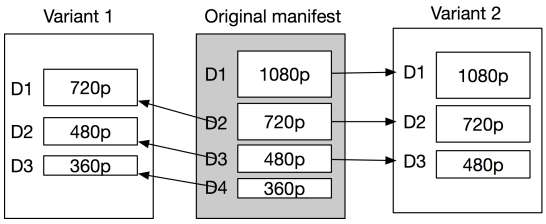

Figure 12: We modify the manifest and shift the mapping between declared bitrate and corresponding media files to generate two streams with the same declared bitrate but different actual bitrate (D in the figure stands for declared bitrate).

usage increases by more than 40%. Across the 14 profiles, the median amount of wasted data, i.e. data associated with downloading segments that were later discarded, as a proportion of the total data usage was 10.8%. This implies that SR should be performed carefully for users with limited data plans.

To better make tradeoff between data usage and video quality improvement, we suggest only discarding segments with low quality when data usage is a concern. As we shall see, discarding segments with lower bitrate has a bigger impact on improving QoE and causes less waste data. To evaluate the proposed concept, we change the SR algorithm to only replace segments no better than a threshold of 720p, and characterize the impact on data usage and video quality. We test with three profiles with the largest amount of waste data. Compared with the case of using no such threshold, for the 3 profiles, the wasted data reduced by 44% on average, while the proportion of time that streaming quality better than 720p was played stayed similar. The results therefore show that this is a promising direction for exploring practical SR schemes. Further work is needed in fine tuning the threshold selection.

In summary, we find proper usage of SR significantly reduces the duration of streaming tracks with poor quality and improves QoE. When making replacement decisions, players should consider each segment individually and only replace segment with higher quality. This requires underlying implementation to support discarding a segment in the middle of the buffer. Due to the implementation complexities, creating a library that supports such operations can greatly benefit the app developer community.

4.2 Using Declared vs. Actual Bitrate

Servers specify the declared bitrate for each track in the manifest as a proxy for its network resource needs, to help client players select proper tracks based on the network bandwidth. However, especially for VBR encoding which is increasingly popular, a single declared bitrate value cannot accurately reflect the actual bitrate across the video. For example, as shown in Figure 5, the declared bitrate of videos from *D2* can be twice of the average actual bitrate. Despite the potentially significant difference between the declared bitrate and actual bitrate, we find that the adaptation logic in some players such as *D2* relies purely on the declared bitrate to make track selection decisions, leading to suboptimal QoE.

Since *D2* uses DASH, it can in theory obtain actual segment bitrates from segment index boxes before playback. To verify whether *D2* takes the actual bitrate into consideration during track selection, we carefully design black-box testing experiments to reveal its internal logic. We modify the manifest to generate two variants with tracks of the same declared bitrate but different actual bitrates. As illustrated in Figure 12, in variant 1 we shift the mapping between the declared bitrate and corresponding media files. We replace the media of each track to the one with the next lower quality level, while keeping the declared bitrate the same. In variant 2, we simply remove the lowest track and keep other tracks unchanged to keep the same number of tracks as variant 1. Thus, comparing these two variants, each track in variant 1 has the same declared bitrate as the track of the same level in variant 2, but the actual bitrate is the same as that of the next lower track in variant 2. We use *D2* to play the two variants using a series of constant available bandwidth profile. We observe that with the same bandwidth profile, the selected tracks for the two variants are always of the same level with the same declared bitrate. This suggests that it only considers the declared bitrate in its decision on which track to select next, else the player would select tracks with different levels for the two variants but with the same actual bitrate.

As the average actual bitrate of videos from *D2* is only half of declared bitrate, failure to consider the actual bitrate can lead to low bandwidth utilization, and thus deliver suboptimal QoE. We use *D2* to play original videos from its server with a stable 2 Mbps available bandwidth network profile. The average achieved throughput is only 33.7% of the available bandwidth in the steady phase. Such low bandwidth utilization indicates that *D2* could potentially stream higher quality video without causing stalls.

There are historical factors underlying the above behavior. HLS was the first widely adopted HAS streaming protocol for mobile apps, and some elements of its design meshed well with the needs of the predominant encoding being used at the time, i.e. CBR. For example, the HLS manifest uses a single declared bitrate value to describe the bandwidth requirements for each track. This is the only information available to the player's track selection logic regarding the bandwidth needs for a segment in a track, before actually downloading the segment. HLS requires setting this value to the peak value for any segment in the track [45]. With CBR encoding, different segments in a track have similar actual bitrates, making the declared bitrate a reasonable proxy for the actual resource needs. Adaptation algorithms [31, 33, 34] therefore traditionally have depended on the declared bitrate to select tracks.

More recently, HAS services have been increasingly adopting VBR video encodings as shown in Figure 5, which offers a number of advantages over CBR in terms of improved video quality. However, different segments in a VBR encoded track can have very different sizes due to factors such as different types of scenes and motion.

As the actual bitrate of different segments in the same track can have significant variability, it becomes challenging to rely on a single declared bitrate value to represent all the segments in a track. With VBR encoding, setting the declared bitrate to average actual bitrate can lead to stall events [27, 54]. On the other hand, setting the declared bitrate to the peak rate and using that as an estimate for a track's bandwidth (as *D2* seems to do) can lead to low bandwidth utilization and suboptimal video quality. The solution to the above is that (i) more granular segment size information should be made available to the adaptation algorithm and (ii) the algorithm should utilize that information to make more informed decisions about track selection.

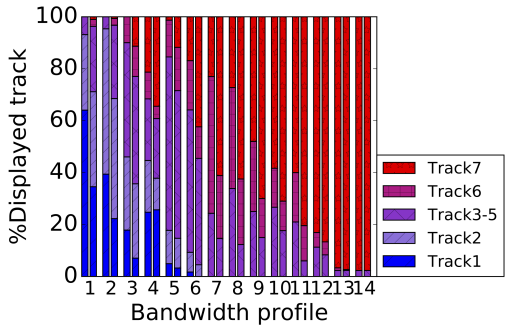

Figure 13: The displayed track percentage without/with considering actual segment bitrate. Each pair of bars are with the same network condition. The left one the is the distribution only considering declared bitrate. The right one is the distribution considering actual bitrate.

HAS protocols are moving towards making this granular information available, but challenges remain. DASH and newer versions of HLS support storing each segment as a sub-range of a media file and expose the segment byte ranges and durations in the manifest file which can be used to determine the actual bitrate for each segment. HLS also supports reporting the average bitrate in the manifest along with the peak bitrate. Thus, in theory, an adaptation logic should now be able to utilize this information. However, we find that the information may still not be exposed to the adaptation algorithm. We checked the implementation of ExoPlayer version 2, the latest version. It provides a unified interface to expose information based on which an adaptation algorithm selects tracks. However, the interface only exposes limited information including track format, declared bitrate, buffer occupancy and bandwidth estimation. It does not expose the actual segment-level bitrate information that is included in the manifest file. This implies that even though app developers can implement customized sophisticated adaptation algorithms, in Exoplayer, currently they still can not leverage actual bitrate information to select tracks.

We next demonstrate that even a simple adaptation algorithm that considers actual segment bitrates can improve QoE. We adjust ExoPlayer's default adaptation algorithm to select the track based on the actual segment bitrate instead of the declared bitrate. To evaluate the performance, we VBR-encode the Sintel test video [14] and create an HLS stream consisting of 7 tracks. For each track we set the peak bitrate (and therefore the declared bitrate) to be twice of the average bitrate. We play the video both with the default adaptation algorithm and the modified algorithm that considers actual bitrate using the 14 collected network profiles.

As shown in Figure 13, when actual bitrate is considered, the duration of playing content with low quality reduces significantly. Across the 14 network profiles the median of average bitrate improvements is 10.22%. For the 3 profiles with the lowest average bandwidth, the duration for which the lowest track is played reduces by more than 43.4% compared with the case of considering only the declared bitrate for track selection. Meanwhile, for all profiles we observe the stall duration stays the same, except for one profile, where it increases marginally from 10 s to 12 s. Note

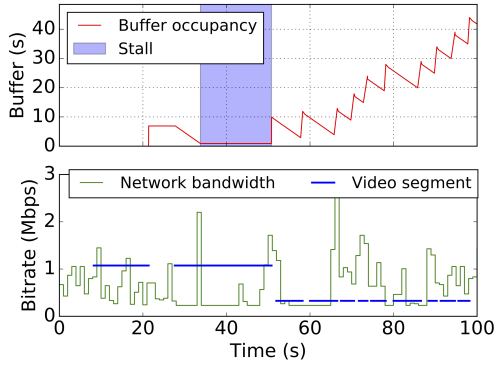

Figure 14: *H3* encounters a stall soon after starting to play.

that the above results just illustrates the potential of using fine-grained segment size information. The development of superior HAS adaptation schemes for VBR to make better tradeoff between video quality and stalls is a separate research topic in itself.

In summary, we suggest the services should expose actual segment bitrate information to the adaptation logic, and that the adaptation logic should utilize such information to improve track selection.

4.3 Improving Startup Logic

We find that some apps such as *H3* always have stalls at the beginning of playback with certain network bandwidth profiles, while other apps do not have stalls under the same network condition. This indicates potential problems with the startup logic. As shown in Figure 14, *H3* first selects the track with a bitrate around 1 Mbps, which is higher than the available network bandwidth . It starts playback after downloading the first segment. For the second segment it keeps selecting the same track as it may not yet have built up enough information about the actual network condition. As the network bandwidth is lower than selected bitrate, the buffer goes empty before the second segment is downloaded, leading to a stall.

The investigation into the design difference between apps with and without QoE problems can give us hints on potential causes and solutions. We find *H3* and *H2* set similar startup buffer durations. However, *H2* does not encounter stalls with the same network, while *H3* does. Further analysis shows that each segment of *H2* is only 2 s long and it downloads 4 segments before starting playback, while the segment duration for *H3* is 9 s and it starts playback once a single segment is downloaded. Based on this observation, we hypothesize that the likelihood of having stalls at the beginning of playback does not only depend on the startup buffer duration in seconds but also on the number of segments in the buffer. Using just 1 segment as startup buffer introduces a high possibility to have stalls at the beginning of playback.

To validate this hypothesis and identify improvements, we characterize the tradeoff brought by startup buffer duration setting between resulting startup delay and stall likelihood at the beginning of a video session, and propose suggestions for determining the setting empirically. We instrument ExoPlayer to set different startup buffer durations and play the Testcard stream, a publicly available DASH stream [7], with different segment durations. We also configure the player to use different startup track settings. We

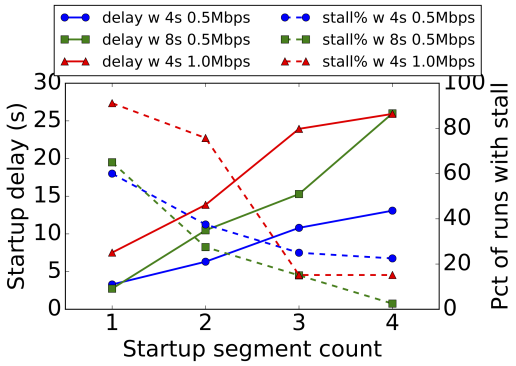

Figure 15: Startup delay and stall ratio with different segment durations, startup tracks and startup segment count. For example, "4s 0.5 Mbps" in the legend means the segment duration is 4 s and the startup track bitrate is 0.5 Mbps.

calculate the average startup delay and the stall ratio, i.e. ratio of runs with stalls, with 50 bandwidth profiles of 1 min generated by dividing the lowest 5 10-min bandwidth profiles. As shown in Figure 15, we have the following observations.

- The stall ratio depends on both the startup buffer duration and the segment duration. With the same startup buffer duration of 8 s, the stall ratio with segments of 4 s is only 57.7% of the ratio with segments of 8 s.
- Compared with using 1 segment as startup buffer, using 2 or 3 segments significantly reduces the stall possibility. In all video settings, the stall ratio for using 3 segments is less than 41.7% of the stall ratio for 1 segment.
- Using a higher bitrate track as startup track can significantly increase stall possibility, especially when startup buffer is only 1 segment. With the startup buffer duration set to be 4s, when increasing the startup track bitrate from 0.5Mpbs to 1Mbps, the stall ratio increases from 60.0% to 91.1%.

Our findings suggest that apps should set the startup buffer duration to 2 to 3 segments. We check the implementation of ExoPlayer. The startup buffer duration is a static value in seconds which developers can configure. *We suggest the player should enforce the startup buffer threshold both in terms of duration and segment count. The startup track bitrate should also be relatively low to avoid stalls.* Similar suggestions can be also applied to the logic when the player recovers from stall events.

5 RELATED WORK

QoE characterization of commercial video streaming systems. Some existing works make effort to characterize streaming QoE of video services. However, none of existing methodologies can be generally applied to the mobile VOD services we study. Some studies [28, 31] extract bitrate information from the request URL based on certain URL patterns. However, the URL pattern differs between services and many services even do not have such patterns. For example, we find that Netflix and Amazon do not directly put bitrate information in the URL. Akhshabi et al. [17] estimate the segment duration based on their sizes. However, we find that many video services use variable bitrate (VBR) encoding. Even for the same track, the actual bitrates of segments vary significantly. Recent studies [19, 21] extract QoE information from statistical reports sent from the client to servers for certain players. However, this can not be generalized to other services. Some other works [20, 21, 46, 47] propose to apply machine learning techniques to get QoE information from network features such as delay and throughput, but it unavoidably introduces errors. A recent work [18] proposes to get initial loading time and stall time from UI events, but it needs app-specific instrumentation and does not reveal bitrate information. Some other studies[29, 49, 51, 53] examine telephony systems and live multimedia streaming systems. We study popular mobile video-on-demand services.

Proposal of novel adaptation algorithms. Many prior works [27, 31, 33, 37, 39, 50, 52] have investigated the opportunities for optimizing the rate adaptation algorithms. Jiang et al. [31] propose an adaptation algorithm that improves fairness between multiple video streaming applications. Li et al. [33] use a TCP-like probe approach to select video bitrate. PiStream [52] leverages physical layer information in LTE network to help predict network bandwidth and adapt video bitrate. Huang et al. [27] select video bitrate based on buffer occupancy. These state-of-the-art algorithms can help improve video streaming performance. In this paper we investigate the algorithms deployed in commercial mobile VOD systems in practice.

Diagnosis of QoE issues in video streaming systems. Prior efforts [23–26, 30, 32, 38, 42–44]have also emphasized the importance and challenges of diagnosing the performance problems of video streaming. Jiang et al. [30] propose to use clustering method over client attribute to identify root cause to video problems. A recent work [22] builds a machine learning model to perform root cause analysis for poor video QoE based on network characteristics. These works focus on identifying problems caused by external environment such as poor network condition. We focus on characterizing the QoE impact of the service design.

6 CONCLUSION

We conduct a detailed measurement study of a wide cross-section of 12 popular mobile streaming VOD services to develop a holistic understanding of their design and performance. Using carefully crafted measurements, we tease out important component designs across the end-end pipeline, including track settings, startup behavior, track switching behavior etc., and identify a number of QoE issues and their underlying causes. Using what-if-analysis, we develop best practice solutions to mitigate these challenges. By extending the understanding of how elements of service design impact QoE, our findings can help developers better navigate the design space and build mobile HAS services with improved performance.

7 ACKNOWLEDGEMENTS

We express our sincerest gratitude towards the anonymous reviewers who gave valuable feedback to improve this work, and our shepherd, Heather Zheng, for guiding us through the revisions. This work is partially funded by NSF under awards CCF-1629347, CCF-1438996, and CNS-1629894.

REFERENCES

[1] 2012. ISO/IEC 23009-1, Information technology - Dynamic adaptive streaming over HTTP (DASH). http://standards.iso.org/ittf/PubliclyAvailableStandards/c057623_ISO_IEC_23009-1_2012.zip. (2012).

[2] 2014. ExoPlayer: Adaptive video streaming on Android - YouTube. https://www.youtube.com/watch?v=6VjF638VObA. (2014).

[3] 2016. Choosing the Optimal Segment Duration. https://streaminglearningcenter.com/blogs/choosing-the-optimal-segment-duration.html. (2016).

[4] 2016. ExoPlayer 2 - Why, what and when? https://medium.com/google-exoplayer/exoplayer-2-x-why-what-and-when-74fd9cb139. (2016).

[5] 2016. ExoPlayer from the other side. https://medium.com/google-exoplayer/exoplayer-from-the-other-side-5909553abae2. (2016).

[6] 2016. WhatsApp For Android Devices. https://tech.blorge.com/2016/09/23/whatsapp-2-16-274-download-available/-android-devices-new-emojis/155538. (2016).

[7] 2017. BBC DASH Testcard Stream. http://rdmedia.bbc.co.uk/dash/ondemand/testcard/. (2017).

[8] 2017. Building Periscope for Android. http://nerds.airbnb.com/building-periscope-for-android/. (2017).

[9] 2017. CBR and VBR Encoding FAQ: What is The Difference? https://www.lifewire.com/difference-between-cbr-and-vbr-encoding-2438423. (2017).

[10] 2017. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html. (2017).

[11] 2017. Curl: command line tool and library for transferring data with URLs. https://curl.haxx.se. (2017).

[12] 2017. Experience Shapes Mobile Customer Loyalty - Ericsson. https://www.ericsson.com/thinkingahead/consumerlab/consumer-insights/experience-shapes-mobile-customer-loyalty. (2017).

[13] 2017. How Xposed works. https://github.com/rovo89/XposedBridge/wiki/Development-tutorial. (2017).

[14] 2017. Sintel - Open Movie by Blender Foundation. https://durian.blender.org/download/. (2017).

[15] 2017. Smooth Streaming Protocol. https://msdn.microsoft.com/en-us/library/ff469518.aspx. (2017).

[16] 2017. Technical Note TN2224. https://developer.apple.com/library/content/technotes/tn2224/_index.html#//apple_ref/doc/uid/DTS40009745-CH1-BITRATERECOMMENDATIONS. (2017).

[17] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. 2011. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 157–168.

[18] Qi Alfred Chen, Haokun Luo, Sanae Rosen, Z Morley Mao, Karthik Iyer, Jie Hui, Kranthi Sontineni, and Kevin Lau. 2014. Qoe doctor: Diagnosing mobile app qoe with automated ui control and cross-layer analysis. In *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 151–164.

[19] Luca De Cicco and Saverio Mascolo. 2010. An experimental investigation of the Akamai adaptive video streaming. In *Symposium of the Austrian HCI and Usability Engineering Group*. Springer, 447–464.

[20] Giorgos Dimopoulos, Pere Barlet-Ros, and Josep Sanjuas-Cuxart. 2013. Analysis of youtube user experience from passive measurements. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. IEEE, 260–267.

[21] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. 2016. Measuring Video QoE from Encrypted Traffic". In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 513–526.

[22] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, Konstantina Papagiannaki, and Peter Steenkiste. 2015. Identifying the Root Cause of Video Streaming Issues on Mobile Devices. In *Proceedings of the 2015 ACM International Conference on Emerging Networking Experiments and Technologies: 1-4 December, 2015: Heidelberg, Germany*. Association for Computing Machinery (ACM).

[23] Mojgan Ghasemi, Partha Kanuparthy, Ahmed Mansy, Theophilus Benson, and Jennifer Rexford. 2016. Performance Characterization of a Commercial Video Streaming Service. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM.

[24] Yihua Guo, Ashkan Nikravesh, Z Morley Mao, Feng Qian, and Subhabrata Sen. 2017. Accelerating Multipath Transport Through Balanced Subflow Completion. In *Proceedings of the 23th annual international conference on Mobile computing and networking*. ACM.

[25] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding on-device bufferbloat for cellular upload. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 303–317.

[26] Gonca Gürsun, Mark Crovella, and Ibrahim Matta. 2011. Describing and forecasting video access patterns. In *INFOCOM, 2011 Proceedings IEEE*. IEEE, 16–20.

[27] TY Huang, R Johari, N McKeown, M Trunnell, and M Watson. 2014. A buffer-based approach to rate adaptation. In *Proc. 2014 ACM Conference on SIGCOMM, SIGCOMM*, Vol. 14. 187–198.

[28] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. 2012. Confused, timid, and unstable: picking a video streaming rate is hard. In *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 225–238.

[29] Yunhan Jack Jia, Qi Alfred Chen, Zhuoqing Morley Mao, Jie Hui, Kranthi Sontinei, Alex Yoon, Samson Kwong, and Kevin Lau. 2015. Performance characterization and call reliability diagnosis support for voice over lte. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 452–463.

[30] Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. 2013. Shedding light on the structure of internet video quality problems in the wild. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 357–368.

[31] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. 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*. ACM, 97–108.

[32] S Shunmuga Krishnan and Ramesh K Sitaraman. 2013. Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking* 21, 6 (2013), 2001–2014.

[33] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. 2014. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications* 32, 4 (2014), 719–733.

[34] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Rate adaptation for adaptive HTTP streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 169–174.

[35] Yao Liu, Sujit Dey, Fatih Ulupinar, Michael Luby, and Yinian Mao. 2015. Deriving and validating user experience model for dash video streaming. *IEEE Transactions on Broadcasting* 61, 4 (2015), 651–665.

[36] Ahmed Mansy, Mostafa Ammar, Jaideep Chandrashekar, and Anmol Sheth. 2014. Characterizing client behavior of commercial mobile video streaming services. In *Proceedings of Workshop on Mobile Video Delivery*. ACM, 8.

[37] RKP Mok, EWW Chan, and RKC Chang. 2011. Improving TCP video streaming QoE by network QoS management. (2011).

[38] Ricky KP Mok, Edmond WW Chan, Xiapu Luo, and Rocky KC Chang. 2011. Inferring the QoE of HTTP video streaming from user-viewing activities. In *Proceedings of the first ACM SIGCOMM workshop on Measurements up the stack*. ACM, 31–36.

[39] Ricky KP Mok, Weichao Li, and Rocky KC Chang. 2016. IRate: Initial video bitrate selection system for HTTP streaming. *IEEE Journal on Selected Areas in Communications* 34, 6 (2016), 1914–1928.

[40] Hyunwoo Nam, Bong Ho Kim, Doru Calin, and Henning Schulzrinne. 2013. A mobile video traffic analysis: Badly designed video clients can waste network bandwidth. In *Globecom Workshops (GC Wkshps), 2013 IEEE*. IEEE, 506–511.

[41] Ana Nika, Yibo Zhu, Ning Ding, Abhilash Jindal, Y Charlie Hu, Xia Zhou, Ben Y Zhao, and Haitao Zheng. 2015. Energy and performance of smartphone radio bundling in outdoor environments. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 809–819.

[42] Ashkan Nikravesh, Yihua Guo, Feng Qian, Z Morley Mao, and Subhabrata Sen. 2016. An in-depth understanding of multipath TCP on mobile devices: Measurement and system design. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 189–201.

[43] Ashkan Nikravesh, David Ke Hong, Qi Alfred Chen, Harsha V Madhyastha, and Zhuoqing Morley Mao. 2016. QoE Inference Without Application Control.. In *Internet-QoE@ SIGCOMM*. 19–24.

[44] Ashkan Nikravesh, Hongyi Yao, Shichang Xu, David Choffnes, and Z Morley Mao. 2015. Mobilyzer: An open platform for controllable mobile network measurements. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 389–404.

[45] Roger Pantos and William May. 2016. HTTP live streaming. (2016).

[46] Raimund Schatz, Tobias Hoßfeld, and Pedro Casas. 2012. Passive youtube QoE monitoring for ISPs. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 358–364.

[47] Muhammad Zubair Shafiq, Jeffrey Erman, Lusheng Ji, Alex X Liu, Jeffrey Pang, and Jia Wang. 2014. Understanding the impact of network dynamics on mobile video user engagement. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42. ACM, 367–379.

[48] Christian Sieber, Poul Heegaard, Tobias Hoßfeld, and Wolfgang Kellerer. 2016. Sacrificing efficiency for quality of experience: YouTube's redundant traffic behavior. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*. IEEE, 503–511.

[49] Matti Siekkinen, Enrico Masala, and Teemu Kämäräinen. 2016. A First Look at Quality of Mobile Live Streaming Experience: the Case of Periscope. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 477–483.

[50] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2016. BOLA: near-optimal bitrate adaptation for online videos. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[51] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y Zhao. 2016. Anatomy of a personalized livestreaming system. In *Proceedings of the 2016 ACM on Internet Measurement Conference*. ACM, 485–498.

[52] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. piStream: Physical layer informed adaptive video streaming over LTE. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 413–425.

[53] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. 2012. Video telephony for end-consumers: measurement study of Google+, iChat, and Skype. In *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 371–384.

[54] Tong Zhang, Fengyuan Ren, Wenxue Cheng, Xiaohui Luo, Ran Shu, and Xiaolan Liu. 2017. Modeling and Analyzing the Influence of Chunk Size Variation on Bitrate Adaptation in DASH. In *Computer Communications, IEEE INFOCOM 2017-The 36th Annual IEEE International Conference on*. IEEE.