A Public Dataset for YouTube's Mobile Streaming Client

Theodoros Karagkioules*†, Dimitrios Tsilimantos*, Stefan Valentin*,
Florian Wamser‡, Bernd Zeidler‡, Michael Seufert‡, Frank Loh‡ and Phuoc Tran-Gia‡
*Paris Research Center, Huawei Technologies France
†LTCI, Télécom ParisTech, Université Paris-Saclay
‡University of Würzburg, Germany
Email: firstname.lastname@{huawei.com*, telecom-paristech.fr†, informatik.uni-wuerzburg.de‡}

***Abstract*—Datasets are a valuable resource to analyze, model and optimize network traffic. This paper describes a new public dataset for YouTube's popular video streaming client on mobile devices. At the moment, we are providing 374 hours of time-synchronous measurements at the network, transport and application layer from two controlled environments in Europe. After describing our experimental design in detail, we discuss how to use our dataset for the analysis and optimization of HTTP Adaptive Streaming (HAS) traffic and point to specific use cases. To assure reproducibility and for community benefit, we publish the dataset at [1].**

I. Introduction

Datasets for Internet traffic are more valuable than ever for communication research. They provide important insight into network and application behavior, especially when covering large-scale Internet services. One of such services is YouTube, a video streaming platform that is generating more than 20% of the mobile data traffic [2]. According to YouTube's own statistics, its users are streaming a billion hours of video every day, with over half of these requests coming from mobile devices [3].

Interestingly, the mobile use of YouTube's service has been only rarely covered by public datasets. Instead, most available data were collected by using a personal computer as a streaming device [4]. This, however, cannot provide representative measurements for mobile use, since YouTube's Web-player (running inside a Web-browser on a personal computer) operates differently than its native application for mobile operation systems [5], [6]. Although YouTube follows the Dynamic Adaptive Streaming over HTTP (DASH) standard [7] on both platforms, it employs different adaptive streaming algorithms between the two [5]. On mobile devices, YouTube also employs the Quick UDP Internet Connections (QUIC) protocol [8] more often than on desktop devices, according to [9] and to our observations from two separate locations in Europe. The few public datasets that include measurement of mobile clients [5], [10], however, merely provide network traces. Since YouTube employs Transport Layer Security (TLS) to encrypt application-layer information, focusing on packet traces alone can only offer a limited view on the streaming process.

Our dataset aims to fill this gap by providing measurements that were simultaneously obtained at the network, transport and application layer. The data were generated with YouTube's native Android application on two Smartphone models, at two locations in Europe, for over 5 months. Currently, 374 hours of curated data are available, of which about 25% are manually labeled according to the buffer state of the player.

At the application layer, we extracted a wide range of adaptive streaming parameters from YouTube's mobile client. This was possible by our recently introduced *Wrapper App* [6], which allows remote control and monitoring of YouTube's native Android application. The source code of our tool can be downloaded at [1] along with the dataset. At the transport and network layer, we used the common tool tcpdump [11] to record unfiltered packet logs on the Smartphones and a gateway. During our entire measurement period, YouTube deployed QUIC, rather than Transmission Control Protocol (TCP), for transporting the streaming data. Consequently, most of the recorded logs refer to User Datagram Protocol (UDP) traffic.

Our aim is not to cover realistic yet loosely controlled scenarios, such as field tests in mobile networks. Instead, the focus of our dataset is on the synchronous measurement at the network, transport and application layer in tightly controlled environments. To this end, we followed the guidelines of the DASH Industry Forum [12] and designed a testbed that allows to control throughput, Packet Error Rate (PER), Round Trip Time (RTT), and streaming quality in real time. This high degree of control and its precise documentation allows to reproduce our data as well as using it for traffic modeling and factor analysis.

Consequently, our dataset can be immediately applied to analyze HAS traffic across multiple layers from the perspectives of packet management, Quality of Experience (QoE) factor analysis [13] and HTTP Adaptive Streaming (HAS) algorithm design. We will discuss further uses and point to ongoing applications of our data below.

The remainder of the paper is structured as follows. Section II documents our experimental setup and design, including our *Wrapper App*. Our measurement procedures and measured variables are described in Section III. Section IV provides an overview of the currently available data. Section V provides ongoing applications and further ideas for using our data and Section VI summarizes the paper.

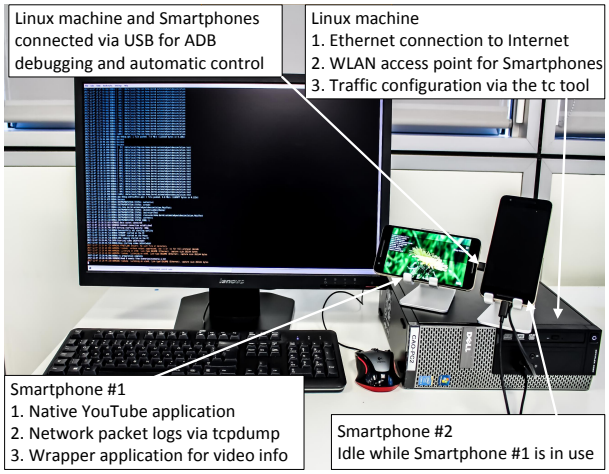

Fig. 1: Hardware setup for regulated and automatic HAS traffic measurements

II. Experimental design

In this section we describe the setup, scenarios, streaming content and location-specific differences for our experiments.

A. Setup

Our experimental setup consists of off-the-shelf consumer hardware, common tools for Linux and Android and customized measurement software for Android and Linux. For the latter, we are using our *Wrapper App* [6] to extract video information and streaming statistics from YouTube's Android player and control scripts to automatize the experiments.

The hardware setup is illustrated in Figure 1. A Linux computer (Kernel 3.16.0-71-lowlatency) controls two Android Smartphones via Universal Serial Bus (USB) connections using the Android Debug Bridge (ADB) [14]. The control computer is connected to the Internet via a T1 line and operates as an Internet-gateway and Wireless Local Area Network (WLAN) access point for the Smartphones. The Smartphones perform video streaming via an IEEE 802.11g WLAN link at a carrier frequency of 2412 MHz. Due to the close distance between phones and access point, the average Signal-to-Interference-plus-Noise Ratio (SINR) was 23 dB, which provides the maximum physical layer rate of 54 Mbit/s.

The measurements are performed on the control computer and on the Smartphones. On all the devices, network and transport-layer information is collected with tcpdump (version 4.7.4, libpcap version 1.7.4). On the Smartphones, the native YouTube application (version 12.32.60) for Android generated HAS traffic over UDP according to QUIC [8]. Although we consistently observed some TCP packets at the beginning of each session, all video streams were transported via UDP/QUIC. We assume that the TCP connections were used for tracking.

The setup allows to control the throughput, Internet Protocol (IP) packet delay and PER that are experienced by the Smartphones. This *traffic control* is done at the control computer via Python scripts that adjust a Token Bucket Filter (TBF) in the Linux kernel [15]. The scripts can automatically modify the above network parameters during an experiment at predetermined or random points in time, which allows to emulate the dynamics of a mobile network in a reproducible manner.

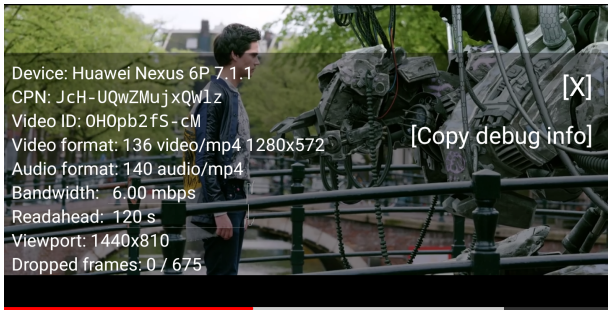

Fig. 2: Screen-shot from the YouTube Android application during measurement with *stats for nerds*

In order to control and to extract application-layer information we designed the *Wrapper App* [6] – a measurement tool for the native YouTube Android application. In principle, our tool records application-layer information that is available through the *stats for nerds* feature of YouTube's Android application. As shown in Figure 2, this feature offers the values of some streaming-related parameters and the option to copy them to other applications via the clipboard. The *Wrapper App* accesses this information basically by executing touch-and-swipe events on the Smartphone, a process which is remote-controlled by the measurement scripts with the help of ADB. In addition, the *Wrapper App* uses Android's *User Interface (UI) automator* [16] to access the video progress bar.

These measurements are started and recorded periodically on the control computer. A measurement cycle is initialized by a control script, which starts the *Wrapper App*. This tool then launches the YouTube application and configures it according to the parameters in the scenario-specific configuration file (e.g., deactivating auto-play, choosing a fixed quality). The *Wrapper App* then starts the measurement by requesting the specified video.

It is important to note that the *Wrapper App* records only the mentioned application-layer measurements but no content (such as video, image data and audio).

B. Scenarios

To reduce the experimental design space and to structure our dataset, we define the 8 scenarios in Table I. More scenarios can be easily defined in individual configuration files.

Beginning with the simpler scenarios, no traffic control is performed in scenarios (s1) to (s3). Scenarios (s1) and (s2) perform video streaming at a constant quality. In (s3) a manual quality change is introduced at a random time in $t_0 \in [120, 240]$ s. This time window is chosen to be in the middle of the video duration, which is specified in Table II, to assure that the client can reach its target buffer level.

In (s4) to (s8), various forms of traffic control are activated. In (s4) a rate limitation at 500 kbit/s is configured, starting at

TABLE I: HAS scenarios

Scenario	Quality	Rate limit
s1	480p	No
s2	720p	No
s3	720p$\xrightarrow{t_0}$480p $t_0 \in [120, 240]\ s$	No
s4	Auto	No $\xrightarrow{t_0}$ 500 kbit/s $\xrightarrow{t_1}$ No $t_0 \in [120, 240]$s, $t_1 = t_0 + 150$
s5	Auto	1024 kbit/s
s6	Auto	(see Figure 3)
s7	Auto	No $\xrightarrow{120s}$ 3000 kbit/s $\underset{t_1}{\overset{t_0}{\rightleftarrows}}$ 100 kbit/s $t_0 = 160 + 85n, n \in \mathbb{N}$ $t_1 = 205 + 85n, n \in \mathbb{N}$
s8	Auto	No $\underset{t_1}{\overset{t_0}{\rightleftarrows}}$ 100 kbit/s $t_0 = \{120, 300\}$s, $t_1 = \{180, 380\}$s

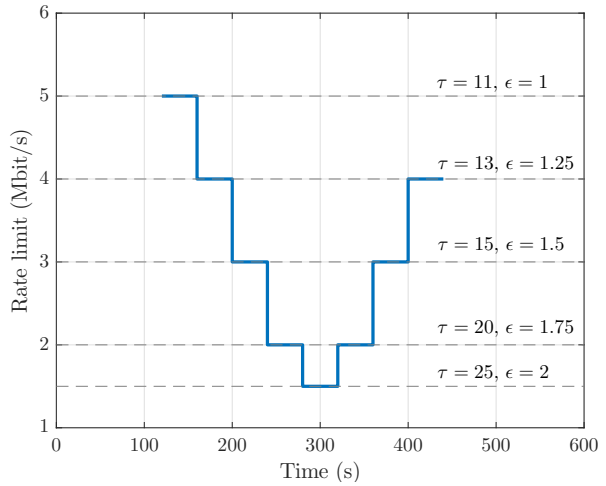

Fig. 3: Traffic configuration of rate (Mbit/s), delay τ (ms) and PER ϵ (%) for (s6) according to DASH-IF guidelines

a random time in the interval $[120, 240]$ s with a total duration of 150 s. In this scenario, we configure the YouTube client to choose the streaming quality automatically according to its built-in HAS policy. Scenario (s5) also allows video quality adaptation but at a constant throughput limit of 1024 kbit/s.

Scenario (s6) implements a test case specified by the DASH Industry Forum (DASH-IF) in [12, Table 5] with quality adaptation. The stepwise traffic control in this scenario is illustrated in Figure 3. Finally, (s7)-(s8) emulate cases with steep throughput drops to 100 kbps, where even the lowest video quality is not supported. These scenarios allow to study the behavior of adaptive streaming policies in the case of coverage loss due to shadowing or handover failures in mobile networks.

C. Streaming content

In order to provide data for representative videos, we chose 3 clips whose average encoding rates correspond to the average rates in [17, Table 4]. Table II provides the average μ and the standard deviation σ of the encoding rate for each streamed quality of the selected videos.

TABLE II: Streaming content and video bit-rate statistics (mean μ and standard deviation σ)

Video	Duration (s)		Bit-rate (kbit/s)					
			144p	240p	360p	480p	720p	1080p
TOS	734	μ	107	240	346	715	1347	2426
		σ	35	78	175	346	641	1152
Nature	561	μ	108	242	398	792	1566	3009
		σ	48	109	231	424	794	1381
TalkShow	559	μ	52	102	282	600	1156	2324
		σ	29	54	137	267	492	919

These videos represent different types of content. The movie Tears of Steel (*TOS*), with YouTube video ID is "OHOpb2fS-cM" [18], contains a mixture of computer generated and natural images in high motion. The clip is commonly used for testing video codecs and recommended in DASH-IF's specification [12, Section 2.1]. As a second clip we chose a nature documentary (*Nature*) [19] with YouTube video ID "2d1VrCvdzbY". This video consists of complex natural scenes (e.g., clouds, trees and water surfaces). Finally, we chose a talk-show (*TalkShow*) [20] as a low-motion clip with YouTube video ID "N2sCbtodGMI".

None of these videos is monetized, which prevents interruptions through advertisements. *TOS*, *TalkShow* and *Nature* are encoded at 24, 25 and 30 fps, respectively and are all available at the horizontal resolutions {144p–1080p}. Each of these qualities is available in two representations. First, as a H.264-encoded stream of DASH segments in an MP4 container and, second, as a VP9-encoded stream in a WebM container. YouTube's streaming application automatically chooses the format and indicates the chosen representation by an 'itag' field that is available in our dataset. The 'itag' index points to a specific field in the DASH manifest file, from which the content URL, format, URL, peak bitrate and other meta-data can be extracted according to the DASH specification [7].

D. Location-specific differences

The measurements were performed at two locations in Europe, i.e. Paris in France and Würzburg in Germany. While the main setup, procedures and scenarios were identical, two relevant differences between the setups were inevitable.

First, the performance of the T1 uplink to the Internet necessarily differed between Würzburg and Paris. While Würzburg's uplink provided 30% higher average throughput, it also came with a 26% higher average RTT. Our measurement results for those links are summarized in Table III.

The second difference was the used Smartphone model. In Paris, two identical phones of the type Huawei Nexus 6P (Model H1512, baseband version: angler-03.78) were used alternately. Switching between these phones was necessary since, despite line power and a deactivated display, the measurements depleted the battery of the active phone faster than it

TABLE III: Mean μ and standard deviation σ for 10000 measurements of RTT and 1954 measurements of throughput

Location	RTT (ms)		Throughput (Mbit/s)	
	μ	σ	μ	σ
Paris	18.64	24.17	9.07	2.17
Würzburg	25.19	34.41	13.01	4.63

could recharge. Using two phones allowed the inactive phone to recharge in order to provide continuous experiments. In Würzburg, the Smartphone Google Pixel XL (Model marlin, baseband version: 8996-012511-1611190200) was used. This device did not suffer from battery depletion, making multiple phones unnecessary at this location. Note that both locations used the same version of the Android operation system, i.e., version 7.1.1 with the security patch from December 5$^{\text{th}}$, 2016.

Although we observed no significant influence of these location-specific differences, the reader should be aware that their effect was not systematically studied.

III. Data collection

In this section, the measurement procedures at the network, transport and application layer as well as the labeling process, are described in detail.

A. Network and transport-layer information

On the control computer and on the Android phones, we use the traffic capturing tool tcpdump to record TCP, UDP and IP information. Contrary to packet sniffing, we are using tcpdump not to record payload but to log meta data from the IP and UDP/TCP headers. For each packet that passes the TCP/IP stack, we configure tcpdump to produce a new record in a log file that contains a timestamp and the fields "*tos*: type of service, *ttl*: time to live, *id*: IP ID, *offset*: fragment offset, *flags*: fragmented diagram, *proto*: protocol type, *length*: packet length" and "source IP $>$ destination IP: protocol, packet length". Packet length is given in bytes and the timestamp is in Unix epoch time (i.e. seconds after 00:00 UTC on 1 January 1970). Table IV shows the most relevant parameters.

Note that we apply no filter to tcpdump, which means that we are logging all IP packets in both directions (ingoing as well as outgoing). Although all packet types are logged, YouTube's Android client employed QUIC for streaming. Thus, most of the recorded traces refer to UDP packets.

B. Application-layer information

We measure information at the application layer with two methods. First, we use our YouTube *Wrapper App* to extract information from YouTube's Android application itself. As described in Section II-A, our tool transfers information from YouTube's statistic module and the video progress bar to the control computer. Video progress is recorded every 0.5 s in the file *Video_progress*, while the statistics are also extracted twice per second but stored in a file named *Statistics*. For each experiment, the *Wrapper App* logs its control events in the *event_log* file. A selection of the recorded variables is summarized in Table IV.

TABLE IV: Selection of recorded parameters relevant to HAS

Layer	File identifier	Parameter
Network	TCPdump	packet arrival time (s)
	TCPdump	packet source IP address
	TCPdump	packet source port number
	TCPdump	packet destination IP address
	TCPdump	packet destination port number
Transport	TCPdump	packet payload size (Bytes)
	TCPdump	packet transport protocol
Application	DNS	query source IP address
	DNS	query source port number
	DNS	query destination IP address
	DNS	query destination port number
	Statistics	"bh": buffer level (ms)
	Statistics	video ID
	Statistics	"fmt": video quality (itag)
	Statistics	"afmt": audio quality (itag)
	Statistics	"bwe": bandwidth estim. (bit/s)
	Video_progress	elapsed video time (s)
	Event_log	network configuration events
	Event_log	Video play-out events

Our second method is to record information from Domain Name System (DNS)-queries that are initiated by the YouTube application. These data are recorded by tcpdump and saved in the file *DNS*. This information allows to measure the initial delay by calculating the time difference between the playback start and the DNS initial request.

In order to illustrate some of the data that were recorded from the YouTube application during a streaming session, Figure 4 plots the level of the play-out buffer in seconds (left y-axis and both 'itag' lines) and the result of the bandwidth estimation (bwe) in Mbit/s over the session time. We observe that the adaptive streaming client initially chooses a quality of 720p (itag 136), which maintains the buffer level at around 120 s. After throughput throttling ($t = 152$ s), the play-out buffer leaves the steady state and starts depleting. YouTube's bandwidth estimation adjusts to this buffer depletion at $t = 208$ s, leading to the selection of the 144p quality (itag 160) at $t = 290$ s. This drastic decrease in video quality allows to quickly fill up the play-out buffer and, thus, avoids stalls. After our control script has removed the throughput limit at $t = 303$ s, YouTube's bandwidth estimation recovers at $t = 336$ s and the client requests 720p-segments again.

C. Labels

By analyzing the previous network and application logs, we are also able to distinguish different states of the play-out buffer. These states are fundamental for the behavior of adaptive streaming clients and are, thus, useful to study and design HAS traffic. We identify the 4 different states *filling, steady, depleting* and *unclear*. In the states *filling* or *depleting*, the client's download rate is above or below the video bitrate, respectively. In the *steady* state, however, the buffer level is approximately constant and the client's

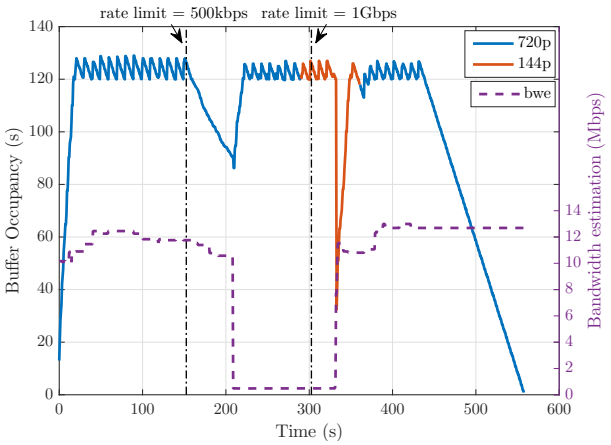

Fig. 4: Example of data from *Statistics* and *Event_log* files, for video [20], under experiment scenario (s4).

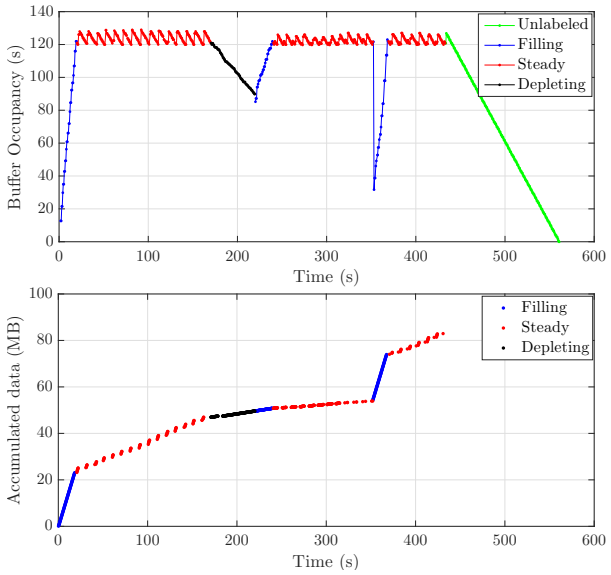

Fig. 5: Example of labeled streaming video [20], using the setup in Section II under experimental scenario (s4).

download rate is approximately equal to the video bitrate [21, Sec. 2.2]. Adaptive streaming clients aim to achieve the steady state by adapting video quality and segment request time. Interested readers are referred to [21], [22] to learn more about adaptive streaming algorithms and buffer states.

We labeled the buffer states manually with the help of a graphical user interface (GUI), that we developed for this reason, which provides plots as in Figure 5. The top figure shows the buffer level in seconds, as recorded in the *Statistics* log file. The bottom figure displays the accumulated data over time for the streaming data, as recorded in the *TCPdump* log file. By inspecting these plots, the GUI allows a user to:

- plot accumulated data separately per packet flow,
- specify time intervals for independent label assignment,
- associate a label with a previously defined interval.

The result of this process is a separate log file with timestamps and label values, that contains the string *Labels* in its name.

IV. Dataset

Let us now provide an overview to the structure and the content of the dataset [1].

Using the setup and tools in Section II and the methodology in Section III, the data were recorded in Paris, France and Würzburg, Germany, over the course of 5 months. The measurement campaign started on the 19th of September 2017 and ended on the the 23rd of February 2018.

During this campaign, 2201 experiments were performed. Each experiment is defined by its scenario index (s1 to s8), the video ID and the iteration index. Overall, 374 hours of streaming traffic were recorded, 25% of which are labeled. Table V provides the numbers of experiments per scenario and number of labeled experiments in parentheses.

The data are provided as text files in UTF-8 character encoding and csv-format. The directory structure is `./$Location/Scenario_$n/Vid_$v/Iteration_$m`, where Location $= \{$'Paris', 'Wurzburg'$\}$, $n = \{1, 2, \ldots, 8\}$, video ID v according to Section II-C and $m = \{1, 2, \ldots, M\}$ with iteration index M according to Table V. For example, the

TABLE V: Number of total experiments and number of labeled experiments in parentheses

Scenario	**Video**			
	Nature	TalkShow	TOS	All videos
s1	90 (21)	102 (33)	109 (32)	301 (86)
s2	86 (17)	97 (27)	108 (31)	291 (75)
s3	80 (18)	94 (27)	108 (32)	282 (77)
s4	90 (25)	103 (35)	119 (41)	312 (101)
s5	63 (10)	82 (19)	102 (31)	247 (60)
s6	86 (32)	107 (40)	120 (43)	313 (115)
s7	81 (11)	86 (6)	87 (6)	254 (23)
s8	69 (7)	65 (3)	67 (4)	201 (14)
Total number	645 (141)	736 (190)	820 (220)	2201 (551)
Total time (h)	132 (29)	114 (30)	128 (34)	374 (93)

directory `./Paris/Scenario_1/Vid_2d1VrCvdzbY/Iteration_1` contains the data of the first iteration of the *Nature* video according to scenario (s1) in Paris. Each directory `Iteration_$m` contains the csv-files, named according to the type of data {'Labels, 'DNS', 'TCPdump', 'event_log', 'Statistics', 'Video_progress'}, as described in Section III. Some file names also indicate the device {'PC', 'Phone'}, as these measurements are simultaneously performed on the control computer ('PC') and on the Smartphone. Note that we, redundantly, add \$Location, $\$n$, $\$v$ and $\$m$ to the file names for simplicity and organization.

Within each file, every line starts with a Unix timestamp [23], that is provided by the respective measurement device {'PC', 'Phone'}. Although the devices were synchronized via Network Time Protocol (NTP), we provide the file called *time_log* that contains both time references at the moment of initiation of the measurement.

V. Applications

Our dataset provides time-synchronized measurements at the ISO/OSI Layers 3, 4 and 7 in controlled scenarios. While these data may be used in various ways, let us now briefly point to specific applications for (i) designing adaptive streaming clients, (ii) analyzing streaming traffic at packet level and (iii) estimating streaming parameters and quality.

First, our dataset allows to study and to reproduce how a popular adaptive streaming client reacts to variable link conditions. By mapping the scenario parameters from Table I to the measures at Layer 7 (cp. Table IV) algorithm designers can analyze, for instance, the reaction time and video quality of YouTube's client depending on throughput, PER and delay. Since our measurement tools are publicly available [1], [6], researchers can even reproduce our testbed and compare the performance of their own bitrate adaptation algorithms against YouTube's adaptive streaming policy. In the same manner, our tcpdump-logs can be used to study and to improve the packet generation of adaptive streaming clients compared to YouTube.

Second, our dataset allows a detailed analysis of adaptive streaming traffic at Layer 3 and 4, depending on various network and video-related factors. A good statistical understanding at the packet level is particularly relevant with the deployment of new transport protocols such as QUIC. Since the majority of our packet logs capture QUIC traffic, our data help operators and vendors to customize their networks for this new kind of traffic. One example of such customization is service classification based on packet arrival patterns, which we developed based on the presented dataset and described in [24]. Such real-time classification systems can then be used for streaming-specific billing and traffic shaping as in T-Mobile's Binge On [25] or to custom-tailor admission control and resource allocation [26] in cellular networks.

Third, our dataset allows to estimate application-layer parameters and quality, based on packet flows. In [27], we used a previous version of this dataset to estimate video encoding rate only by observing the size and arrival time of the related IP packets. Such simple mechanisms already provide sufficient accuracy for adjusting resource allocation functions [26] and for estimating QoE [13]. Similarly, our dataset can be used to estimate further QoE factors such as initial buffering delay, adaptation frequency as well as stalling frequency and delay.

VI. Summary

We described our public [1] dataset for YouTube's mobile streaming client and the methods to reproduce it.

Our experimental setup allows to design controlled experiments with automatic variation of network and streaming parameters during a video session. Besides measuring network statistics, streaming variables are extracted from YouTube's native Android application. Our procedures support the test cases of the DASH Industry Forum [12] and provide state labels for YouTube's adaptive streaming logic.

At the moment, our dataset offers 374 hours of synchronous measurements at the network, transport and application layer in two controlled environments. All of these data are generated by YouTube's native Android application and most of the recorded logs refer to QUIC and UDP traffic.

We are already using this dataset to analyze adaptive streaming traffic and to design traffic managing functions for cellular networks [24]. We believe that publishing our data and tools will enable the research community to better understand how to model, manage and control adaptive streaming traffic.

References

[1] T. Karagkioules et al. (2018) A public dataset for YouTube's mobile streaming client. Open dataset. [Online]. Available: http://qoecube.informatik.uni-wuerzburg.de/

[2] Sandvine, "Global Internet phenomena: Latin America & North America," *Report*, Jun. 2016.

[3] YouTube. (2018) Youtube for press. [Online]. Available: https://www.youtube.com/yt/about/press/

[4] S. Alcock et al. (2015) Analysis of YouTube application flow control. Dataset. [Online]. Available: https://wand.net.nz/ salcock/youtube/

[5] A. Rao et al., "Network characteristics of video streaming traffic," in *Proc. ACM CoNEXT*, Dec. 2011.

[6] M. Seufert et al., "A wrapper for automatic measurements with YouTube's native Android app," in *Proc. IEEE TMA*, Jun. 2018.

[7] ISO/IEC, "Dynamic adaptive streaming over HTTP (DASH)," *International Standard 23009-1:2014*, May 2014.

[8] J. Roskind, "QUIC: Multiplexed stream transport over UDP," *Design Document and Specification Rationale*, Dec. 2013.

[9] A. Langley et. al., "The quic transport protocol: Design and internet-scale deployment," in *Proc. ACM SIGCOMM*, Jun. 2017.

[10] S. Sengupta et al. (2015) Crawdad dataset (v. 2015-11-26). Dataset. [Online]. Available: https://crawdad.org/iitkgp/apptraffic/20151126/

[11] The tcpdump group. (2015) tcpdump packet analyzer v. 4.7.4, (libpcap v. 1.7.4). Manual page. [Online]. Available: http://www.tcpdump.org/

[12] DASH Industry Forum, "Guidelines for implementation: DASH-AVC/264 test cases and vectors," *Report*, Jan. 2014.

[13] F. Wamser et al., "Modeling the YouTube stack: from packets to quality of experience," *Computer Networks*, Apr. 2016.

[14] developer.android.com. (2017) Android debugging tool (adb)-Android SDK platform. Manual page. [Online]. Available: https://developer.android.com/studio/command-line/adb.html

[15] A. N. Kuznetsov. (2001) Linux traffic configuration tool (tc). Manual page. [Online]. Available: https://linux.die.net/man/8/tc

[16] developer.android.com. (2017) Android UI automator-Android SDK platform. Manual page. [Online]. Available: https://developer.android.com/training/testing/ui-automator.html

[17] C. Kreuzberger et al., "A comparative study of DASH representation sets using real user characteristics," in *Proc. ACM NOSSDAV*, 2016.

[18] Blender Foundation. (2013) Tears of steel. Video. [Online]. Available: https://www.youtube.com/watch?v=OHOpb2fS-cM

[19] National Institute of Fundamental Studies Sri Lanka. (2016) Science copies nature's secrets. Video. [Online]. Available: https://www.youtube.com/watch?v=2d1VrCvdzbY

[20] Zweites Deutsches Fernsehen. (2017) Heute show. Video. [Online]. Available: https://www.youtube.com/watch?v=N2sCbtodGMI

[21] T. Huang et al., "A buffer-based approach to rate adaptation: Evidence from a large streaming service," in *Proc. ACM SIGCOMM*, Aug. 2014.

[22] K. Spiteri et al., "BOLA: near-optimal bitrate adaptation for online videos," in *Proc. IEEE INFOCOM*, Apr. 2016.

[23] The Open Group, "IEEE standard for information technology - Portable Operating System Interface (POSIX(R))," *IEEE Std 1003.1, 2004 Edition The Open Group Technical Standard*, Dec. 2008.

[24] D. Tsilimantos et al., "Classifying flows and buffer state for YouTube's HTTP adaptive streaming service in mobile networks," in *Proc. ACM MMSys*, Jun. 2018.

[25] A. Molavi Kakhki et al., "BingeOn under the microscope: Understanding T-Mobile's zero-rating implementation," in *Proc. ACM Internet-QoE*, Aug. 2016.

[26] D. Tsilimantos et al., "Anticipatory radio resource management for mobile video streaming with linear programming," in *Proc. IEEE ICC*, May 2016.

[27] D. Tsilimantos et al., "Traffic profiling for mobile video streaming," in *Proc. IEEE ICC*, May 2017.