Can You See What I See? Quality of Experience Measurements of Mobile Live Video Broadcasting

Can You See What I See? Quality of Experience Measurements of Mobile Live Video Broadcasting

MATTI SIEKKINEN*, Aalto University
TEEMU KÄMÄRÄINEN, Aalto University
LEONARDO FAVARIO, Politecnico di Torino
ENRICO MASALA, Politecnico di Torino

Broadcasting live video directly from mobile devices is rapidly gaining popularity with applications like Periscope and Facebook Live. The Quality of Experience (QoE) provided by these services comprises many factors, such as quality of transmitted video, video playback stalling, end-to-end latency, and impact on battery life, and they are not yet well understood. In this paper, we examine mainly the Periscope service through a comprehensive measurement study and compare it in some aspects to Facebook Live. We shed light on the usage of Periscope through analysis of crawled data and then investigate the aforementioned QoE factors through statistical analyses as well as controlled small scale measurements using a couple of different smartphones and both versions, Android and iOS, of the two applications. We report a number of findings including the discrepancy in latency between the two most commonly used protocols RTMP and HLS, surprising surges in bandwidth demand caused by the Periscope app's chat feature, substantial variations in video quality, poor adaptation of video bitrate to available upstream bandwidth at the video broadcaster side, and significant power consumption caused by the applications.

CCS Concepts: • **Information systems** → **Multimedia streaming**; *Social networks*; • **Networks** → *Network measurement*; *Network protocols*; *Mobile networks*; • **Human-centered computing** → *Ubiquitous and mobile computing*;

Additional Key Words and Phrases: QoE, Facebook Live, Periscope, mobile video streaming, adaptive streaming, DASH, live video, video transmission, latency

ACM Reference format:
Matti Siekkinen, Teemu Kämäräinen, Leonardo Favario, and Enrico Masala. 2017. Can You See What I See? Quality of Experience Measurements of Mobile Live Video Broadcasting. *ACM Trans. Multimedia Comput. Commun. Appl.* 1, 1, Article 1 (September 2017), 24 pages.
https://doi.org/0000001.0000001

*The corresponding author

Author's addresses: M. Siekkinen and Teemu Kämäräinen, Dept. of Computer Science, School of Science, Aalto University, Finland; L. Favario and E. Masala, Control and Computer Engineering Department, Politecnico di Torino, Italy.

1551-6857/2017/9-ART1 $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

Mobile live video broadcasting has become a very popular class of mobile applications over the last couple of years with Periscope, Meerkat, and Facebook (FB) Live. They provide a service that enables users to transmit live video to a large number of viewers with their mobile devices. On its one year birthday, Periscope announced in March 2016 that their users watch over 110 years of live video every day [22].

These live streaming systems can be very large scale and heterogeneous and consist usually of a mixture of protocols and need to cope with varying bandwidth both at broadcaster and viewer side. Another difference to traditional live streaming applications is that these services allow users to give real-time feedback to the broadcaster. Therefore, they must provide low enough end-to-end latency from camera to viewer's screen to allow good interactive experience.

A number of factors play a role in determining the quality of experience (QoE) of using these applications. These factors include quality of transmitted video, video playback stalling due to bandwidth fluctuations, delays and buffer sizing strategy, end-to-end latency, and impact on battery life, and they are not yet well understood. In this article, we report on a measurement study of mainly the Periscope service and application. We also contrast the results to FB Live in some parts of the study.

We first measured Periscope in two ways: We crawled the ongoing live streams to analyze the usage patterns of over 200K broadcasts and we recorded a few thousand viewing sessions by automating the watching of Periscope broadcasts with an Android smartphone to examine the different QoE factors. Then, we performed controlled experiments with both Periscope and FB Live in order to understand the bitrate adaptation logic used by the applications when broadcasting a live stream. Finally, we studied the application induced energy consumption on a smartphone.

We made several interesting discoveries: 1) The Periscope application's chat feature appears to sometimes fiercely compete for bandwidth with the actual video stream because of downloading of chatting users' profile pictures. Because of this, there seems to be a limit for the access network bandwidth below which startup latency and video stalling start to increase and this limit is clearly higher than the video bitrate. 2) The delay of viewers is highly influenced by the choice of protocol between RTMP and HLS to deliver the stream and this choice depends on the popularity of the broadcast. 3) The video bitrate and quality may exhibit significant short-term variations that can be attributed to extreme time variability of the captured content. 4) Periscope applications seem to choose video bitrate according to an estimate of the available bandwidth, whereas FB Live always streams at the same target bitrate. The iOS version of the Periscope application appears to also react to a situation where the bandwidth drops but not when it increases, while the Android version does not react at all. 5) Both applications are power hungry, which is typical for such applications using most of the power hungry components, such as camera, display, network, and processors. With Periscope, the power consumption grows dramatically when the chat feature is turned on while watching a broadcast, which is caused by the way it downloads and displays profile pictures. Especially the bitrate adaptation results suggest that there is plently of room to improve the QoE of these applications.

The rest of this article is organized as follows: In Section 2, we explain mobile live video broadcasting and the two applications in detail. In Section 3, we analyze Periscope usage patterns. In Sections 4 and 5 we analyze the QoE factors, such as video stalling, latency, and video quality of Periscope in detail. In Section 6, we study the video bitrate adaptation

logic of the two applications and in Section 7 we measure their power consumption. Finally, we compare our work to related work in Section 8 before concluding in Section 9.

2 MOBILE LIVE VIDEO BROADCASTING

Many mobile live video broadcasting services have emerged in the last couple of years. Among the most popular are Periscope and Facebook Live, which are the focus of our work too. All the services enable users to broadcast live video using their mobile device for other, mainly mobile users to view it. The audience may consist of thousands of viewers, which differentiates them from video call applications and introduces extra challenges to the video delivery system with respect to scalability.

Most of these applications, including Periscope and FB Live, include features that allow the audience to give real-time feedback to the broadcaster. Hence, the end-to-end latency matters for the overall QoE with the service, both from the broadcaster and viewer perspectives. Another important QoE factor is of course video quality. As both the broadcaster and viewer have wireless and possibly mobile connectivity, variations in the achievable upstream and downstream data rates present a challenge and ideally the applications would support adaptation of the video bitrate to match the achievable data rates in one way or another.

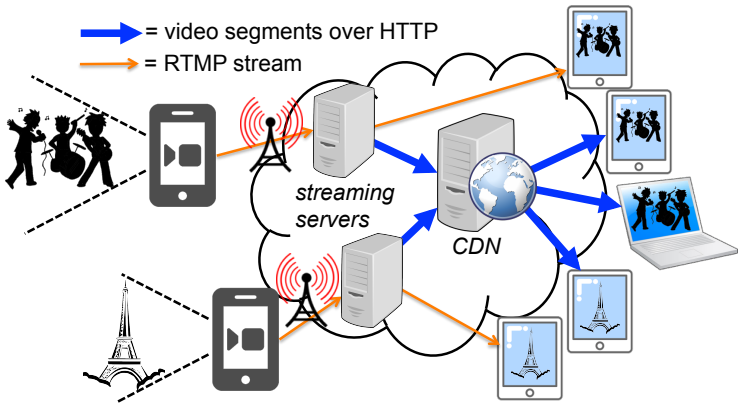


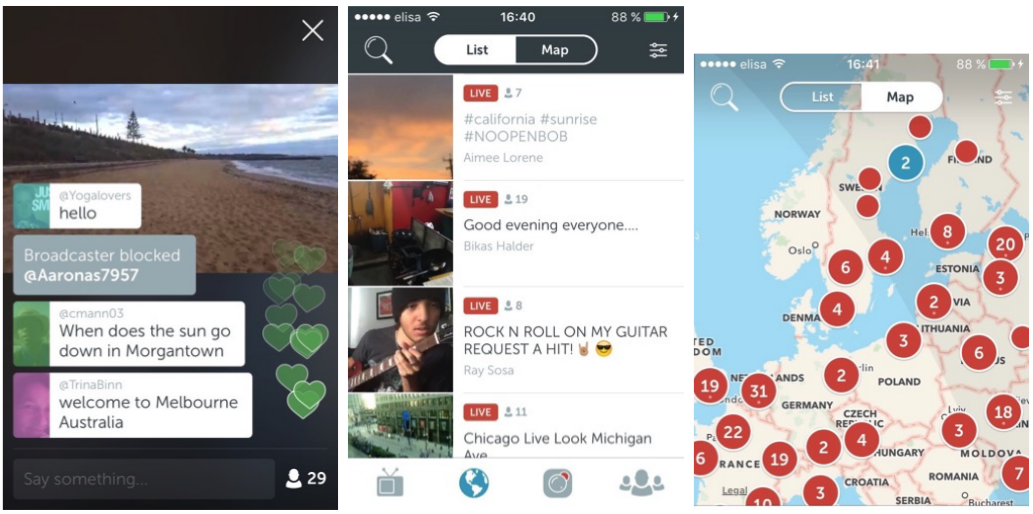
Fig. 1. Typical mobile live video broadcasting system.

Figure 1 illustrates a typical mobile live video broadcasting system. It consists of dedicated stateful streaming servers (e.g., RTMP) in different geographic locations to which the broadcasting clients transmit the live stream. The live stream then is either relayed frame by frame to clients receiving it directly from the streaming server or packaged into video and audio segments and delivered through a CDN using HTTP. The picture does not include the real-time feedback from the clients, which may also be delivered through dedicated servers (e.g., using Websockets) or with the help of a CDN.

Our measurements study focuses mainly on Periscope (Sections 3 to 5) but we also include FB Live in the investigations of video bitrate adaptation and power consumption (Section 6). We next briefly describe how the two chosen applications work.

2.1 Periscope

Periscope allows both public and private broadcasting. Private streams are only viewable by chosen users. The application supports using also a GoPro instead of the in-built camera of



(a) Live feedback from viewers (b) List of ongoing broadcasts (c) Map of broadcasts

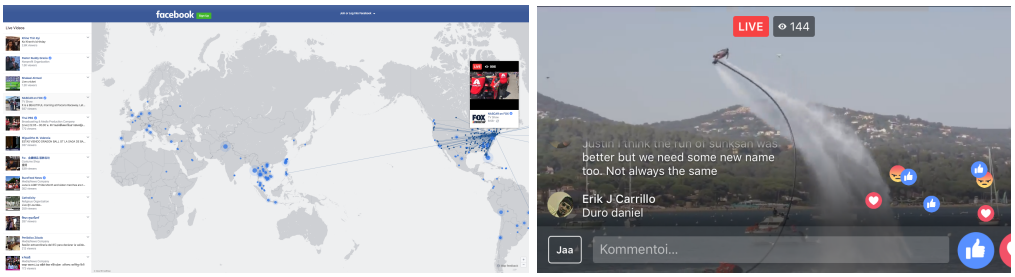
Fig. 2. Periscope screenshots.

the mobile device. Viewers can send text messages floating "hearts" as real-time feedback to the broadcaster, which is shown in Figure 2(a). The chat becomes full when certain number of viewers have joined after which new joining users cannot send messages. Broadcasts can also be made available for replay.

A user can discover public broadcasts in different ways. The application displays a list of ranked broadcasts, as shown in the screen capture in Figure 2(b), and a few featured ones. The user can also explore the map of the world in order to find a broadcast in a specific geographical region or perform a search query. Figure 2(c) shows an example. The map shows only a fraction of the broadcasts available in a large region and zooming in reveals more broadcasts. Finally, with some application versions, it is possible to click on a "Teleport" button to start watching a randomly selected broadcast.

Periscope uses two kinds of protocols for the video stream delivery: Real Time Messaging Protocol (RTMP) using port 80 and HTTP Live Streaming (HLS) because RTMP enables low latency (Section 4.5). The reason to use HLS may be related to scalability and/or costs [36].

We investigated the IP addresses and hostnames of the servers used by Periscope in more detail and discovered that the RTMP streams are always delivered from servers running on Amazon EC2 instances. For example, consider an RTMP server running at vidman-eu-central-1.periscope.tv. When we perform a reverse DNS lookup with the IP address that the application obtained when resolving that hostname, we obtain a different hostname revealing where the EC2 instance is located: ec2-54-67-9-120.us-west-1.compute.amazonaws.com. HLS video segments are delivered by Fastly CDN. We have observed segment durations between two to four seconds. RTMP streams use only one connection, whereas HLS may sometimes use multiple connections to different servers in parallel to fetch the segments, possibly for load balancing and/or resilience reasons. We study the logic of selecting the protocol and its impact on user experience in Section 4. Public streams



(a) Map of broadcasts (b) FB Live app screenshot of viewing a broadcast

Fig. 3. Periscope screenshots.

are delivered using plaintext RTMP and HTTP, whereas the private broadcast streams are encrypted using RTMPS and HTTPS for HLS. The chat uses Websockets to deliver messages.

2.2 Facebook Live

FB Live differs from Periscope in that it is an integral part of the Facebook application, not a separate application. Discovering live Facebook broadcasts also differs from Periscope. The mobile app does not have a map of world or show a list of ongoing live broadcasts, although live broadcast map is available on a web page https://www.facebook.com/livemap/, which is displayed in Figure 3(a). The usual way is through a notification that a friend or a followed person is broadcasting live. The broadcasting user can set visibility of the broadcast to be public or restricted only to friends, for instance. Similar to Periscope, the viewers can send live feedback through chat messages and emoticons, as the screenshot in Figure 3(b) shows.

FB Live apparently uses the same two protocols than Periscope[17]. We cannot confirm this through experiments because all the traffic is encrypted and the mobile app uses certificate pinning which makes it impossible to use an SSL proxy to inspect the traffic. However, viewing FB Live streams using a Web browser reveals that HTTP/2 is used to deliver the stream traffic, which we did not observe with Periscope at the time of measurements. We observed that FB Live mainly uses one second long video segments but we also noticed it to sometimes use 2s segments.

3 OVERVIEW OF USER BEHAVIOR

We first wanted to learn about the usage patterns in mobile live video broadcasting services. In this study, we focus only on Periscope. The application does not display a complete list of broadcasts and the user needs to discover broadcasts in ways described in the previous section. In late March of 2016, over 110 years of live video were watched every day through Periscope [22], which roughly translates into 40K live broadcasts ongoing all the time. We collected the data used in this analysis during the first half of 2016.

3.1 Data Collection

The Periscope app communicates with the servers using an API so that the communication is protected by SSL. Hence, we set up an SSL proxy called mitmproxy [20] in between the mobile device and the Periscope service as a transparent proxy. The proxy intercepts the

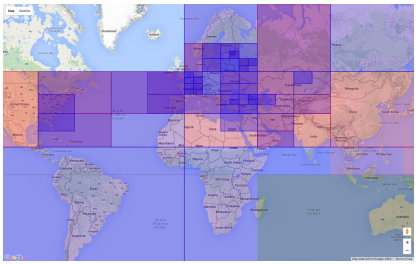
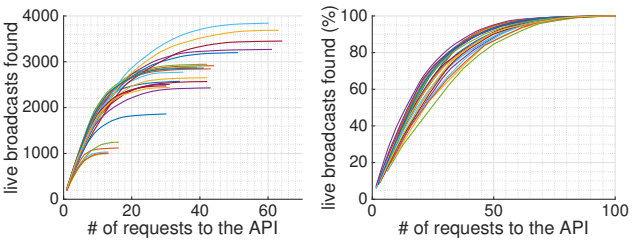


Fig. 4. Map showing the top zones selected for targeted crawl. Colour indicates how many top zone lists of the different crawls the zone appears in (dark blue is one, dark red is all).



(a) absolute (b) relative

Fig. 5. Cumulative number of broadcasts discovered as a function of crawled areas (# of requests). Each curve corresponds to a different deep crawl.

HTTPS requests sent by the mobile device and pretends to be the server to the client and to be the client to the server. The proxy enables us to examine and log the exchange of requests and responses between the Periscope client and servers. The Periscope iOS app uses the so called *certificate pinning* in which the certificate known to be used by the server is hard-coded into the client. Therefore, we only used the Android app in these experiments[1].

To collect data about user behavior, we used Android emulators (Genymotion [7]) together with the SSL proxy. We developed a so called inline script for the proxy that crawls through the service by continuously querying about the ongoing live broadcasts. The script takes advantage of Periscope API's `/mapGeoBroadcastFeed` request which allows specifying coordinates of a region to query for ongoing broadcasts. The script intercepts the initial request made by the application after being launched and replays it repeatedly in a loop with modified coordinates and writes the response contents to a file. The response to the request includes also replayable already finished broadcasts by default but our script sets the `include_replay` attribute value to false so that we only discover live broadcasts. In addition, the script intercepts `/getBroadcasts` requests and replaces the contents with the broadcast IDs found by the crawler since previous request and extracts the viewer information from the response to a file.

This approach presents some challenges: When specifying a large geographical region in the request, the response only contains a subset of all the live broadasts within that region and more broadcasts can be discovered by specifying a smaller region within the large region. This behavior is visible to the user when exploring the map of the world and zooming in. Hence, to find a large fraction of the broadcasts, the script must explore the world by specifying small enough regions. In addition, Periscope servers use rate limiting so that too frequent requests will be answered with HTTP 429 ("Too many requests"), because of which our script needs to pace the requests. This obviously increases the completion time of a crawl. The longer a crawl takes, the more broadcast information will be missed because the sampling rate of a given region decreases accordingly.

To find a suitable tradeoff between number of regions to query and crawl time, we first perform a *deep crawl* after which we select only the most active regions from that crawl and query only them, which we call a *targeted crawl*. In deep crawl, the crawler zooms into each region by dividing it into four smaller regions and recursively continues doing that

[1] At the time of writing this paper, also the Android app seems to have enabled certificate pinning.

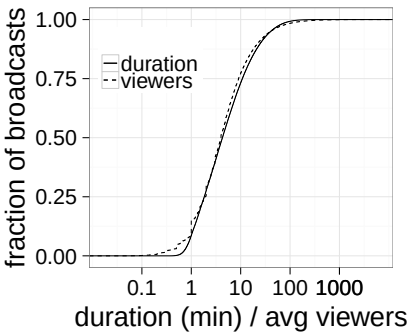


Fig. 6. Distribution of broadcast duration and number of viewers.

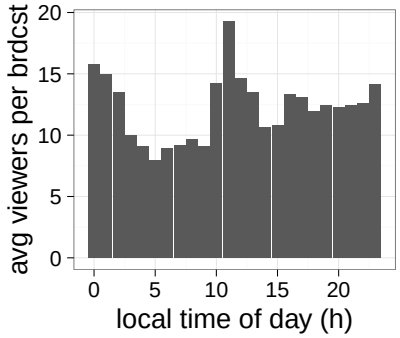


Fig. 7. Average number of viewers as a function of broadcast start time.

until it no longer discovers substantially more broadcasts. With this approach, we discover 1K-4K broadcasts per crawl[2]. Figure 5 shows the cumulative number of broadcasts found as a result of crawls performed at different times of day. Figure 5(b) reveals that for all the different crawls, half of the regions contain at least 80% of all the broadcasts discovered in the crawl. We select those regions from each crawl, 64 areas in total, for a targeted crawl. The regions are annotated in the map in Figure 4. We divide them into four sets assigned to four different simultaneously running crawlers, i.e., four emulators running Periscope with different user logged in (avoids rate limiting) that repeatedly query the assigned areas. Such targeted crawl completes in about 50s. We performed four different 4h-10h long targeted crawls started at different times of the day and only include information about broadcasts that ended during the crawl, i.e. not having been seen during the last 60s of a crawl, in the results, which gives us a total of roughly 220K distinct broadcasts. We call the resulting data set `crawl1`.

3.2 Broadcast and Viewer Statistics

Figure 6 shows CDF plots of broadcast duration and average number of simultaneous viewers during broadcasts captured in `crawl1` (note: both variables use the same x-scale). The duration was calculated by subtracting its start time, which is included in the broadcast description, from the timestamp of the last observation of the broadcast by the crawler. Most of the broadcasts last between 1 and 10 minutes and roughly half are shorter than 4 minutes. The distribution has a long tail with some broadcasts lasting for over a day. Over 90% of broadcasts have less than 20 viewers on average but some attract thousands of viewers. Over 10% of broadcasts have no viewers at all and over 80% of them are unavailable for replay afterwards (replay information is contained in the descriptions we collect about each broadcast), which means that they were never viewed by anyone. They are typically much shorter than those that have viewers (avg durations 2min vs. 13 min) although some last for hours and represent only about 2% of the total tracked broadcast time.

The local time of day shown in Figure 7 is determined based on the broadcaster's time zone. This means that it reflects local time of also those viewers located in the same time zone. We can identify some patterns in the number of viewers: There is a low point during

[2]This number is much smaller than the assumed 40K total broadcasts but we miss private broadcasts and those with location undisclosed.

the early hours of the day, a peak in the morning, and an increasing trend towards midnight. The existence of such patterns suggests that a relatively large number of viewers come from a time zone that is the same than or nearby the broadcaster's time zone, which makes sense especially from possible language preference point of view (users can change the language preferences in the app settings). Besides the difference between broadcasts with and without any viewers, the popularity is only very weakly correlated with its duration.

4 PLAYBACK SMOOTHNESS AND LATENCY

We next investigate how smooth the video playback is, in other words how frequently the video playback stalls because of empty playback buffer, and the latency perceived by users. To this end, we made smartphones automatically view Periscope broadcasts one after another, collected data during the viewing, and analyze the the collected data afterwards. We could not do these experiments with FB Live because the app does not provide suitable means for automatically discovering new broadcasts and the data collection is impossible without the use of SSL proxy.

4.1 Measurement Setup

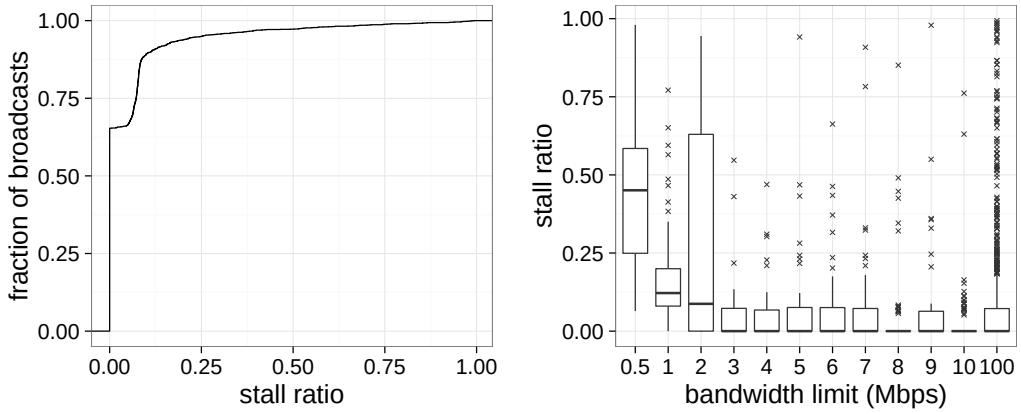
We automated the broadcast viewing process by leveraging the application's "Teleport" button which takes the user directly to a randomly selected live broadcast. We developed a script that sends tap events through Android debug bridge (adb) to click on the Teleport button, wait for 60s, click on "close", click on "home" button and then repeat the same process. The script also captures all the video and audio traffic using tcpdump. We simultaneously executed an SSL proxy script that dumped for each broadcast viewed a description and playback statistics, such as delay and stall events, which the application reports to a server at the end of a viewing session. These statistics are mainly useful for the RTMP streaming sessions since the app only reports the number of stall events for the HLS sessions.

We used two different Android phones: Samsung Galaxy S3 and S4. The phones were located at Aalto University and reverse tethered to the Internet through a USB connection to a Linux desktop machine providing them with over 100Mbps of available bandwidth both up and down stream. We imposed artificial bandwidth limits with the `tc` command on the Linux host. Mainly for the sake of latency measurements, NTP was enabled on the desktop machine and used the same server pool as the Periscope app. We recorded in total 4615 1-minute viewing sessions: 1796 RTMP and 1586 HLS sessions without a bandwidth limit and 1233 sessions with different bandwidth limits. In order to understand whether we should treat the data from the two different phones separately, we performed a series of Welch's t-tests to check whether the data sets differ statistically significantly. The results indicate that besides video frame rate, there are no statistically significant differences between the two datasets and we used data from both devices together in the analysis that follows.

In addition to the one-minute long viewing sessions, we recorded longer sessions with the S4 phone in which the viewing stops only when broadcast ends or when 1h has passed. No bandwidth limit was enforced. There are 1597 of these sessions and their average duration is 18 minutes. We use these only for the analysis in Section 4.4.

4.2 Protocol Selection and Client to Server Mapping

Since the choice of protocols between RTMP and HLS has an impact on the resulting latency and possibly also on the amount of rebuffering experienced during playback, we first try to understand the logic behind this choice as well as the client to server mapping.



(a) no bandwidth limiting

(b) bandwidth limiting

Fig. 8. Analysis of the stall ratio for RTMP streams with and without bandwidth limiting.

Selection of viewer client's protocol appears to be such that HLS is used only when a broadcast is very popular. Looking at the average number of viewers of RTMP and HLS stream suggests that HLS gets employed once the number of viewers grows beyond 100 or so, which has also been observed in [36]. By examining the IP addresses from which the video was received, we noticed that 87 different Amazon servers were employed to deliver the RTMP streams. We could locate only nine of them using `maxmind.com`, but among those nine there were at least one in each continent, except for Africa, which indicates that the server is chosen based on the location of the broadcaster. In contrast, HLS streams were delivered from only two distinct IP addresses. Their locations were somewhere in Europe and in San Francisco. It appears that the RTMP stream gets repackaged and delivered to Fastly CDN by Periscope servers, possibly the RTMP servers to which the video is transmitted by the broadcaster in the first place. Our single vantage point for measurements explains the difference in server locations observed between the protocols. As confirmed by analysis in [36], the RTMP server nearest to the broadcasting device is chosen when the broadcast is initialized, while the Fastly CDN server is chosen based on the location of the viewing device.

4.3 Playback Stalling

To analyze playback stalling, we leverage the statistics that the app reports for RTMP streams after playback, which include the number of stall events and the average stall time of an event. For HLS streams, the app only reports the number of stall events. We calculate the *stall ratio* for the RTMP streams by summing up stall time and dividing it by the total stream duration that includes all the stall and playback time. We plot a CDF of the results in Figure 8(a). The playback of most streams does not stall but a notable number of streaming sessions have a stall ratio of 0.05-0.09, which in most cases corresponds to a single stall event that lasts for 3-5 seconds.

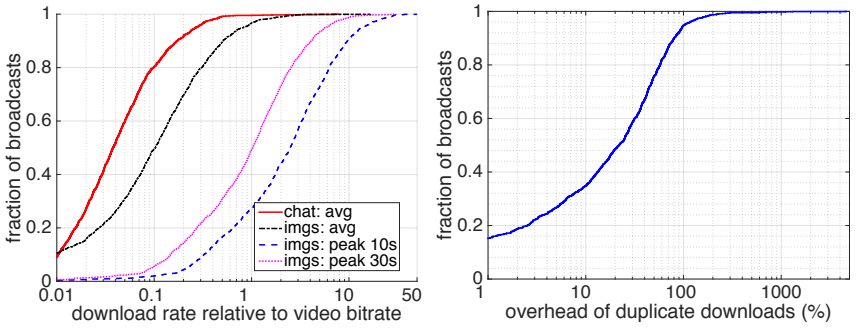
To understand how an artificial rate limit changes the results, we compute the stall ratio separately for experiments with different rate limits (100 means unlimited rate). The

boxplots in Figure 8(b) demonstrate that there is a relatively small chance of experiencing a stall event when viewing broadcasts with more than 2 Mbps of access network bandwidth, which suggest that a vast majority of the broadcasts are streamed with a bitrate inferior to 2 Mbps. As for the broadcasts streamed using HLS, comparing their stall count to that of the RTMP streams indicates that stalling is rarer with HLS than with RTMP, which may be caused by a larger buffer size (one chunk duration minimum) used by the application with HLS streams.

The average video bitrate is usually between 200 and 400 kbps (see Section 5), which is much less than 2 Mbps. Hence, there is an extra source of traffic that increases the total bandwidth demand so that stall events become more commonplace when the viewing device has less than 2 Mbps of available access network bandwidth. With a closer look at the traffic, we discovered that the chat feature is the most likely culprit, as enabling it clearly increases the traffic volumes.

4.4 The Chat Effect

The JSON encoded chat messages are not causing the increase in bandwidth demand, as they are received even when chat is turned off, but when the chat is on, continuous download of images from Amazon servers emerges in the traffic. Figure 2(a) reveals the reason: The app downloads profile pictures of chatting users and displays them next to their messages. The pictures are small when shown on display but some are large when downloaded and apparently downscaled afterwards on the mobile device. There are two kinds of profile pictures used: Periscope and Twitter profile pictures and the latter kind appear to be already downscaled at the server while the former are not.



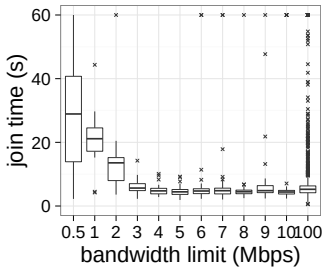
(a) Chat and image download rates relative to video bitrate (1 means same as video bitrate).

(b) Duplicate image downloads aggravate the effect.

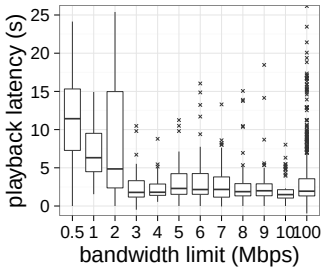
Fig. 9. Chat effect.

The profile picture downloading effect can be sizable. We happened to capture a case where turning the chat on in the middle of viewing a broadcast caused an increase of the aggregate data rate from roughly 500kbps to 3.5Mbps. The precise impact on bandwidth demand depends on the number of chatting users, their messaging rate, the fraction of them having a profile picture, and the format and resolution of profile pictures.

We analyzed the long viewing sessions in order to quantify the chat effect. We were able to isolate the image downloads by filtering the recorded traffic with server name and content



(a) join time



(b) playback latency

Fig. 10. The boxplots show that playback latency and join time of RTMP streams increases when bandwidth is limited. Notice the difference in scales.

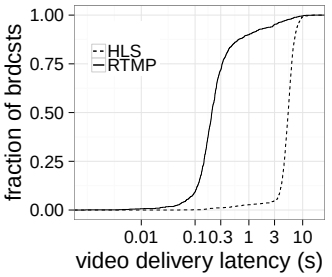


Fig. 11. HLS video delivery latency is much longer than that of RTMP streams.

type. We plot the CDF of download rates of the profile pictures and chat relative to the video bitrate in Figure 9(a). The increase in traffic is only 10% for half of the broadcasts when computed over the whole duration of viewed broadcasts. However, when we look at the peak rates computed over window of 10 or 30 seconds, the results look more severe: with half of the broadcasts, the 30s peak rate matches the video bitrate, hence effectively doubling the bandwidth demand, and with 10s peak rate, the demand is tripled for more than half of the broadcasts. This is problematic because the playback buffer size for live streaming applications has to be relatively small so that latency does not grow too much, which means that already a short-lived but severe competition for bandwidth by the image downloading may cause the playback buffer to run dry.

We also noticed that some pictures were downloaded multiple times, which indicates that the app does not cache them. We identified the redundant downloads by finding downloads with the same HTTP entity tag (ETag) value, if it was enabled, or alternatively the same MD5 digest. Figure 9(b) plots a CDF of the overhead caused by redundant downloads. Half the broadcasts generate 20-30% of extra traffic because of this and for some the redundant downloads multiply the download traffic by tens of times.

Fortunately, these chat related issues are easy to resolve by downscaling the profile images already at server and by caching the images at the app.

4.5 Latency

The duration of each viewing session was exactly 60s from the moment the Teleport button was pushed until terminating the viewing. We calculate the *join time*, also sometimes called startup latency, by subtracting the sum of playback and stall time from 60s. The boxplots in Figure 10(a) display the results for RTMP streams. In addition, we plot in Figure 10(b) the playback latency, as reported by the app, likely equivalent to the end-to-end latency. The y-axis scale was cut leaving out some outliers. Both increase when bandwidth is limited. In particular, join time grows dramatically when bandwidth drops to 2Mbps and below, partly because the app downloads the map of world to show where the broadcast about to be watched is located when using the Teleport feature. The average playback latency was roughly a few seconds when the bandwidth was not limited.

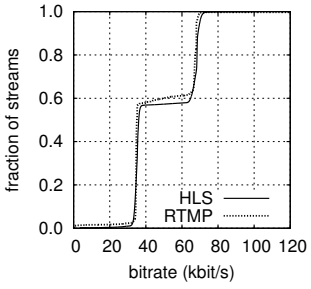


Fig. 12. Characteristics of the captured audio tracks.

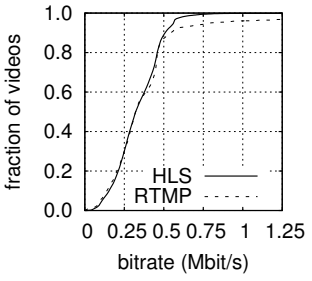


Fig. 13. Characteristics of the captured video tracks.

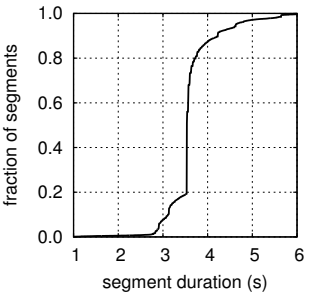


Fig. 14. Average duration of the HLS segment.

We performed some experiments where we controlled both the broadcasting and receiving client and captured both devices' traffic. We noticed that the broadcasting client application embeds NTP timestamps into the video data and that that these timestamps match very closely the tcpdump timestamps in a trace of the broadcasting device's traffic captured by the tethering machine. Hence, we could calculate the *delivery latency* by subtracting the NTP timestamp value from the time that the viewing client receives the packet containing it for both types of streams. We calculate the average over all the latency samples for each broadcast and plot the CDF of those average values in Figure 11. Only sessions that were not bandwidth limited are included. The results clearly demonstrate the latency difference between the two protocols. RTMP stream delivery is very fast happening in less than 300ms for 75% of broadcasts on average, which means that the majority of the few seconds of playback latency with those streams comes from buffering. In contrast, the delivery latency with HLS streams is over 5s on average. The biggest contributor to the HLS latency is that video is packaged into over 3 seconds long segments, which immediately adds 3s to the latency, whereas RTMP delivers the video frame by frame. For a more detailed analysis of the latency in Periscope, we refer the reader to [36].

5 AUDIO AND VIDEO QUALITY

5.1 Data Processing

For detailed analysis of audio and video quality, we use the pcap traffic from the Periscope dataset that we used for playback smoothness and latency analysis (refer to Section 4.1). We first reconstructed the video data of each session from the packet traces after which we analyzed it using a variety of scripts and tools.

After finding and reconstructing the multimedia TCP stream using wireshark [38], single segments are isolated by saving the response of HTTP GET request which contains an MPEG-TS file [12] ready to be played. For RTMP, we exploit the wireshark dissector which can extract the audio and video segments. We used the libav [19] tools to inspect the multimedia content and decode the video in full.

We also focused on estimating the quality of the video by making use of the technique described in [2], which provides an estimate of the Peak Signal-to-Noise Ratio (PSNR) for the decoded AVC video signal. Such a value can be used as an indication of the quality of the encoded video. More details are given in Section 5.3.

5.2 Audio and Video Coding

Both RTMP and HLS communications employ standard codecs for audio and video, that is, AAC (Advanced Audio Coding) for audio [14] and AVC (Advanced Video Coding) for video [13]. In more details, audio is sampled at 44,100 Hz, 16 bit, encoded in Variable Bit Rate (VBR) mode at about either 32 or 64 kbps, as shown in Figure 12, which seems enough to transmit almost any type of audio content (e.g., voice, music, etc.) with the quality expected from capturing through a mobile device.

Video resolution is always 320×568 (or vice versa depending on orientation). The video frame rate is variable, up to 30 fps. Occasionally, some frames are missing hence concealment must be applied to the decoded video. This is probably due to the fact that the uploading device had some issues, e.g., glitches in the real-time encoding or during upload.

Fig. 13 shows the video bitrate, typically ranging between 200 and 400 kbps. Moreover, there is almost no difference between HLS and RTMP except for the maximum bitrate which is higher for RTMP. Analysis of such cases reveals that poor efficiency coding schemes have been used (e.g., I-type frames only). In fact, in real applications rate control algorithms try to keep its average close to a given target, but this is often challenging as changes in the video content directly influences how difficult is to achieve such bitrate. To this aim, the so called quantization parameter (QP) is dynamically adjusted [3]. More details about quality will be given in the next section.

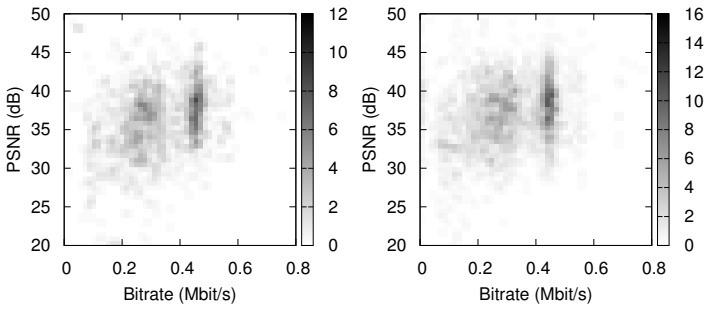
An interesting parameter for the HLS case is the segment duration. Figure 14 shows the observed segment duration. The large majority of cases present segment lengths equal to 3.6 s. Such value corresponds to 108 frames at 30 fps. Some last less, which is what happens when the duration of the video is not multiple of 3.6, but some last more than 3.6, indicating that there could be some flexibility in deciding the segment length parameter.

Finally, we investigated the frame type pattern used for encoding. Most use a repeated IBP scheme. Few encodings (20.0 % for RTMP and 18.4% for HLS) only employ I and P frames only (or just I in 2 cases). After about 36 frames, a new I frame is inserted. Although one B frame inserts a delay equal to the duration of the frame itself, in this case we speculate that the reason they are not present in some streams could be that some old hardware might not support them for encoding.

5.3 No-Reference Quality Assessment

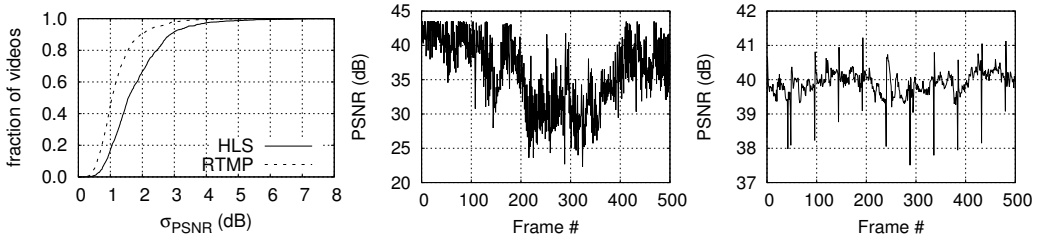
To get a deeper insight in the quality of the video provided to the users, we implemented the video quality estimation technique described in [2]. Although it is always difficult to estimate the quality of video content without having the original uncompressed reference, as in our case, such a technique relies on a statistical analysis of the quantized coefficients in the decoded video blocks to determine the parameters of a probability distribution function that models the original, not quantized, coefficients. Then, the difference between the energy of the not quantized and quantized coefficients is estimated, determining a Mean Squared Error (MSE) which is then converted in the more usual PSNR measure at the frame level. PSNR is then averaged over the whole video segment to provide an indication of the quality of the video compression process. Differently from a generic technique based, e.g., on simply analyzing the quantization parameter as done in our previous work [29], such a technique also considers the video content, allowing for more definitive conclusions about video quality and its evolution over time.

Figure 15 shows the PSNR value as a function of the video bitrate for the case of the RTMP and HLS protocols. The graphs are presented in the form of heatmaps instead of showing



(a) RTMP (b) HLS

Fig. 15. PSNR as a function of the video bitrate for each session.



(a) CDF (b) Bad (c) Good

Fig. 16. PSNR behavior over time: CDF for all sessions and two sample cases (first 500 frames)

the single points that correspond to each single session to avoid point superimposition that would hamper readability. Also, note that for the HLS protocol, all segments belonging to a session are considered together and represented by only one pair of bitrate and PSNR value. In fact, the user is typically not aware of the fact that the content is served in a chunked fashion, as it is played back continuously. From the graphs it can be surmised that there is no significant variation between the two protocols. Both show that about 450 kbit/s is a popular average bitrate for the sessions. Within that range, however, variations can be significant, ranging from about 25 to 50 dB PSNR.

We also investigated the behavior over time of the quality through the PSNR value of each single video frame. To this aim, we computed the standard deviation σ_{PSNR} of the PSNR value over time for each session. Figure 16(a) shows the cumulative distribution function (CDF) of the σ_{PSNR} value for the RTMP and HLS sessions. It is possible to notice that the two types of sessions exhibit a significantly different behavior. Large variations (higher than 2 dB PSNR) are present in only 10% of the cases for RTMP but in more than 30% for HLS.

An example of the behavior of the PSNR for two extreme cases (σ_{PSNR}=0.41 dB and 4.66 dB respectively) as a function of time is shown in Figure 16(b) and 16(c). In the former it is clear that there are significant quality variations over time. Visual inspection of some sequences that exhibit a behavior similar to Figure 16(b) confirms that the quality variations are significant and very noticeable. In Figure 17 we reported two sample pictures extracted from the high-σ_{PSNR} video for a "good" and "bad" frame that confirm such observation. The

quality is, instead, much more stable for the video represented in Figure 16(c). The visual analysis manually performed on some high-σ_{PSNR} videos indicates that such short-term quality variations can be very often attributed to the extreme time variability of the captured content. For instance, shots are very unstable and sudden movements are very common since devices are often held in hand by people doing other tasks, such as walking or moving in general.

Regarding the video content, it is also interesting to note that, for the case of HLS sessions, in each segment both the bitrate and the quality can vary significantly, as shown in Figure 18. This is in agreement with Figure 16(a) where higher PSNR variability over the time span of the session could be observed. Also, note the area of the graph at very low bitrate but very high quality: that is typically a series or black or very dark frames, which are often encountered in Periscope videos because the capture device might be moved, turned or positioned in a way where the camera view is temporarily obstructed. Regarding the larger variability of the segment bitrates, this could be caused by the fact that each segment is probably processed and encoded independently. 95.4% of the segments are, in fact, do not exhibit coding dependencies on the next segment in the same session, i.e., they end with a P or I-type frame.

6 VIDEO BITRATE ADAPTATION

So far we have seen statistics about playback stalling, latency, and video quality but we also wanted to understand in more detail where these statistics stem from. In particular, we wish to know how the video bitrate is determined, which in turn affects also the playback smoothness. It is difficult to conclude anything about the broadcaster's connectivity and its impact on the video bitrate using the data we have collected and studied so far. Hence, we performed a limited set of controlled experiments with both Periscope and FB Live specifically to reveal how the video encoding bitrate gets chosen at the broadcaster and whether it is varied and according to which logic, i.e. what kind of video bitrate adaptation is used. We mainly focus on the broadcaster side adaptation but we also briefly study the viewer side.

(a) #36 (b) #142

Fig. 17. Sample frames from the "bad" sequence in Fig. 16(b), showing high and low PSNR values.

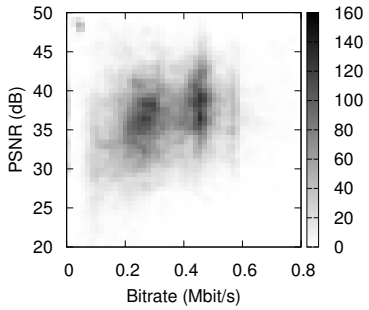
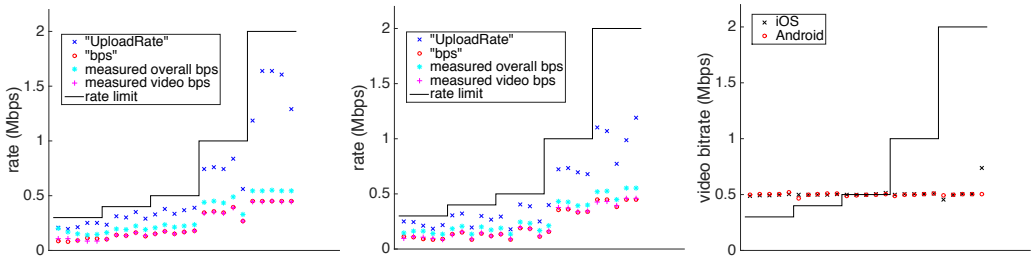


Fig. 18. PSNR as a function of the video bitrate for each single segment in the HLS sessions.



(a) Periscope on Android (b) Periscope on iOS (c) FB Live

Fig. 19. Bitrates with constant upload rate limit.

6.1 Experiment Setup

We used two newer smartphones in these experiments, an iPhone SE (iOS 10.3.2) and Samsung Galaxy S7 (Android 7.0). We used the latest Pericope and Facebook app versions available at the time of writing this paper. We used Mac OS Internet sharing with Wi-Fi to connect the phone. The tethering device was a Macbook Air which had a 100 Mbps wired connection to the Internet. We used the Network Link Conditioner to enforce symmetric rate limits upstream and downstream.

For experiments with Periscope, we varied the rate limit and streamed live video for approximately one minute. The video being shot was the beginning of the movie called Big Buck Bunny playing on a laptop screen. Afterwards, we replayed the video using a Web browser on a desktop machine with again 100 Mbps Internet connection and recorded the traffic using tshark. During replay, we also logged SSL key exchange information by setting the SSLKEYLOGFILE environment variable, which enabled us to decrypt the traffic and reconstruct the video data in the way we explain in Section 5.1.

With FB Live, we did live capturing of the broadcast instead of replaying. Our setup was otherwise similar to the setup with Periscope except that a desktop machine was viewing the stream live. We developed new scripts to reconstruct FB Live stream video and audio from packet traces because a Chrome browser receives FB Live streams as fragmented MP4 (i.e., uses DASH) over HTTP/2 instead of MPEG-TS over HTTP/1.1 used by Periscope.

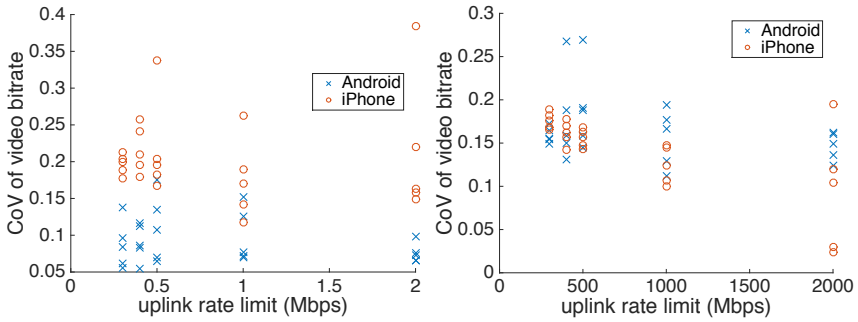
6.2 Broadcasting with Constant Available Bandwidth

We first take a look at the bitrate selection of the applications when there is constant amount of bandwidth available. Figure 19 shows the results we measured. The Android version of the Periscope application seems to choose the video bitrate rather conservatively, as we can see from Figure 19(a). The two attribute values `UploadRate` and `bps` are embedded into the video stream and the former seems to be the application's estimate of the available amount of bandwidth and the bps reflects the chosen target video bitrate, since the measured video bitrate matches very well this attribute value. If this interpretation is accurate, the app seems to systematically underestimate the amount of available bandwidth in this experiment setup. The ratio of the chosen bitrate to the bandwidth estimate is consistently in between 0.4 and 0.5 except for the experiments with 2 Mbps available bandwidth, which suggests that the video bitrate of slightly below 0.5 Mbps appears to be the maximum used by the application. Results with iPhone plotted in Figure 19(b) indicate that the behavior is

qualitatively similar to the Android app although the iPhone app appeared to underestimate the bandwidth somewhat more than the Android app.

In addition to the systematic underestimation, both Periscope versions occasionally underestimate the bandwidth with a large margin, which may be caused by other activity that happens in parallel to the bandwidth estimation (no other apps were running). For instance, we noticed that if we tried to broadcast right after launching the iPhone app in an experiment where the bandwidth was limited below 0.5 Mbps, the app would systematically report poor connectivity and refuse to begin broadcasting. This behavior is probably caused by downloading recent broadcast information including their thumbnails shown on the front page by the app upon startup, which is still ongoing while the bandwidth is being estimated.

To our surprise, the FB Live application does not seem to adjust the video bit rate according to any bandwidth estimates. Instead, in our experiments, it always streamed video with a target bitrate around 0.5 Mbps, except for some outliers, as is clearly visible in Figure 19(c). While we viewed and recorded the broadcasts live, we observed playback stalling when the there was insufficient amount of bandwidth available, i.e. limit was below 0.5 Mbps. Sometimes we noticed a "spinning circle" and sometimes a large text announcing "Live Video Interrupted."



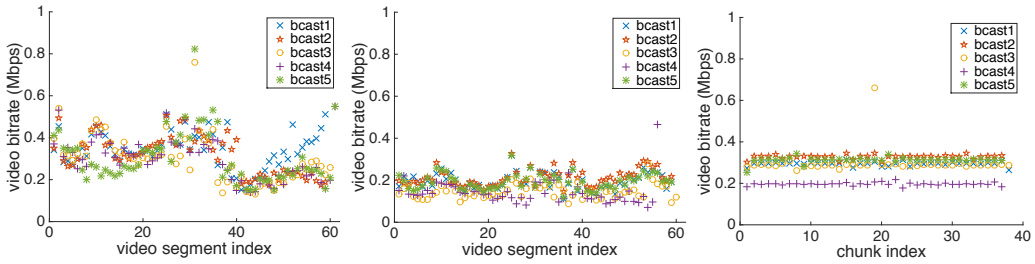
(a) Periscope (b) FB Live

Fig. 20. Coefficient of variation of video bitrate with constant upload rate limit.

We also computed the coefficient of variation of the video bitrate for both version of the two applications and the different bandwidth limits (Figure 20). The most interesting thing to note from Figure 20(a) is the difference between the Android and the iOS versions, which suggests that the two versions may use a different encoding rate control strategy, such as CBR for Android vs. constant QP for iOS. We will observe this difference also in Figures 21(b) and 21(c).

6.3 Broadcasting with Changing Available Bandwidth

We next conducted experiments where the amount of available bandwidth changes during the broadcast. We only used Periscope because we could not find any evidence of bitrate adaptation with FB Live. Figure 21 shows bitrates per video segment of Periscope broadcasts of the two first minutes of the same Big Buck Bunny video. After the first minute, the bandwidth limit was changed from 0.5 to 1 Mbps or vice versa. We notice that the iOS version of the application tries to adapt to the available bandwidth, whereas the Android



(a) iOS, from 1 to 0.5 Mbps (b) iOS, from 0.5 to 1 Mbps (c) Android, from 1 to 0.5 Mbps

Fig. 21. Video segment bitrates of Periscope broadcasts when upload rate limit changes in the middle of the broadcast.

version does not. The iOS version only appears to react to a decrease in the amount of available bandwidth but not noticeably to an increase of it. The bitrate is adjusted with some delay to a decrease in the available bandwidth and in some cases the bitrate increases afterwards again (bcast1).

With Android, the bitrate is essentially a flat line for all experiments we tried. The experiment of which the results are plotted in Figure 21(c) played without problems because the video bitrate chosen with 1 Mbps bandwidth limit was below the lower 0.5 Mbps limit as well. However, when we reduced the bandwidth more dramatically to 0.3 Mbps after the first minute of broadcasting, the bitrate remained the same but the delay started to grow and not all the video was ever uploaded to the server. To a live viewer, such behavior manifests as periodic playback stalling.

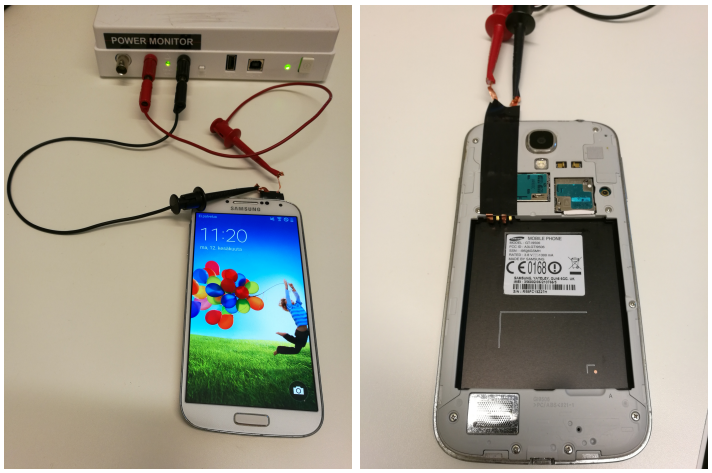
These results demonstrate that there are opportunities to improve the QoE in terms of video quality playback smoothness by incorporating smarter bitrate adaptation logic into these applications.

6.4 Viewer Side Bitrate Adaptation

On the viewer side, both applications use adaptive streaming protocols (HLS and DASH) except for the low-latency RTMP streams of Periscope. However, we could not find multiple representations of the video delivered using HLS with Periscope but we cannot say anything conclusive about it. With FB Live, we observed roughly half of the times two different video representations included in the MPD (bandwidth: 500000/250000, FBQualityLabel: 360p/240p) and half of the time only the higher quality one. We did not perform any experiments where the viewer bandwidth is choked below that of the broadcaster's upstream bandwidth. Such a situation corresponds to a traditional case of bitrate adaptation when streaming to a mobile device from a server, and the behavior of these adaptive streaming protocols in such a situation is already fairly well understood.

7 POWER CONSUMPTION

Energy efficiency continues to be a major concern with smartphones. While they pack an incredible amount of computing, communication, and sensing technology and power today, the battery capacities have grown at a modest rate. Power consumption of a mobile application or service is first and foremost a Quality of Experience factor as it directly



(a) Smartphone connected to Monsoon

(b) Connection to phone's battery terminals using copper and electrical tape

Fig. 22. Power measurement setup.

affects the battery life of the device and, hence, recharging frequency. As our final set of measurements, we studied the power consumption of Periscope and FB Live applications.

7.1 Measurement Setup

We connected a Samsung Galaxy S4 4G+ smartphone to a Monsoon Power Monitor [21] in order to measure its power consumption as instructed in [35]. The mobile phone was connected to the power monitor by bypassing the battery of the device. The voltage terminal of the battery was covered with insulating tape and copper foil tape was used to allow connecting the Monsoon Power Monitor device to the phone. The connection setup is presented in Figure 22. We used the PowerTool software to record the data measured by the power monitor and to export it for further analysis. This kind of setup is considered as the gold standard in smartphone power measurements [35].

The screen brightness was full in all test cases and the sound was off. The phone was connected to the Internet through non-commercial WiFi and LTE networks[3].

7.2 Results

Figure 23 shows the results for Periscope app. We measured the idle power draw in the Android application menu to be around 900 to 1000 mW both with WiFi and LTE connections. With the Periscope app on without video playback, the power draw grows already to 1537 mW with WiFi and to 2102 mW with LTE because the application refreshes the available videos every 5 seconds.

Playing back old recorded videos with the application consume an equal amount of power as playing back live videos. The power consumption difference of RTMP vs HLS is also very small. Interestingly, enabling the chat feature of the Periscope videos raises the power

[3]It is a full-fledged LTE network operated by Nokia. DRX was enabled with typical timer configuration.

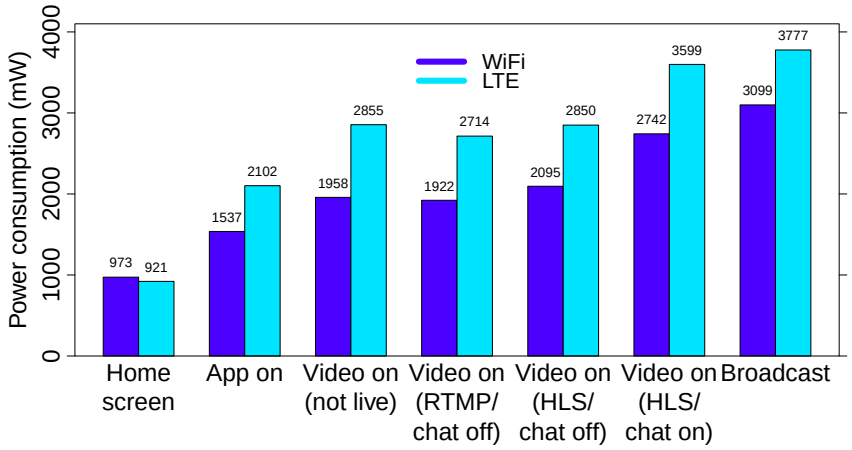


Fig. 23. Average power consumption with Periscope and idle device.

consumption to 2742 mW with WiFi and up to 3599 mW with LTE. This is only slightly less than when broadcasting from the application. However, the test broadcasts had no chat displayed on the screen.

We further investigated the impact of the chat feature by monitoring CPU and GPU activity and network traffic. Both processors use DVFS to scale power draw to dynamic workload [35]. We noticed an increase by roughly one third in the average CPU and GPU clock rates when the chat is enabled, which implies higher power draw by both processors. As we discovered in Section 4.4, the chat feature may increase the amount of traffic, especially with streams having an active chat, which inevitably increases the energy consumed by wireless communication. The energy overhead of chat could be mitigated by caching profile pictures, downscaling them at the server, and allowing users to disable their display in the chat.

The results of FB Live power measurements are presented in Figure 24. Recall that the Facebook application logic is different from Periscope in that it does not display a list of ongoing broadcasts, only gives notifications of your friends or those that you follow. Hence, it also does not download periodic updates like the Periscope application does, which results in significantly lower power consumption without video playback. In video playback, we did not observe large variations between playing a recorded video or a live stream. Similarily to Periscope, broadcasting a video consumes a lot of energy compared to the other scenarios. The chat feature affects the power consumption much less than with Periscope even though both apps show chat messages in the same way including the profile pictures. There were also no noticeable differences between viewing a highly popular broadcast or one with only a few viewers.

8 RELATED WORK

Many papers have been published on transmission of on-demand video over wireless networks, too many to be cited here. Live streaming over wireless networks has also received a lot of attention. Within that body of work, live video broadcasting or multicasting over wireless networks is most related to this work. For example, SVC has been studied as a suitable way to encode the video for broadcasting or multicasting in [10, 11, 15, 30, 39], to name a

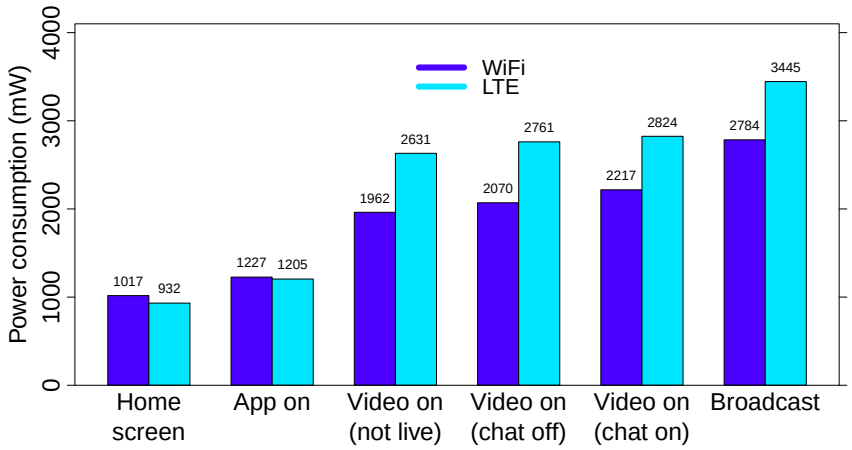


Fig. 24. Average power consumption with Facebook Live and idle device.

few, but only [30] considers mobile device as the source of video. In contrast to using SVC, transcoding based approach has also been rather thoroughly examined [9, 25, 41]. Some research has considered direct device-to-device communication [42].

QoE aspects of on-demand video streaming have been studied extensively. For example, Shafiq et al. present results from a large scale measurement study of mobile video usage [27]. Krishnan et al. [16] study the effect of initial joining time and buffering events on the engagement in watching videos. Dobrian et al. [5] provide some insights into mapping the considered QoE metrics to user behavior through a utility function and present a predictive model of Internet video QoE in their follow-up work [1]. A survey on QoE with adaptive video streaming is presented by Seufert et al. [26].

As for smartphone power consumption, Tarkoma et al. provide a comprehensive survey on the measurement techniques as well as approaches for power modeling and optimization[35]. Numerous papers have studied the effect of on-demand streaming to mobile device power consumption. In comparison, relatively few have studied smartphone power consumption while live streaming video [24, 28, 29, 33].

Live mobile video broadcasting has been studied from the human computer interaction, particularly engagement, point of view [8, 32, 34, 37]. Even sociological aspects have been investigated [31]. Yet, only a couple of technical measurement studies concerning existing mobile live broadcasting applications and services exist so far. Most of the research has focused on systems where the mobile device is only the receiver of the live streaming, like Twitch.Tv [4, 23, 40] (although Twitch appears to be expanding towards mobile live video broadcasting too), or other mobile VoD systems [18]. The recent work from Wang et al. [36] and us [6, 29] are the first to present measurement studies of the anatomy and performance of a popular mobile live video broadcasting applications. Compared to these papers, while we build on our preliminary work, we also have dived deeper into the video quality aspects by adding no-reference video quality assessment and experiments on bitrate adaptation, investigated the Periscope chat effect in more detail, and included FB Live in those parts of the study where possible.

9 CONCLUSIONS

In this work we investigated different aspects of the quality of experience offered by two popular live mobile streaming services, i.e. Periscope and FB Live. Our insights have been derived from statistical analyses of data obtaining through crawling as well as experiments using a few different mobile devices, using both the Android and iOS versions. Our study focused mainly on aspects such as user behavior, playback smoothness and latency, quality of the media content, bitrate adaptation and issues related to power consumption on mobile devices. Some of the major results include: the relatively high difference in latency due to the use of different protocols (RTMP and HLS) to serve the content to the final users, which seems to be related to the number of users; the surprising surges in bandwidth demand for the chat feature of the Periscope app; the significant quality of experience variations over time for the users watching the video content; the generally poor adaptation strategies to the available upstream bandwidth; the significant power consumption caused by the applications. We hope that this research will help to shed more light in the complex field of live mobile streaming that requires significant coordination between different domains of expertise (e.g., networking, multimedia, hardware) to achieve the best quality of experience.

ACKNOWLEDGMENTS

The work is supported by the Academy of Finland under Grant No.: 278207 and 297892 and Tekes - the Finnish Funding Agency for Innovation.

REFERENCES

[1] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. 2013. Developing a Predictive Model of Quality of Experience for Internet Video. In *Proc. of the ACM SIGCOMM Conference*. Hong Kong, China, 339–350.

[2] T. Brandao and M. P. Queluz. 2008. No-reference PSNR estimation algorithm for H.264 encoded video sequences. In *16th European Signal Processing Conference (EUSIPCO)*. IEEE, 1–5.

[3] Zhenzhong Chen and King Ngi Ngan. 2007. Recent advances in rate control for video coding. *Signal Processing: Image Communication* 22, 1 (2007), 19–38.

[4] Jie Deng, Gareth Tyson, Felix Cuadrado, and Steve Uhlig. 2017. Internet scale user-generated live video streaming: The Twitch case. In *International Conference on Passive and Active Network Measurement*. Springer, 60–71.

[5] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. 2011. Understanding the Impact of Video Quality on User Engagement. In *Proc. of the ACM SIGCOMM Conference*. Toronto, ON, Canada, 362–373.

[6] L. Favario, M. Siekkinen, and E. Masala. 2016. Mobile live streaming: Insights from the periscope service. In *2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP)*. 1–6.

[7] Genymotion 2017. Genymotion. (2017). https://www.genymotion.com/.

[8] Oliver L. Haimson and John C. Tang. 2017. What Makes Live Events Engaging on Facebook Live, Periscope, and Snapchat. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 48–60.

[9] Q. He, J. Liu, C. Wang, and B. Li. 2016. Coping With Heterogeneous Video Contributors and Viewers in Crowdsourced Live Streaming: A Cloud-Based Approach. *IEEE Transactions on Multimedia* 18, 5 (May 2016), 916–928.

[10] Cheng-Hsin Hsu and Mohamed Hefeeda. 2010. Achieving Viewing Time Scalability in Mobile Video Streaming Using Scalable Video Coding. In *Proc. of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys)*. ACM, Scottsdale, AZ, 111–122.

[11] Cheng-Hsin Hsu and M. Hefeeda. 2011. Flexible Broadcasting of Scalable Video Streams to Heterogeneous Mobile Devices. *IEEE Transactions on Mobile Computing* 10, 3 (March 2011), 406–418.

[12] ISO/IEC 13818-1. 2007. MPEG-2 Part 1 - Systems. (Oct. 2007).

[13] ISO/IEC 14496-10 & ITU-T H.264. 2003. Advanced Video Coding (AVC). (May 2003).

[14] ISO/IEC 14496-3. 2005. MPEG-4 Part 3 - Audio. (Dec. 2005).

[15] Kyungtae Kang, Yongwoo Cho, Jinsung Cho, and Heonshik Shin. 2007. Scheduling Scalable Multimedia Streams for 3G Cellular Broadcast and Multicast Services. *IEEE Transactions on Vehicular Technology* 56, 5 (Sept. 2007), 2655–2672.
[16] S. Shunmuga Krishnan and Ramesh K. Sitaraman. 2012. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs. In *Proc. of the 2012 ACM Conference on Internet Measurement Conference (IMC)*. Boston, MA, 211–224. https://doi.org/10.1145/2398776.2398799
[17] Federico Larumbe and Abhishek Mathur. 2015. Under the hood: Broadcasting live video to millions. https://code.facebook.com/posts/1653074404941839/under-the-hood-broadcasting-live-video-to-millions/. (Dec. 2015).
[18] Zhenyu Li, Jiali Lin, Marc-Ismael Akodjenou, Gaogang Xie, Mohamed Ali Kaafar, Yun Jin, and Gang Peng. 2012. Watching videos from everywhere: a study of the PPTV mobile VoD system. In *Proc. of the 2012 ACM conf. on Internet Measurement Conference*. ACM, 185–198.
[19] Libav 2017. LibAV Project. (2017). https://libav.org/.
[20] Mitmproxy 2017. Mitmproxy Project. (2017). https://mitmproxy.org/.
[21] msoon 2017. Monsoon. (2017). http://www.msoon.com.
[22] Periscope. 2016. Year One. https://medium.com/@periscope/year-one-81c4c625f5bc#.mzobrfpig. (March 2016).
[23] Karine Pires and Gwendal Simon. 2015. YouTube Live and Twitch: A Tour of User-generated Live Streaming Systems. In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys '15)*. ACM, New York, NY, USA, 225–230.
[24] S.V. Rajaraman, M. Siekkinen, and M.A. Hoque. 2014. Energy consumption anatomy of live video streaming from a smartphone. In *Personal, Indoor, and Mobile Radio Communication (PIMRC), IEEE 25th Annual International Symposium on*. 2013–2017.
[25] B. Seo, W. Cui, and R. Zimmermann. 2012. An experimental study of video uploading from mobile devices with HTTP streaming. In *Proceedings of the 3rd ACM Multimedia Systems Conference*. 215–225.
[26] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hobfeld, and Phuoc Tran-Gia. 2015. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 469–492.
[27] 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 *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. Austin, TX, 367–379.
[28] Yousef O. Sharrab and Nabil J. Sarhan. 2013. Aggregate Power Consumption Modeling of Live Video Streaming Systems. In *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys '13)*. ACM, New York, NY, USA, 60–71.
[29] 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 Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA, 477–483.
[30] Matti Siekkinen, Enrico Masala, and Jukka K Nurminen. 2017. Optimized upload strategies for live scalable video transmission from mobile devices. *IEEE Transactions on Mobile Computing* 16, 4 (2017), 1059–1072.
[31] Daxton Stewart and Jeremy Littau. 2016. Up, Periscope: Mobile Streaming Video Technologies, Privacy in Public, and the Right to Record. *Journalism and Mass Communication Quarterly, Special Issue: Information Access and Control in an Age of Big Data* (2016).
[32] D. Stohr, T. Li, S. Wilk, S. Santini, and W. Effelsberg. 2015. An analysis of the YouNow live streaming platform. In *Local Computer Networks Conference Workshops (LCN Workshops), 2015 IEEE 40th*. 673–679.
[33] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. 2016. A High-precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, New York, NY, USA, Article 14, 12 pages.
[34] John C. Tang, Gina Venolia, and Kori M. Inkpen. 2016. Meerkat and Periscope: I Stream, You Stream, Apps Stream for Live Streams. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4770–4780.
[35] Sasu Tarkoma, Matti Siekkinen, Eemil Lagerspetz, and Yu Xiao. 2014. *Smartphone Energy Consumption: Modeling and Optimization*. Cambridge University Press.

[36] Bolun Wang, Xinyi Zhang, Gang Wang, Haitao Zheng, and Ben Y. Zhao. 2016. Anatomy of a Personalized Livestreaming System. In *Proc. of the 2016 ACM Conference on Internet Measurement Conference (IMC)*.

[37] Stefan Wilk, Dimitri Wulffert, and Wolfgang Effelsberg. 2015. On Influencing Mobile Live Video Broadcasting Users. In *2015 IEEE International Symposium on Multimedia (ISM)*. IEEE, 403–406.

[38] Wireshark 2017. Wireshark Project. (2017). https://www.wireshark.org/.

[39] Dapeng Wu, Yiwei Thomas Hou, and Ya-Qin Zhang. 2001. Scalable video coding and transport over broadband wireless networks. *Proc. IEEE* 89, 1 (Jan. 2001), 6–20.

[40] Cong Zhang and Jiangchuan Liu. 2015. On Crowdsourced Interactive Live Streaming: A Twitch.Tv-based Measurement Study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '15)*. ACM, New York, NY, USA, 55–60.

[41] Y. Zheng, D. Wu, Y. Ke, C. Yang, M. Chen, and G. Zhang. 2016. Online Cloud Transcoding and Distribution for Crowdsourced Live Game Video Streaming. *IEEE Transactions on Circuits and Systems for Video Technology* PP, 99 (2016), 1–1.

[42] Liang Zhou. 2016. Mobile Device-to-Device Video Distribution: Theory and Application. *ACM Trans. Multimedia Comput. Commun. Appl.* 12, 3, Article 38 (March 2016), 23 pages.