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 quality, 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 (April 2017), 23 pages.
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

*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.

1551-6857/2017/4-ART1 $15.00
https://doi.org/0000001.0000001

Table 1. QoE factors under investigation and variables measured.

QoE factor	measured variables
playback & buffer mgmt	playback stall time (Sec 5.3), video bitrate adaptation (Sec 7)
delay (Sec 5.5)	startup latency, video delivery latency, playback latency
video/audio quality (Sec 6)	frame rate, bitrate, resolution, codec and frame types, no reference PSNR
battery life (Sec 8)	power consumption

one year birthday, Periscope announced in March 2016 that their users watch over 110 years of live video every day [21].

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 influence 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. Table 1 summarizes the different QoE influence factors investigated and the related variables measured (section indicates where the corresponding analysis is presented). We limit the scope of this work to only objective, indirect measurements of QoE and do not perform subjective experiments to quantify QoE. The selection of influence factors and variables to study has been guided by related work on QoE modeling (Section 2) as well as intuition (e.g., delay is particularly important with mobile live broadcasting services due to real time feedback from viewers).

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, the startup latency and video stalling start to increase already when the amount of available access network bandwidth is still clearly above 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 choose the 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 3, we explain mobile live video broadcasting and the two applications in detail. In Section 4, we analyze Periscope usage patterns. In Sections 5 and 6 we analyze the QoE factors, such as video stalling, latency, and video quality of Periscope in detail. In Section 7, we study the video bitrate adaptation logic of the two applications and in Section 8 we measure their power consumption. Finally, we compare our work to related work in Section 2 before concluding in Section 10.

2 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 [11, 12, 17, 29, 38], to name a few, but only [29] considers mobile device as the source of video. In contrast to using SVC, transcoding based approach has also been rather thoroughly examined [10, 24, 40]. Some research has considered direct device-to-device communication [41].

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 [26]. Krishnan et al. [18] study the effect of initial joining time and buffering events on the engagement in watching videos. Dobrian et al. [7] 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 [3]. A survey on QoE with adaptive video streaming is presented by Seufert et al. [25]. According to [7], the important variables that influence Internet video streaming engagement are video bitrate, playback stalling, and, to a smaller degree, startup delay. In addition to these, we include in our study more traditional video quality measures as well as latency given the nature of the services under study. As a word of caution, it is possible that the behavior of users of mobile live broadcasting services differs in some way from traditional mobile streaming. For example, perhaps users are more accustomed and tolerant of poor video quality. Hence, subjective experiments for QoE modeling with these services are certainly of interest but out of scope of this paper. However, we believe that the same factors are important although the magnitude of their impact may differ.

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[34]. 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 [23, 27, 28, 31].

Live mobile video broadcasting has been studied from the human computer interaction, particularly engagement, point of view [9, 30, 33, 37]. 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 [6, 22, 39] (although Twitch appears to be expanding towards mobile live video broadcasting too), or other mobile VoD systems [20]. The recent work from Wang et al. [35] and us [8, 28] 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.

3 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.

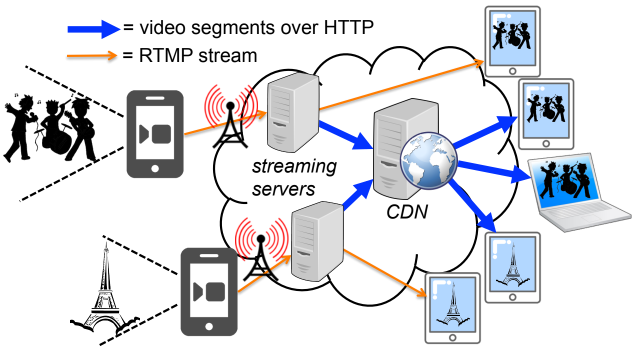


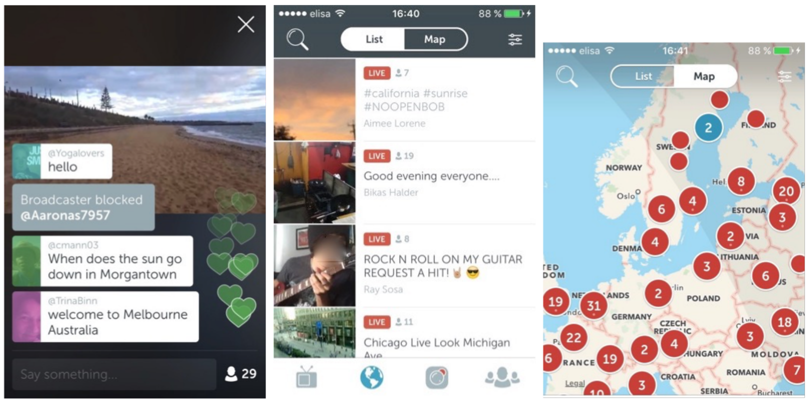
Fig. 1. Typical mobile live video broadcasting system.

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 4 to 6) but we also include FB Live in the investigations of video bitrate adaptation and power consumption (Section 7). We next briefly describe how the two chosen applications work.

3.1 Periscope

Periscope allows both public and private broadcasting. Only chosen users can view private streams. The application also supports using a GoPro instead of the in-built camera of the mobile device.


@Yogalovers
hello

Broadcaster blocked
@Aaronas7957

@cmann03
When does the sun go down in Morgantown

@TrinaBinn
welcome to Melbourne Australia

Say something...

LIVE 7
#california #sunrise #NOOPENBOB
Aimee Lorene

LIVE 19
Good evening everyone....
Bikas Halder

LIVE 8
ROCK N ROLL ON MY GUITAR REQUEST A HIT!
Ray Sosa

LIVE 11
Chicago Live Look Michigan Ave

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

Fig. 2. Periscope screenshots.

Viewers can send text messages and "floating hearts" as real-time feedback to the broadcaster, as 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.

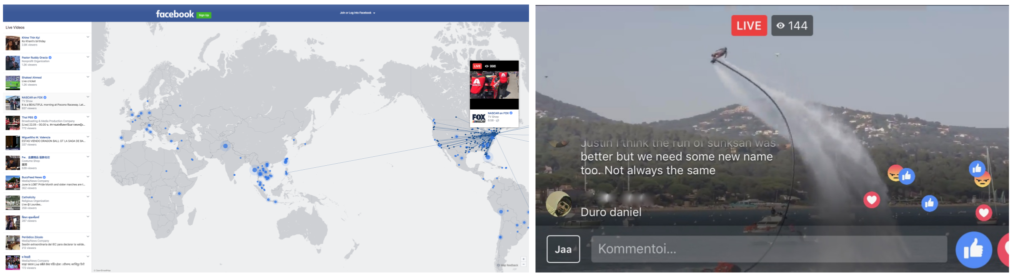
A user can discover public broadcasts in different ways. The application displays a list of ranked broadcasts, as shown in the screen capture in Figure 2(b), and a few featured ones. The user can also explore the map of the world in order to find a broadcast in a specific geographical region or perform 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. 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 5.5). The reason to use HLS may be related to scalability and/or costs [35].

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 5. Public streams 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.

3.2 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



Justin I think the run of sunksan was better but we need some new name too..Not always the same

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

Fig. 3. Periscope screenshots.

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[19]. 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.

4 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 [21], 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.

4.1 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 (http://mitmproxy.org) in between the mobile device and the Periscope service as a transparent proxy. The proxy intercepts the 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) 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

[1]At the time of writing, also the Android app has certificate pinning enabled.

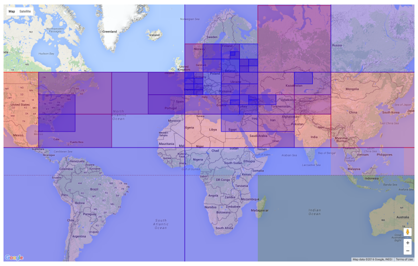


Fig. 4. Map of selected top zones. Colour indicates how many top zone lists of the different crawls the zone appears in (dark blue is one, dark red is all).

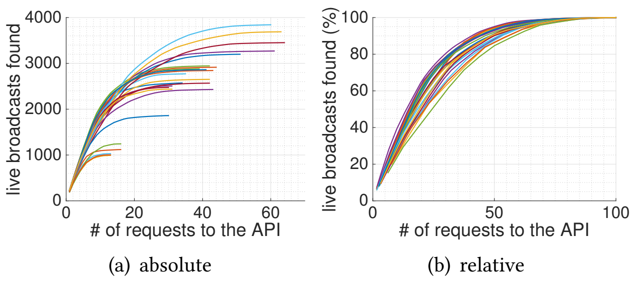


Fig. 5. Cumulative number of broadcasts discovered as a function of crawled areas (# of requests). Each curve corresponds to a different deep crawl.

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 broadcasts 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 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

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

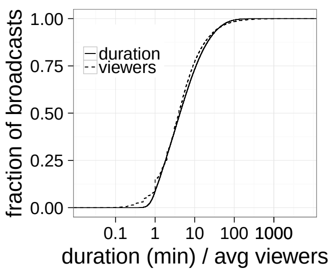


Fig. 6. Distribution of broadcast duration and number of viewers.

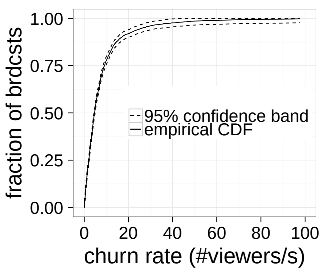


Fig. 7. Churn rate distribution of broadcasts.

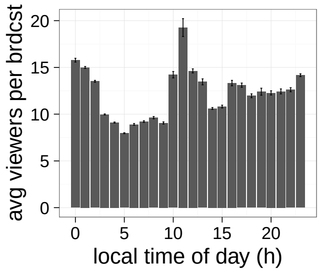


Fig. 8. Popularity per start time (error bars mark the 95% CI).

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.

4.2 Broadcast and Viewer Statistics

Figure 6 shows CDF plots of broadcast duration and average number of simultaneous viewers in the dataset (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.

Figure 7 plots the CDF of viewer churn rate per broadcast. It is computed from a smaller dataset in which we were able to track both the cumulative and current number of viewers (see Section 5.1 for data collection details). The 95% confidence band is computed using Kolmogorov-Smirnov statistics. For half of the broadcasts, churn rate is five users per second or smaller and the higher quartile goes from 10 to over 200 (x-axis of the figure is cropped), which is high given the relatively small values of average audience.

The local time of day shown in Figure 8 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 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.

5 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.

5.1 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 (null hypothesis could not be rejected with high confidence, p-value range: 0.04-0.7) 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 5.4.

5.2 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. 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 [35].

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 suggests that the server is chosen based on the location of the broadcaster, also observed by the authors of [35]. Some results of further investigation are plotted in Figure 9. Each line represents an observed mapping from broadcaster to RTMP server whose end points indicate their locations. The mapping of broadcaster is indeed usually to the closest server but there are

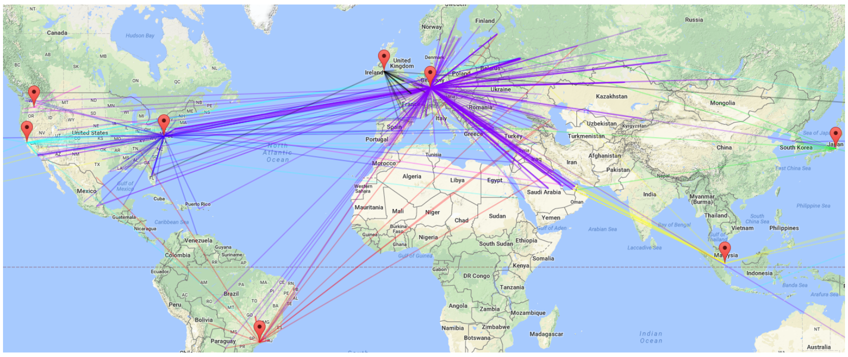


Fig. 9. Broadcaster to RTMP server mapping samples. Markers are RTMP server locations.

also exceptions, which means that besides broadcaster coordinates, there may be other variables that play a role in the mapping.

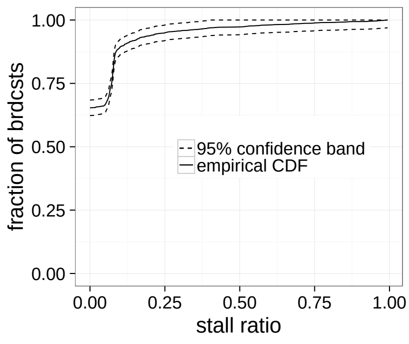
The HLS streams were delivered from only two distinct IP addresses. Their locations were somewhere in Europe and in San Francisco. It appears that the RTMP stream gets repackaged and delivered to Fastly CDN by Periscope servers, possibly the RTMP servers to which the video is transmitted by the broadcaster 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 [35], 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.

5.3 Playback Stalling

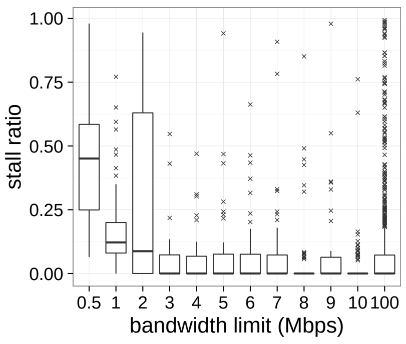
To analyze playback stalling, we leverage the statistics that the app reports for RTMP streams after playback, which include the number of stall events and the average stall time of an event. For HLS streams, the app only reports the number of stall events. We calculate the *stall ratio* for the RTMP 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 10(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 get an idea what this could mean in terms of user engagement, in [7] the authors report a nearly linear and steep drop (by over 50%) in user engagement with live streaming when stall ratio grows from 0 to 0.1.

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 10(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 6), 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.



(a) no bandwidth limiting

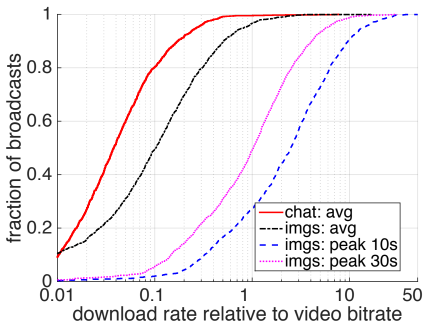


(b) bandwidth limiting

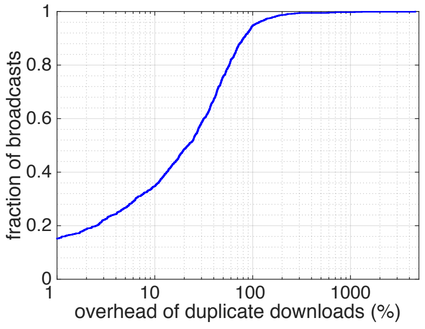
Fig. 10. Analysis of the stall ratio for RTMP streams with and without bandwidth limiting.

5.4 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. 11. 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.

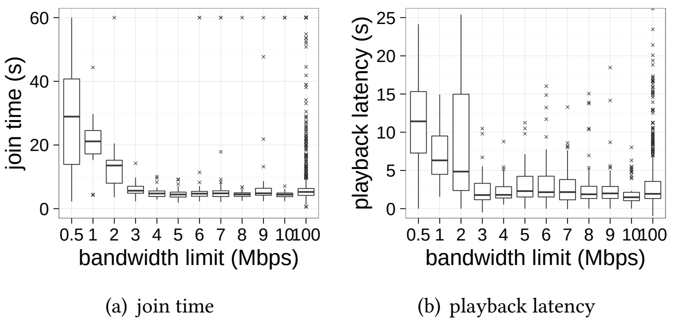


Fig. 12. The boxplots show that playback latency and join time of RTMP streams increases when bandwidth is limited. Notice the difference in scales.

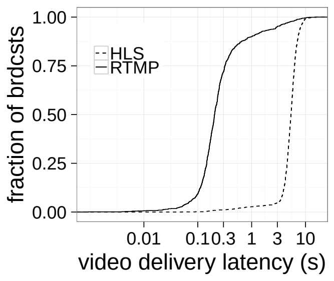


Fig. 13. HLS video delivery latency is much longer than that of RTMP streams.

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 type. We plot the CDF of download rates of the profile pictures and chat relative to the video bitrate in Figure 11(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 11(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.

5.5 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 12(a) display the results for RTMP streams. In addition, we plot in Figure 12(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.

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

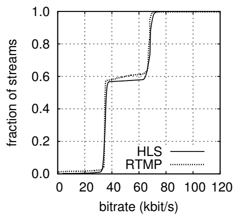


Fig. 14. Characteristics of the captured audio tracks.

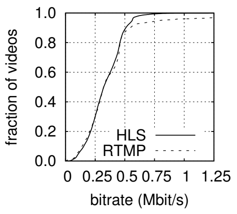


Fig. 15. Characteristics of the captured video tracks.

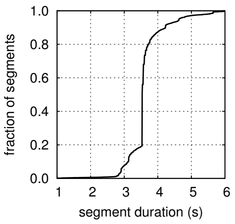


Fig. 16. Average duration of the HLS segment.

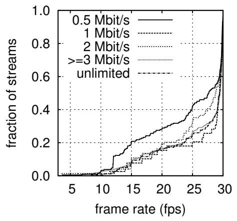


Fig. 17. Frame rate vs bandwidth limit (RTMP).

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 13. 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 [35].

6 AUDIO AND VIDEO QUALITY

6.1 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 in Section 5.1. 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 reconstructing the multimedia TCP stream using Wireshark, single segments are isolated by saving the response of HTTP GET request which contains an MPEG-TS file [14] ready to be played. For RTMP, we exploit the Wireshark dissector that can extract the audio and video segments. We used the libav (http://libav.org) 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 [4], 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 6.3.

6.2 Audio and Video Coding

Both RTMP and HLS streams use standard codecs for audio and video, namely AAC (Advanced Audio Coding) for audio [16] and AVC (Advanced Video Coding) for video [15]. Audio is sampled at 44,100 Hz, 16 bit, encoded in Variable Bit Rate (VBR) mode at about 32 or 64 kbps, as shown in Figure 14, sufficient to transmit almost any type of audio content (voice, music, etc.) at a quality expected from a capture with 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, probably due to the fact that the uploading device had some issues, e.g., glitches in the real-time encoding or during upload. Hence, concealment must be applied to the decoded video.

Fig. 15 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 [5]. More details about quality will be given in the next section.

An interesting parameter for the HLS case is the segment duration. Figure 16 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.

We also studied the video frame rate of RTMP streams. Figure 17 shows the CDF of frame rate for different bandwidth limits. When bandwidth is particularly low, i.e., 0.5 Mbps, lower frame rates happen about twice as often compared to other bandwidth conditions. Such behavior indicates that lower video bitrates are also achieved by limiting the frame rate. On the contrary, at about 1 Mbps or above enough bandwidth is available, therefore dropping frames is unnecessary.

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.

6.3 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 [4]. 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 [28], such a technique also considers the video content, allowing for more definitive conclusions about video quality and its evolution over time.

Figure 18 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 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

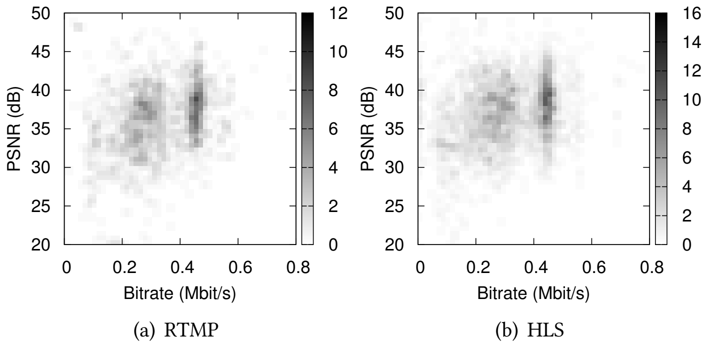


Fig. 18. PSNR as a function of the video bitrate for each session.

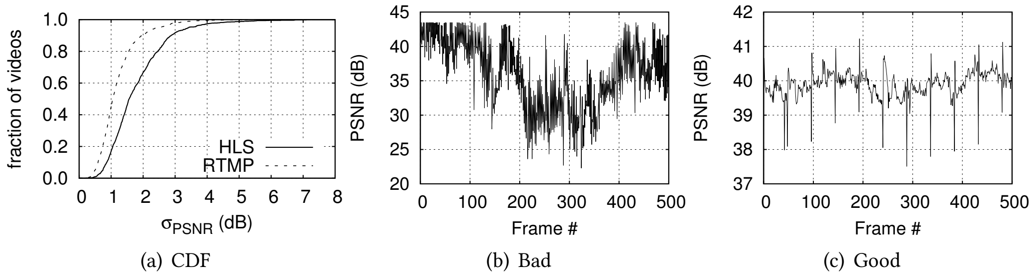


Fig. 19. PSNR behavior over time: CDF for all sessions and two sample cases (first 500 frames)

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. Consider that variations of 5-6 dB PSNR correspond to a significant variation of the corresponding Mean Opinion Score (MOS) [32], which is in practice the user satisfaction level of the video quality. In [32] 25 dB is considered to be a fair video quality, whereas at least 37 dB are required to achieve an excellent one. Therefore, it is quite surprising that most encoders stay at such moderate bitrate and do not attempt to increase video quality even when the available bandwidth would allow it.

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 19(a) shows the 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.

Such variations can be interpreted as significant variations of the user quality of experience. PSNR has, in fact, been shown to be reasonably well correlated with subjective quality if the same content and codec is considered [13].

An example of the behavior of the PSNR for two extreme cases (σ_{PSNR}=4.66 dB and 0.41 dB respectively) as a function of time is shown in Figure 19(b) and 19(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 19(b) confirms that the quality variations are significant and

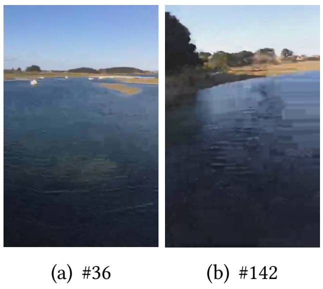


Fig. 20. Sample frames from the "bad" sequence in Fig. 19(b), showing high and low PSNR values.

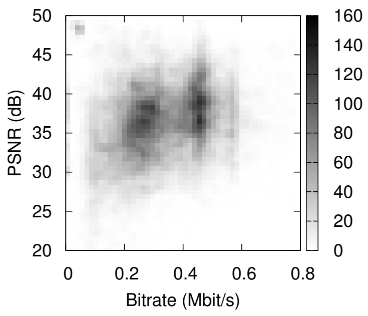


Fig. 21. PSNR as a function of the video bitrate for each single segment in the HLS sessions.

very noticeable. In Figure 20 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 19(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.

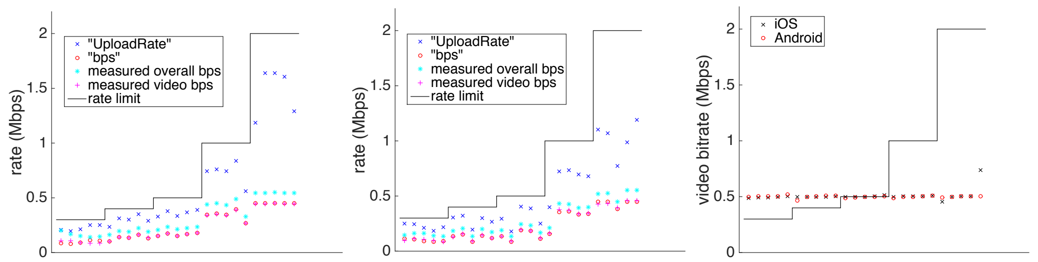
Regarding the video content, it is also interesting to note that, for the case of HLS sessions, in each segment both the bitrate and the quality can vary significantly, as shown in Figure 21. This is in agreement with Figure 19(a) where higher PSNR variability over the time span of the session could 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. Nevertheless, such large video quality variability among chunks is quite surprising, since HLS streams necessarily need some server-side processing that could potentially contribute to make the quality more stable.

7 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.

7.1 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



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

Fig. 22. Bitrates with constant upload rate limit.

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 6.1.

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.

7.2 Broadcasting with Constant Available Bandwidth

We first measure the bitrate selection of the applications when there is constant amount of bandwidth available. Figure 22 shows the results. The Android version of the Periscope application chooses the video bitrate rather conservatively, as we can see from Figure 22(a). The two attribute values `UploadRate` and `bps` are embedded into the video stream and the former is likely to be the application's estimate of the available amount of bandwidth, while the bps value matches the chosen target video bitrate. In light of these measurements, the app exercises caution and tends to underestimate the amount of available bandwidth, at least 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 22(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, 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, 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

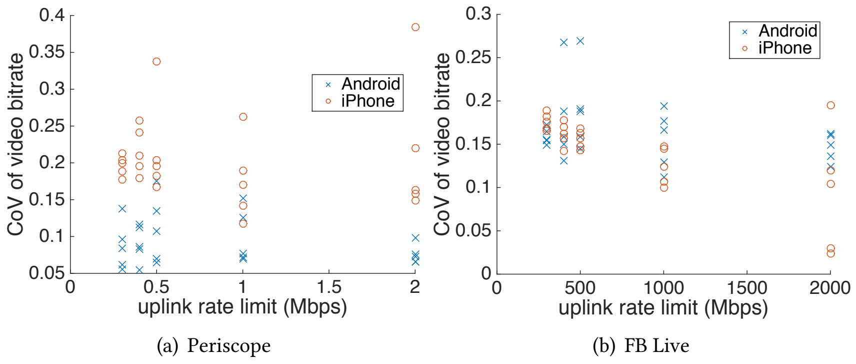


Fig. 23. Coefficient of variation of video bitrate with constant upload rate limit.

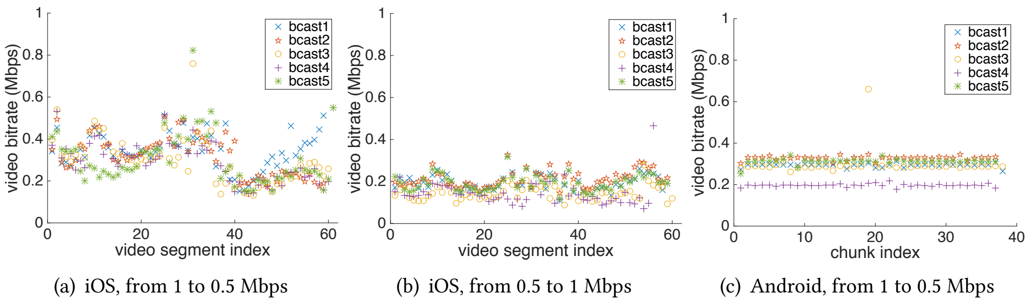


Fig. 24. Video segment bitrates from Periscope when upload rate limit changes in the middle of the broadcast.

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 did not adjust the video bit rate according to any bandwidth estimates. Instead, it always streamed video with a target bitrate around 0.5 Mbps, except for some outliers, as is clearly visible in Figure 22(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."

We also computed the coefficient of variation of the video bitrate for both version of the two applications and the different bandwidth limits (Figure 23). The most interesting thing to note from Figure 23(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 24(b) and 24(c).

7.3 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 24 shows bitrates per video segment of Periscope broadcasts of the two first minutes of the same Big Buck Bunny video (each segment lasts two seconds). After the first minute,

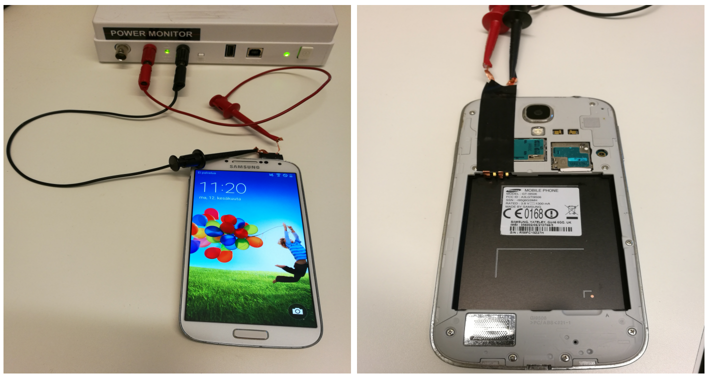


Fig. 25. Power measurement setup.

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 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 24(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.

7.4 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 did not find multiple representations of HLS delivered Periscope videos, whereas 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 further experiments because viewer side bitrate adaptation is fairly well understood by the research community.

8 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 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.

8.1 Measurement Setup

We connected a Samsung Galaxy S4 4G+ smartphone to a Monsoon Power Monitor (http://www.msoon.com) in order to measure its power consumption. The setup is presented in Figure 25. The battery of the phone is bypassed by covering the voltage terminal of the battery with insulating tape and copper foil tape connects the power monitor to the phone power input. 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 [34]. Screen brightness was full in all experiments and the sound was off. The phone was connected to the Internet through non-commercial WiFi and LTE networks (full-fledged LTE network deployed at Aalto University having DRX enabled with typical timer configuration).

8.2 Results

Figure 26 shows the results for both apps. 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. 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.

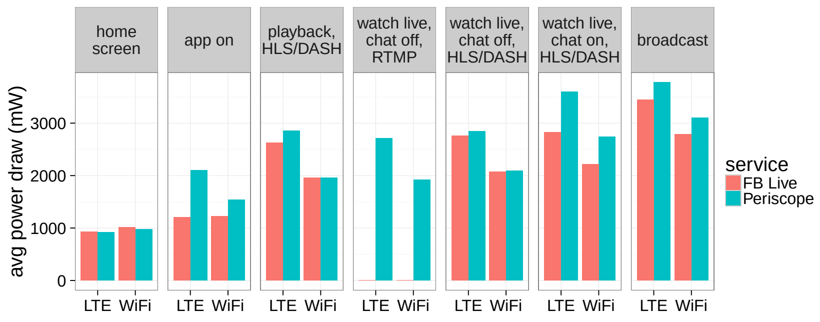


Fig. 26. Average power consumption of the two apps in different modes.

Playing back non-live videos with Periscope consumes a similar amount of power as watching live videos. The power consumption difference of RTMP vs HLS with the Periscope app is also insignificant. In general, we did not observe notable differences between viewing a highly popular broadcast or one with only a few viewers. Interestingly, enabling the chat feature of the Periscope videos raises the power consumption substantially, which is not the case for FB Live even though both apps show chat messages in the same way including the profile pictures. Broadcasting video consumes the most power with both apps. 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 [34]. 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 5.4, 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 turn off the display of those pictures in the chat.

Table 2 lists the resulting battery life of the mobile device used in the experiments. Live broadcasting empties the battery in less than three hours with both applications when LTE interface is

Table 2. Battery life (hours) of Samsung Galaxy S4 4G+ with Periscope and FB Live.

App	Radio	home screen	app on	playback, HLS/DASH	live, chat off, RTMP	live, chat off, HLS/DASH	live, chat on, HLS/DASH	broadcast
Periscope	WiFi	10.2	6.4	5.0	5.1	4.7	3.6	3.2
	LTE	10.7	4.7	3.5	3.6	3.5	2.7	2.6
FB Live	WiFi	9.7	8.1	5.0	-	4.8	4.5	3.5
	LTE	10.6	8.2	3.8	-	3.6	3.5	2.9

used for wireless communication and no other applications consuming power at the same time. Viewing live streams results into battery life of 2.7-5.1 hours mainly depending on the radio used and, only with Periscope, whether the chat is activated. Compared to idle device with screen on (home screen), the effect of the app in all modes is substantial.

9 LESSONS LEARNED

In this section, we discuss how the findings made through this measurement study can be used to improve mobile live video broadcasting systems or design better ones in the future.

- Our measurements indicate that there are very large differences in broadcast popularity and its variation (viewer churn). Ability to predict the popularity dynamics and viewer origins can enable (cost) optimization of the video content delivery, e.g., through the use of multiple different CDNs, a common practice with "traditional" video streaming providers [2], with different pricing schemes. However, developing such predictive models requires more extensive datasets than the ones we currently possess.
- The use of different streaming protocols to serve the same content to users leads to different end-to-end latencies for the users as well as different video quality variability over time at the frame level (higher for the HLS streams than for the RTMP ones). As short term solution, server processing could help smoothen the quality variation in the HLS case. In longer term, using a short segment length with HTTP streaming, which is feasible with HTTP/2.0 using server push without the request explosion problem [36], should deliver low enough latency in which case RTMP streaming to viewers is unnecessary. Furthermore, directly transmitting DASH segments from the broadcasting device would alleviate the need for re-encoding, and possible transcoding if scalable video coding is used [29], prior to CDN delivery.
- Limited rate adaptation at the broadcaster side of the current applications calls for better solutions as rate adaptation directly affects the QoE of all the viewing clients. The solutions used for on-demand streaming may not be applicable as such because the roles of the mobile device and server are reversed. We believe that mobile network assisted rate adaptation solutions could be especially useful here and the recent SAND (server and network assisted DASH) as part of the DASH standard [1] provides a framework to design such solutions.
- A small detail of a simple application feature such as profile pictures in a chat can have a large impact on behavior of the mobile application itself, which underlines the need for careful design of the entire system.

10 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 is 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] ISO/IEC 23009-5. 2017. Dynamic adaptive streaming over HTTP (DASH) – Part 5: Server and network assisted DASH (SAND). (2017).

[2] Vijay K. Adhikari, Yang Guo, Fang Hao, Volker Hilt, Zhi-Li Zhang, Matteo Varvello, and Moritz Steiner. 2015. Measurement Study of Netflix, Hulu, and a Tale of Three CDNs. *IEEE/ACM Trans. Netw.* 23, 6 (Dec. 2015), 1984–1997.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Quan Huynh-Thu and Mohammed Ghanbari. 2008. Scope of validity of PSNR in image/video quality assessment. *Electronics letters* 44, 13 (2008), 800–801.

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

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

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

[17] 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.

[18] 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.

[19] 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).

[20] 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.

[21] Periscope. 2016. Year One. https://medium.com/@periscope/year-one-81c4c625f5bc#.mzobrfpig. (March 2016).

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] Lingfen Sun, Is-Haka Mkwawa, Emmanuel Jammeh, and Emmanuel Ifeachor. 2013. *Guide to voice and video over IP: for fixed and mobile networks*. Springer Science & Business Media, p. 151.

[33] 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.

[34] Sasu Tarkoma, Matti Siekkinen, Eemil Lagerspetz, and Yu Xiao. 2014. *Smartphone Energy Consumption: Modeling and Optimization*. Cambridge University Press.

[35] 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)*.

[36] Sheng Wei and Viswanathan Swaminathan. 2014. Low Latency Live Video Streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop (NOSSDAV '14)*. ACM, New York, NY, USA, Article 37, 6 pages.

[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] 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.

[39] 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.

[40] 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.

[41] 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.