Mobile Live Streaming: Insights from the Periscope Service

Mobile Live Streaming: Insights from the Periscope Service

Leonardo Favario
Control and Computer Eng. Dept.
Politecnico di Torino, Italy
Email: leonardo.favario@polito.it

Matti Siekkinen
School of Science
Aalto University, Finland
Email: matti.siekkinen@aalto.fi

Enrico Masala
Control and Computer Eng. Dept.
Politecnico di Torino, Italy
Email: enrico.masala@polito.it

***Abstract*—Live video streaming from mobile devices is quickly becoming popular through services such as Periscope, Meerkat, and Facebook Live. Little is known, however, about how such services tackle the challenges of the live mobile streaming scenario. This work addresses such gap by investigating in details the characteristics of the Periscope service. A large number of publicly available streams have been captured and analyzed in depth, in particular studying the characteristics of the encoded streams and the communication evolution over time. Such an investigation allows to get an insight into key performance parameters such as bandwidth, latency, buffer levels and freezes, as well as the limits and strategies adopted by Periscope to deal with this challenging application scenario.**

I. Introduction

The popularity of new services for live streaming from mobile devices is increasing rapidly. For instance, in a recent announcement (March 2016), Periscope, acquired by Twitter even before the launch of its service, stated that more than 110 years of live video was being watched every day through their application [1]. Other competing services such as Meerkat and Facebook Live are following the same trend.

However, how such system work and in particular the quality of experience (QoE) that they are able to deliver has not yet been investigated in details. This work focuses on the Periscope service by analyzing the characteristics of publicly available live streams that any user can access through the platform. We automated the viewing process of Periscope live streams on an Android smartphone capturing a few thousand sessions while logging data traffic and other useful types of data exchanged between the application and the servers. Data have been later postprocessed to examine various aspects including network parameters (e.g., protocols, bitrate, latency), service-related parameters (e.g., estimated session duration, device movements) and QoE in terms of video playback buffers and smoothness.

The key insight obtained through this work include understanding how the two application-layer protocols employed by Periscope impact on the service, considering bitrates, coding schemes, and QoE. Moreover, media timestamps in the encoded streams allowed us to investigate the behavior of the main communication parameters, and in particular the playout buffer level and latency, also determining which is the minimum initial playback delay necessary to avoid freeze events. Other parameters embedded by the Periscope services have also been analyzed, including the bitrate values reported by the system and the position of the capturing device.

The work is organized as follows. Sec. II describes related work in the area. Sec. III briefly overviews the Periscope service and how data collection has been performed, then Sec. IV analyzes in depth the information that can be extracted by the captures. Sec. V discusses the results by summarizing the main observations. Conclusions are drawn in Sec. VI.

II. Related Work

Several research works addressed the challenges of live streaming involving mobile devices. Issues such as optimal content distribution have been investigated [2], including scenarios where it is possible to communicate directly among devices [3], or where heterogeneous contributors and viewers are involved [4]. Other work investigated, for instance, how cloud-based systems can help in live streaming [5]. However, most of the research focused on systems where the mobile device is the receiver of the live streaming, as for Twitch.Tv [6], or other mobile VoD systems [7].

For the specific case of live streaming from mobile devices, some works studied user activities investigating, for instance, how to influence them [8] or how human factors can play a role in Periscope and Meerkat [9]. Regarding technical issues, optimization frameworks for uplink video transmission have been proposed for specific cellular communication scenarios [10], however little is known about how live mobile applications work in real-world conditions.

In this work we tackle the issue of analyzing the Periscope protocol through network traffic analysis, as already done in literature for other widespread applications such as Skype [11]. We already tried to dissect the Periscope service in general in [12], focusing on usage characteristics, protocols, QoE and mobile power consumption. In this work we aim specifically at the aspects of multimedia coding and transmission, analyzing in details the media coding parameters and the communication behavior over time considering in particular the playout buffer levels and freeze events.

III. Background

A. Periscope Overview

Periscope allows its users to send live video streams into the Internet and let other users view them. Access can be restricted

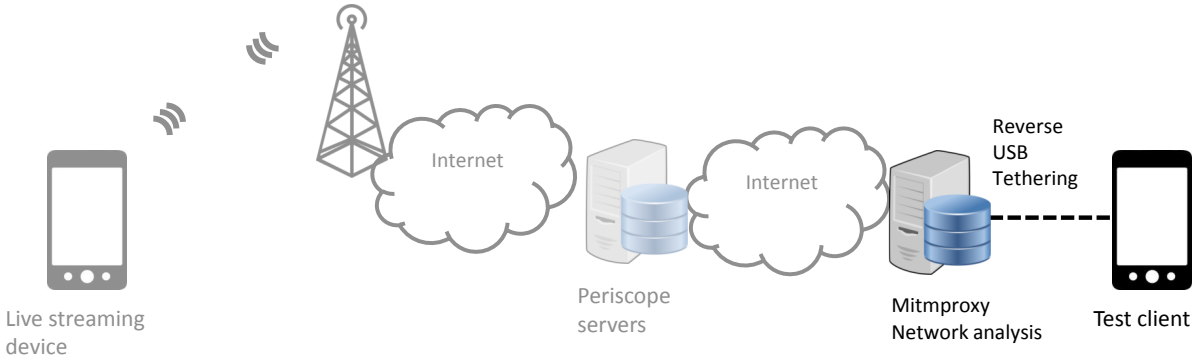


Fig. 1. Experimental setup: the right part shows our server that acts as mitmproxy, storage for the captured network traces and large bandwidth provider for the mobile device. In gray the rest of the system that cannot be directly controlled.

to chosen users; alternatively, streams can be made public.

A user has three options to discover public broadcasts. First, the app shows broadcasts in a rank. Alternatively, the map of the world can be explored to find broadcast within a specific geographical area. Last, a "Teleport" button is available in the app. When pressed, a randomly selected broadcast can be watched. The Periscope application communicates with the servers using an API whose specifications are not public and communication is protected by SSL.

B. Data Acquisition

Our goal was to collect information about a relatively large number of Periscope broadcasts in an automated fashion. In order to enable studying also the SSL traffic between the mobile client and Periscope servers, we set up a so called *man-in-the-middle proxy*, i.e. mitmproxy [13], in between the mobile device and the Periscope service as a transparent proxy. Fig. 1 depicts our test scenario, in which our server runs both the mitmproxy and a network analyzer which stores all packets. With some effort, we were able to examine the exchange requests between the clients and the servers and automate the requests using smartphones. Our scripts sent commands to the device through the Android debug bridge (adb) to press the app "Teleport" button which takes the user directly to a randomly selected live broadcast, then waited for 60 s, closed the session and started again. The script also captures, for each broadcast, all the data traffic between the device and the Internet. Therefore, the maximum media data length has been limited to 60 s in all experiments.

Two different smartphones have been used in our automated experiments: one Samsung Galaxy S3 and one S4. To rule out any issues due do the smartphone download speed they were connected to the Internet by means of reverse tethering through a USB connection to a PC connected to the university LAN with more than 100 Mbps of available bandwidth. In total, we collected 4,023 sessions.

IV. Multimedia Traffic Analysis

To carry out the traffic analysis we employed a wide variety of tools ranging from standard ones such as *wireshark* [14] for network trace analysis to *libav* [15] to inspect and decode the multimedia content. In addition, we developed a set of custom programs and scripts to extract information not directly available through such programs.

A. Protocols

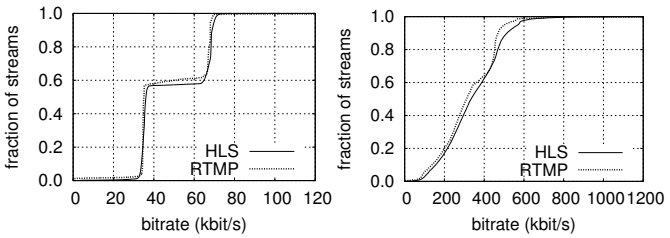
In our experiments we observed that Periscope uses either the Real Time Messaging Protocol (RTMP) [16] using source port 80 or the HTTP Live Streaming (HLS) to serve multimedia content. Our experiments collected 1,762 RTMP sessions and 2,261 HLS sessions. Further investigation showed that RTMP is delivered through IP addresses corresponding to Amazon EC2 instances whereas HLS content relies on the Fastly Content Delivery Network (CDN).

RTMP is used when only few people watch the stream [12]. RTMP can convey different types of data. In this work we focused on the audio and video segments. Each RTMP segment is prefixed by a header which includes, in particular, the media type, the body size, and timestamp information. Each body includes one control byte that specifies the multimedia format followed by the body data itself with the multimedia compressed content. By trial and errors we determined that some of data at the beginning of the body contains some Periscope proprietary information which must be discarded in order to properly analyze the media content. Once the media content has been extracted, it can be analyzed with standard toolchains (e.g., *libav* [15]) which provide summary information as well as the possibility to decode the content itself.

When many users watch the stream, a different protocol is used. Periscope servers resort to the so called HTTP Live Streaming (HLS), by serving content as HTTP resources (i.e., *segments*) which are periodically requested and downloaded by the client. Such resources are encoded according to the MPEG Transport Stream (MPEG-TS) [17] format which can multiplex both audio and video and synchronizing them using the Presentation Time Stamp (PTS) information embedded into the MPEG-TS format.

B. Audio Characteristics

All the audio content is encoded using the Advanced Audio Coding (AAC) [18], sampled at 44,100 Hz 16 bit. The content



(a) audio (b) video

Fig. 2. Bitrate of audio and video streams: two preferred values can be clearly observed for the case of audio.

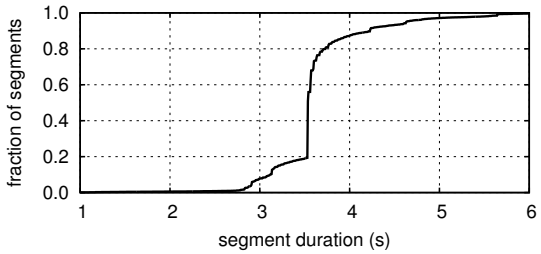


Fig. 3. Duration of HLS segments.

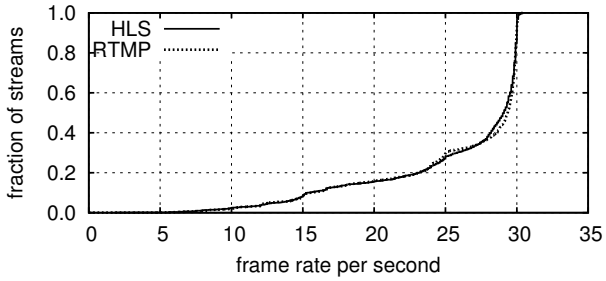


Fig. 4. Average frame rate of video stream.

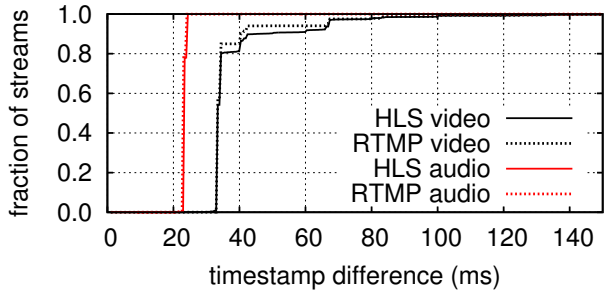


Fig. 5. Cumulative distribution of difference between two subsequent timestamps.

is encoded using a Variable Bit Rate (VBR) codec, hence each audio frame has a different size. However, the variation of the average bitrate is limited. The average bitrate is typically either 32 or 64 kbps. Fig. 2a shows the cumulative distribution of the bitrate. Note that the audio content has been reconstructed in Audio Data Transport Stream (ADTS) format [18] so the values in the figure are slightly higher than 32 or 64 kbps due to the inclusion of ADTS headers. When the bitrate is lower than 32 kbps, typically the content is almost silence. Also, we observe that there is no significant difference between the measures for the HLS and the RTMP case.

C. Video Characteristics

All video content is encoded using the Advanced Video Coding (AVC) standard [19], using resolution always equal to 320×568, up to 30 frames per second (fps). Fig. 2b shows the video bitrate whose typical value ranges between 100 and 600 kbps. The difference between HLS and RTMP is limited. For the HLS case, Fig. 3 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. Nevertheless, the corresponding bitrate can vary significantly as shown in Fig. 2b.

Figure 4 shows the average frame rate of the video streams: there is a strong tendency to use 30 fps, but other rates are possible since occasionally frames are skipped, i.e., they are not encoded nor transmitted. Note that there is no significant difference between the HLS and RTMP cases.

Moreover, occasionally it seems that some frames are missing (not just skipped as in the previous case). Such condition is detected by unexpected gaps in the Picture Order Count (POC). In this case, to decode the video some concealment technique must be applied. We attribute such a fact to the uploading device which might have experienced some encoding or transmission issues.

Fig. 5 shows the difference between timestamps of consecutive frames in the video stream. Timestamps are extracted either from the RTMP header or directly from the MPEG-TS PTS values. The cumulative distribution shows that 33.3 ms (equivalent to 30 fps) represents the large majority of cases. Other common difference values are 40 and 66.6 ms (respectively equivalent to 25 and 15 fps).

The video encoding pattern typically uses the IBPBP...I structure, where one frame containing B-type slices is inserted between frames containing either I-type or P-type slices. Only in 17.3% of the cases B-type slices are not used. For HLS, a closed Group-Of-Picture (GOP) structure is used, i.e., there is no coding dependency among different HLS segments, whereas RTMP uses both open and closed GOPs. The HLS solution potentially allows to seamlessly switch between different representations if necessary. However, detecting representation changes is not easy due to the variability of the video content which might cause a rate change on its own. Moreover, currently we did not observe any explicit hint of such feature being implemented, e.g., changes in the structure of the name of requested URLs. However, in our setup we did not expect such changes since there are no significant download bandwidth variations.

Most of the video content is encoded using one slice per frame, whereas occasionally two slices are used to split the picture horizontally in two parts. We speculate this could be due to the encoding device which might speed up encoding by parallelizing the coding operations for one frame.

Each frame containing I-type slices is preceded by an AVC

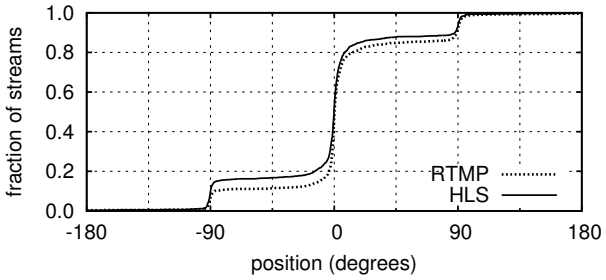


Fig. 6. Average position of the mobile device during the whole video session. Zero represents vertical, 90 or -90 is horizontal.

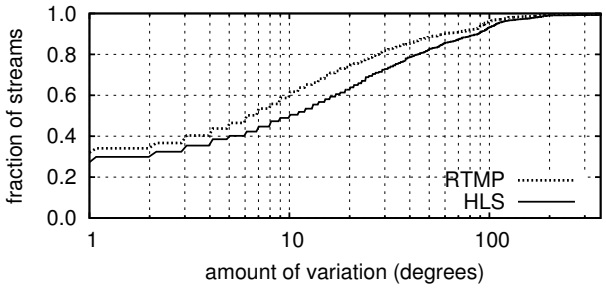


Fig. 7. Amount of degrees covered by the position of the mobile device during the whole video session.

Sequence Parameter Set (SPS) and a Picture Parameter Set (PPS). This allows any viewer joining at any time to have all the necessary information to start decoding from that point. SPS and PPS are typically inserted every 36 frames (that is 1.2 s at 30 fps).

D. Embedded Information by Periscope

Additional information is embedded into the video stream just after the SPS and PPS, using the AVC Supplemental Enhancement Information (SEI) NALU data type, with an application-defined format. Although we were not able to decode all information, simple inspection of the data allowed to extract the name of some properties and their value. In particular, four fields attracted our attention: the *uploadrate* which we assumed to be the available rate estimated by the application, the *bps* which seems to be the actual transmitted bitrate, the *ntp* which seems to be the time at which the data was produced or received by the Periscope server, and the *rotation* which seems to be the position of the device as detected by the sensors of the mobile device.

The rotation information is useful to roughly estimate, by means of a physical world measure, how much the device is moved across the session. Fig. 6 show the average position of the mobile device during the whole video session. When the device is hold vertically, the value is zero degrees. Values equal to 90 or -90 correspond to horizontal position. Since devices are often moved during the video session, Fig. 7 show how many positions (quantized as integer degree values) are covered by the position of the mobile device during the whole video session. The majority of the devices are held in a relatively stable position for the whole duration of the session (60% of the devices is moved less than 10 degrees) whereas

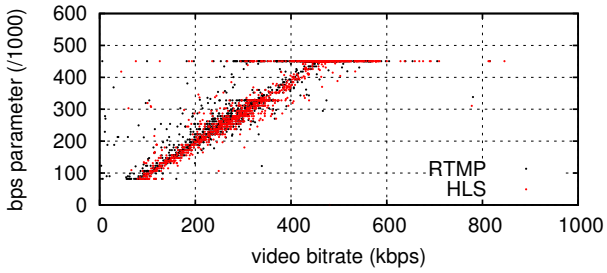


Fig. 8. Values of the *bps* parameter versus the effective average video bitrate.

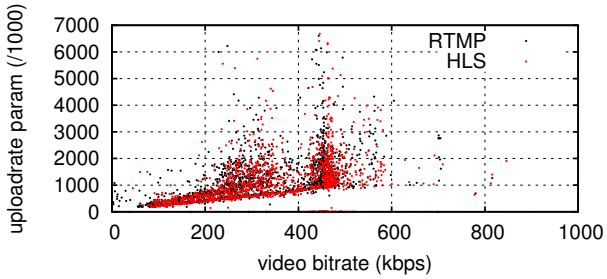


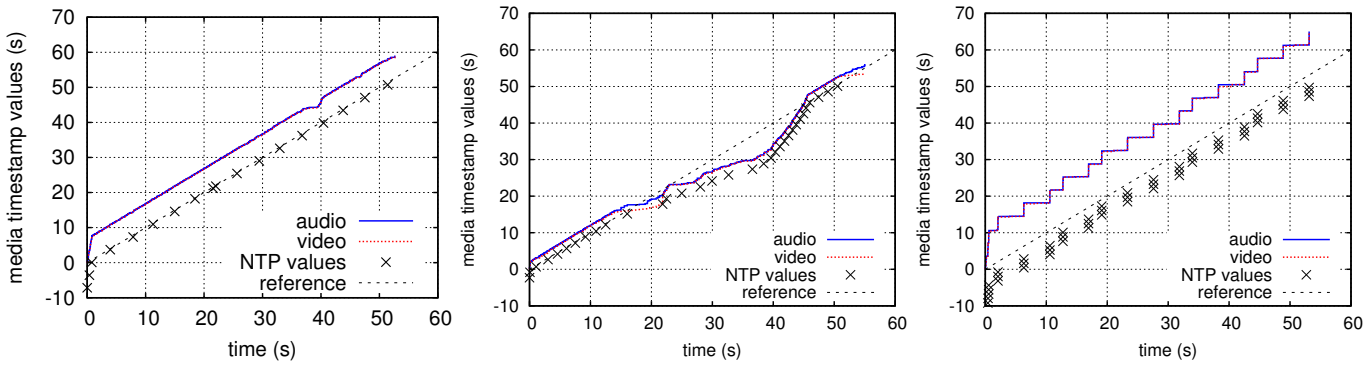
Fig. 9. Values of the *uploadrate* parameter versus the effective average video bitrate.

others might significantly change (e.g., about 10% rotates more than 90 degrees). Large values are relatively common since devices are often held in hand as opposed to being left in a fixed stable position. By visually inspecting the decoded content, we noticed that many movements often yield a low quality video, therefore the rotation parameter could suggest the presence of video segments with low quality.

Fig. 8 shows the value of the *bps* parameter as a function of our measured video bitrate. From the strong correlation of the points in the graph we speculate that the *bps* parameter could be the target encoding bitrate decided by the application and that, in any case, is capped at 450,000. Analogously, Fig. 9 shows the value of the *uploadrate* parameter. It seems reasonable to assume that this could be an estimate of the available upload bandwidth as seen from the mobile device which, in fact, can be relatively high, up to about 6,000 kpbs. However, the bitrate of the received video is about half of such value, probably not to saturate the uplink of the mobile device, and in any case is capped at about 450 kbps. Finally, note that no differences between RTMP and HLS seem to be present for both parameters.

E. Real-Time Communication Aspects

To investigate in more details the real-timeliness of the communication system we analyzed in depth the captured network traces and the evolution of the header timestamps of each segment (in case of RTMP) or the PTS values (in case of HLS). The PTS or RTMP timestamp and the arrival time of each byte in the TCP flow, which can be precisely determined through the network traces, allow to investigate the behavior of the session over time. For instance, Fig. 10 shows three sample sessions. All of them show, on the vertical axis, the timestamp of the audio and video frames (in the RTMP flow or as PTS in the MPEG-TS), as well as the Periscope NTP values embedded



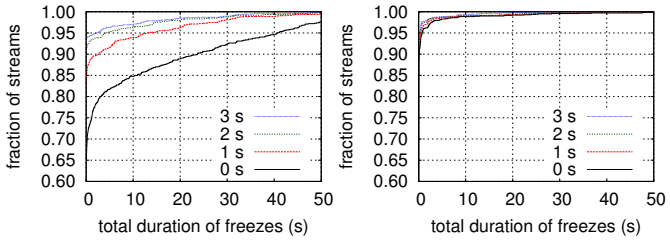
(a) RTMP sample #1 (b) RTMP sample #2 (c) HLS sample #1

Fig. 10. Timestamps of audio and video frames, and of the Periscope NTP value encoded in the AVC SEI NALUs shown as a function of their reception time. A reference playback line is also shown assuming playback starting when the first packet is received.

in the AVC SEI NALU. Moreover, a reference playback line is also plotted, assuming as a time reference the reception of the first packet. Audio and video timestamps are almost always superimposed, in fact they are multiplexed and transmitted very closely in time. Fig. 10a shows an initial fast buffering phase (about 1 s), then the TCP transmission rate matches the one of the media content, except occasionally when there might be network communication issues (e.g., between 37 and 40 s). If such impairments are limited, they have no effect on the playback, but if the rate drop significantly for a larger amount of time a stall or freeze event may happen (i.e., the video freezes and the audio stops), as it happens in Fig. 10b. Note that the vertical difference between a 45-degree line and timestamp values allow to determine how much media data is present in the playout buffer. The larger the value of the initial playback delay, the lower the position of the 45-degree line in the figure. Hence, in Fig. 10b a larger initial playback delay would help in mitigating the length of the freeze event, at the expenses of a higher latency. Also, note that in case of transmission impairments the system tends to privilege audio over video (see the interval 10 to 15 s where the audio curve is higher than the video one). Finally, Fig. 10c shows the same plot for an HLS session. Audio and video timestamps exhibit a staircase behavior, since during the vertical phase a new HLS segment is quickly downloaded, then nothing happens until the next HTTP request. Note also that our setup includes a fast Internet connection (over 100 Mbps) which allows to download segments from the CDN servers very quickly.

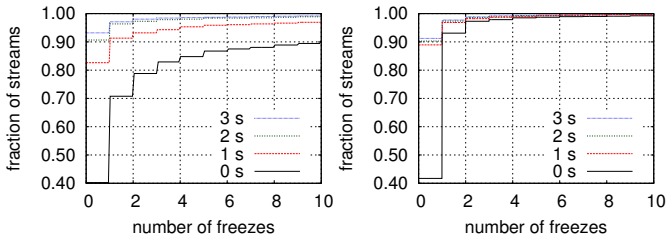
Despite the fact that we do not know exactly the delay between the reception of the first packet and the start of the playback, we can compute a lower bound on such value by assuming that the playback starts immediately upon the reception of the first media data. Such condition is represented in Fig. 10 by the reference playback line.

To understand the influence of the initial playback delay on the number of freeze events and their total duration, we computed the cumulative distributions for several initial playback delay values. Results are shown in Fig. 11 and Fig. 12. Clearly,



(a) RTMP (b) HLS

Fig. 11. Duration of freezes for the RTMP and HLS sessions with the initial playback delay specified in the legend.



(a) RTMP (b) HLS

Fig. 12. Number of freezes for the RTMP and HLS sessions with the initial playback delay specified in the legend.

as the delay increases the number and duration of freezes decrease. Note the strong difference between the RTMP and the HLS case. The latter has much shorter total duration of freezes probably because it benefits from higher latency compared to RTMP. However, a small initial playback delay is essential not to incur in freezes just after playback starts, which is the reason for the high fraction of streams with 1 freeze in Fig. 12. It is also possible to compute the cumulative distribution of the minimum initial playback delay that would be needed to avoid any freeze event. Fig. 13 shows that less than about 20% (RTMP) or 10% (HLS) of the streams need to have an initial playback delay of more than 1 s to compensate for network communication issues. At about 7 s curves are almost flat, i.e., there is no further advantage in increasing the latency.

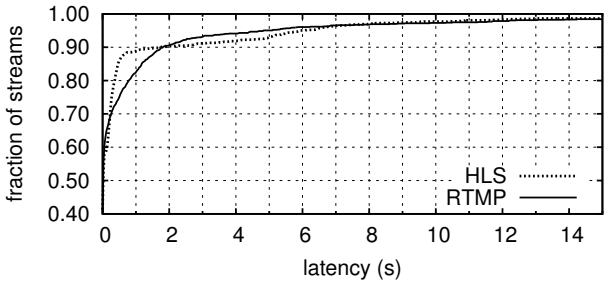


Fig. 13. Initial playback latency necessary to avoid freezes.

V. Discussion

Summarizing the findings of the previous sections, in general it seems that Periscope can provide a reasonably good mobile live streaming service. In our setup we avoided downlink bandwidth limitations, therefore all effects are due to either the uploading device or Periscope server processing. In general, the typical constraints of the scenario can be fulfilled by the uploading device, especially in terms of reasonable QoE and limited latency. Also, by observing the NTP timestamp values it seems that a relatively low latency can be achieved when using the RTMP protocol, despite the underlying connection is TCP which is notoriously ill-suited for real-time communications. Moreover, in case the download bandwidth is not a limitation, setting the initial playback delay to 1 s is enough to avoid playback disruptions for 80% (RTMP) or 90% (HLS) of the sessions. Further increase of such value up to about 7 s allows smooth playback for 95% of the session.

Variable frame rate video, observed through variable differences between timestamps, is used in more than 20% of the cases to mitigate bandwidth limitations by intentionally skipping a few frames in the video sequence. No such event happens for audio probably due to its limited bandwidth requirements.

However we also noticed that occasionally some video frames appears to be missed unintentionally in the compressed stream. This situation is detected by observing the evolution over time of the AVC Picture Order Count (POC) field in NALU header. Probably, data are passed as is by Periscope servers to the viewers which need to detect and deal with such issue, e.g., applying some concealment technique.

Concerning the characteristics of the audio and video content, no significant differences are observed when data is transmitted using either RTMP or HLS. The only discriminant between the use of HLS as opposed to RTMP seems to be the number of viewers [12].

VI. Conclusion

In this work we investigated in details the characteristics of the Periscope mobile live streaming service. The characteristics of a large number of publicly available streams have been investigated in depth, focusing on multimedia content and communication timings. The latter aspect showed that the real-time streaming service can be delivered over TCP to a multiplicity of users using both a specialized application level protocol such as RTMP and a more scalable HTTP streaming approach suitable for a large number of viewers by accepting higher latency. We hope that these findings, both qualitative and quantitative, will contribute to better understand the issues faced by actual multimedia coding and transmission systems in the real world conditions in which mobile live streaming systems operate. Future work will be devoted to investigate the behavior of the system when the download bandwidth is limited to understand possible adaptation strategies.

References

[1] Periscope, "Year one," Online (accessed 27 May 2016): https://medium.com/@periscope/year-one-81c4c625f5bc#.mzobrfpig, Mar. 2016.

[2] T. Lohmar, T. Einarsson, P. Fröjdh, F. Gabin, and M. Kampmann, "Dynamic adaptive HTTP streaming of live content," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, June 2011, pp. 1–8.

[3] L. Zhou, "Mobile device-to-device video distribution: Theory and application," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 3, pp. 38:1–38:23, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2886776

[4] Q. He, J. Liu, C. Wang, and B. Li, "Coping with heterogeneous video contributors and viewers in crowdsourced live streaming: A cloud-based approach," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 916–928, May 2016.

[5] Y. Zheng, D. Wu, Y. Ke, C. Yang, M. Chen, and G. Zhang, "Online cloud transcoding and distribution for crowdsourced live game video streaming," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, no. 99, pp. 1–1, 2016.

[6] C. Zhang and J. Liu, "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*, ser. NOSSDAV '15. New York, NY, USA: ACM, 2015, pp. 55–60. [Online]. Available: http://doi.acm.org/10.1145/2736084.2736091

[7] Z. Li, J. Lin, M.-I. Akodjenou, G. Xie, M. A. Kaafar, Y. Jin, and G. Peng, "Watching videos from everywhere: a study of the PPTV mobile VoD system," in *Proc. of the 2012 ACM conf. on Internet Measurement Conference*. ACM, 2012, pp. 185–198.

[8] S. Wilk, D. Wulffert, and W. Effelsberg, "On influencing mobile live video broadcasting users," in *2015 IEEE International Symposium on Multimedia (ISM)*. IEEE, 2015, pp. 403–406.

[9] J. C. Tang, G. Venolia, and K. M. Inkpen, "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*, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 4770–4780. [Online]. Available: http://doi.acm.org/10.1145/2858036.2858374

[10] A. El Essaili, L. Zhou, D. Schroeder, E. Steinbach, and W. Kellerer, "QoE-driven live and on-demand LTE uplink video transmission," in *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP)*. Hangzhou, China: IEEE, Oct. 2011, pp. 1–6.

[11] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi, "Detailed analysis of skype traffic," *IEEE Transactions on Multimedia*, vol. 11, no. 1, pp. 117–127, 2009.

[12] M. Siekkinen, E. Masala, and T. Kämäräinen, "Anatomy of a Mobile Live Streaming Service: the Case of Periscope," *ArXiv e-prints*, May 2016. [Online]. Available: https://arxiv.org/abs/1605.04270

[13] "Mitmproxy Project: https://mitmproxy.org/."

[14] "Wireshark Project: https://www.wireshark.org/."

[15] "LibAV Project: https://libav.org/."

[16] H. Parmar and M. Thornburgh, "Adobe's real time messaging protocol v. 1.0," Online (accessed 27 May 2016): https://www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf, Dec. 2012.

[17] ISO/IEC 13818-1, "MPEG-2 Part 1 - Systems," ISO/IEC, Oct. 2007.

[18] ISO/IEC 13818-7, "MPEG-2 Part 7 - Advanced Audio Coding (AAC)," ISO/IEC, Jan. 2006.

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