Measurements of Live Video Uploads

Vikram Nathan, Eric Manzi
Massachusetts Institute of Technology
{vikramn,ermanzi}@mit.edu

1. MOTIVATION

Live videos are morphing the landscape of internet video streaming. Periscope (now Twitter) launched its live streaming service in 2014 [10]. One year later, Facebook launched the Live Video feature to all its users [5]. Even more recently, in 2016, YouTube made Live Streaming a prominent part of its mobile app to compete with Twitter and Facebook.

Being a young platform, relatively little is known about live video streaming systems. Companies are reluctant to disclose sensitive details about scale or implementation, and it has only recently been the subject of research interest. Recent work on Periscope has surveyed the scale of the system and the video quality experienced by viewers but has focused mostly on the download component instead of the upload [1].

However, the upload pipeline is what makes live streaming significantly more complex than traditional video streaming systems, which typically contain only a download component. First, more latent variables are involved: applications must choose an upload bitrate for the streamer in addition to the download bitrate from the server. The upload bitrate upper bounds the download bitrate and thus the viewer's video quality.

Second, live streaming systems must also minimize *end-to-end delay* or *liveness*, the time between when a frame is recorded by the sender to when it is played on the viewer's device. Liveness is valuable to streaming systems like Facebook Live, which allow realtime viewer interactivity in the form of 'reactions' and comments. A large end-to-end delay means that a sender receives viewer feedback far too late to adjust for it; a minimal delay is thus critical to the sender's experience. End-to-end (E2E) delay is a function of bitrate and link conditions for both the upload and download, further adding to the interplay between the network and user experience.

Both the upload flow and E2E delay add complexity to any analysis of live video streaming. Yet no work to our knowledge has specifically addressed how the upload flow affects the streaming experience. In this study, we use Facebook Live to examine the effect of the upload bitrate and bandwidth on various quality of experience metrics: rebuffer rate, download bitrate and video quality, and E2E delay. Since mobile phones are the predominant device used to stream, we evaluate uploads over cell traces as well. Our results establish relationships between upload variables and download metrics, while simultaneously identifying avenues for improvement in existing live streaming implementations.

2. BACKGROUND

Live streaming involves a single user streaming to multiple watchers, who may change during the course of the live stream. In the case of Facebook Live, the sender uploads its video to a Facebook server through RTMP[1] (Real-Time Messaging Protocol), a TCP-based protocol [7]. The server transcodes the stream into several predetermined bitrates, creating one second MPEG DASH segments for each bitrate. Watchers access the stream through one of several Facebook PoP (Point-of-Presence) caches using DASH. If the cache incurs a miss for a requested video segment, it fetches the segment from the transcoding server. Figure 1 illustrates the live streaming flow.

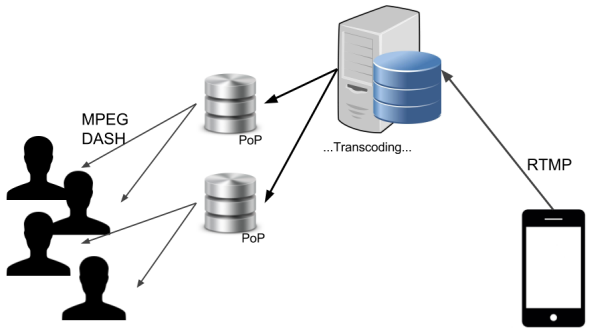


Figure 1: Life of a typical Facebook Live video.

Most Facebook Live streams are initiated by the mobile app, which does not give us the freedom to choose the upload bitrate. Therefore, we use Facebook's other

[1]Traffic first passes through a Facebook RTMP proxy.

streaming option, intended for publishers [14]: a Facebook Page can publish a live video by creating a live streaming link and using third-party software to stream to that link.

3. METHOD

All experiments were streamed through Facebook's infrastructure. We created a Facebook page to host the live videos, queried the Facebook Graph API [9] to create live streaming URLs, and ffmpeg [8] to send an RTMPS stream to those URLs. We then used the Facebook embedded viewer to "watch" the video using Chrome in an X virtual framebuffer (Xvfb) using pyvirtualdisplay [3]. This entire process is scripted, using Selenium [2] to automatically play the video in full-screen.

The video upload over RTMP is run in a Mahimahi link shell [11]. We collect data in three settings: a) with a fixed target upload bitrate and varying stable uplink capacity, b) with a fixed uplink capacity and varying upload bitrate, and c) with a fixed upload bitrate and mobile uplink traces scaled by different factors to simulate different congestion levels. The video download is run over an extremely high capacity network, to avoid the watcher's downlink being a bottlenck in our study.

To measure E2E latency, we timestamp each frame as it is uploaded using ffmpeg to superimpose the UTC time on top of the frame. On the viewer's end, we use the virual framebuffer to capture screenshots of the video playback every $\approx$ 200 ms and keep track of the time when each screenshot was captured. This gives us the playback time. We then use tesseract [13], an optical character recognition tool, to read the upload timestamp from each screenshot. The E2E latency for each frame is then computed as the difference between the playback time and the upload time. Note that this method is more accurate than measuring the download time as the time of that frame's download. Since the watcher may build up a play buffer, our approach most closely measures when a frame was actually displayed on the screen and thus seen by the viewer.

We measure playback continuity by finding video freezes in the playback. Multiple sequential identical screenshots indicates that the video froze. We find these by pixel-by-pixel comparison and use the screenshot timestamps to measure how long the playback was stalled for. Since a playback freeze is a good indicator of a rebuffer event, we measure the rebuffer count (the number of interruption events during the playback), the mean rebuffering time, and the rebuffering rate (total rebuffering time / playback duration). Note that this approach slightly underestimates the rebuffering time by at most $\approx$ 200ms per rebuffer; however, our decision to use the Facebook embedded viewer prevents accessing the buffer length directly. We also measure start-up latency this as the the time since the initial request for the live stream was issued until the first frame was played.

We measure actual upload and download bitrates by analyzing traffic recorded with TCP dump.

4. RESULTS

4.1 Varying upload capacity with fixed bitrate

We fix the target upload bitrate at 1000 Kbits/s since we measured (using TCP dump and Wireshark [12]) that the Facebook mobile app streams at this bitrate. We vary the network bandwidth and observe that the viewer has smooth video playback throughout the entire live stream, and the E2E latency is fairly constant at $\approx$ 10 seconds when the uploader's bandwidth isn't constrained (12 Mbps). We constrain the bandwidth and at 3 Mbits/s, we observe a jump in E2E latency 40 seconds into the playback, but no interruptions in the playback.

At 0.75 Mbits/s, the viewer's playback stalls on the first frame for the duration of the entire playback, even though the uploader is using the 750 Kbits/s of bandwidth available. Perhaps Facebook is dropping the frames it receives from the uploader because they are too stale, in the interest of liveness. This would mean the uploader's bandwidth is being wasted.

As expected, there's a negative correlation between E2E latency and the uploader's bandwidth. No correlation is observed between start-up delay and bandwidth.

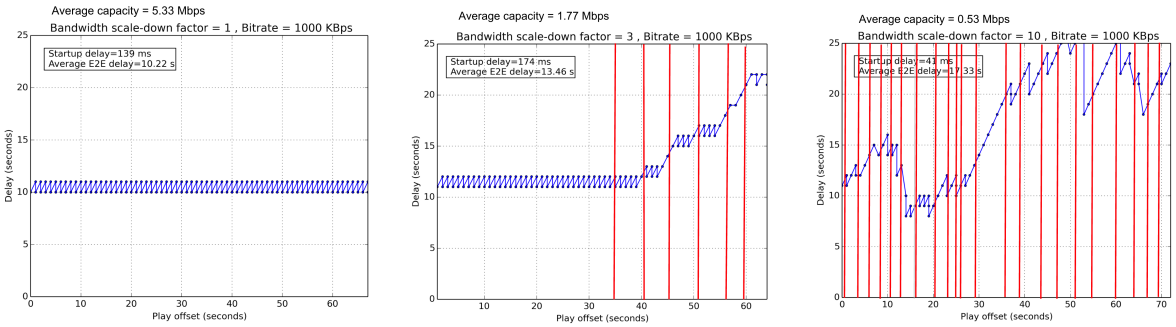
4.2 Varying upload bitrate with fixed bandwidth

With the uploader's bandwidth fixed at 1.5 Mbits/s, we observe that the E2E latency remains relatively constant at $\approx 10s$ as the target bitrate varies. For bitrates under 1500 Kbits/s, the viewer never experiences any rebuffers. At 1500 and 2000 Kbits/s, we observe 20 and 30 rebuffering events, respectively, and a mean rebuffering time of 0.782 and 0.822 ms, respectively. There's positive correlation between the rebuffer rate and the upload bitrate when the bitrate exceeds the available bandwidth.

These rebuffers are short, typically lasting less that one second, but occur often. This leads to a very choppy live stream, which is in contrast to regular streaming experiences where apps can buffer for a while before resuming playback.

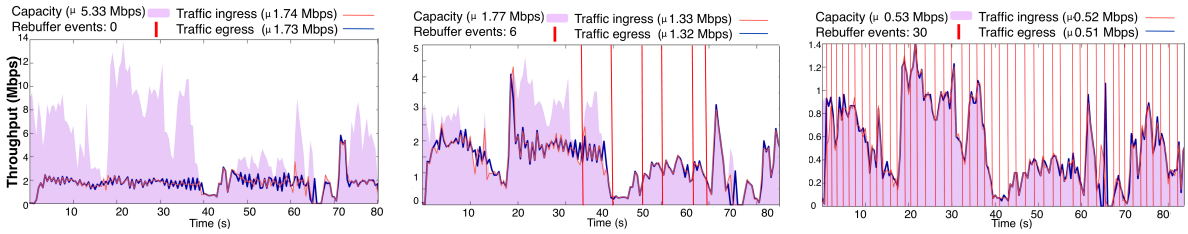
4.3 Varying mobile uplink capacity with fixed bitrate

As expected, the latency is far less stable due to the spiky nature of the cell trace. Figures 2 and 3 illustrate a negative correlation between available capacity and latency, most pronounced in figures 2c and 3c, and a positive correlation between capacity and rebuffer rate (red vertical lines indicate rebuffer events). Figures 3a



(a) Scale-down factor: 1 (b) Scale-down factor: 3 (c) Scale-down factor: 10

Figure 2: Correlation between uploader's capacity on cell trace and E2E latency



(a) Scale-down factor: 1 (b) Scale-down factor: 3 (c) Scale-down factor: 10

Figure 3: Correlation between uploader's capacity on cell trace and playback continuity

and 3b show that Facebook doesn't use all the available bandwidth. Notice that even when we hit a target capacity of 1Mbps in figure 3b, there is so much traffic queued up that we still incur rebuffer events. As shown in figure 4, the upload bitrate is almost twice as high the download bitrate. This means Facebook is streaming down lower quality videos than it receives. This is a waste of the uploader's bandwidth.

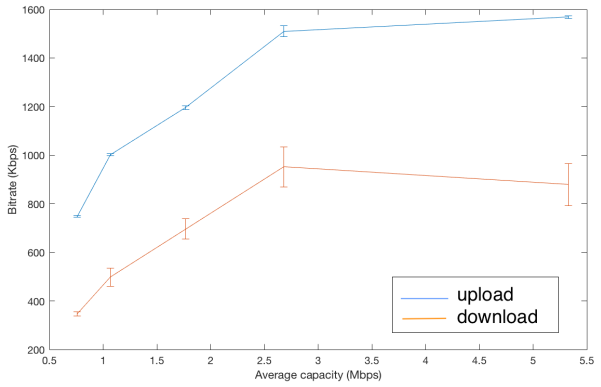


Figure 4: Upload and download bitrates

5. SUMMARY AND FUTURE WORK

We studied the effect of upload parameters on live video streaming using Facebook Live. The results of our investigation are threefold:

1. We establish relationships between upload parameters (bitrate and bandwidth) and important download metrics. We show an expected negative correlation between upload bandwidth and both delay and rebuffer rate. Surprisingly though, we show that the relationship between delay and rebuffer rate is not intuitive.

2. We show that live streaming and typical video streaming platforms should be optimized differently. Rebuffers in a live streaming system are shorter but more frequent.

3. We demonstrate shortcomings in Facebook's current implementation. The difference between upload and download bitrates suggest that Facebook is unnecessarily streaming lower quality video to its users. Additionally, the interaction of RTMP with variable cellular bandwidth produces poor streaming experiences for users using crowded mobile networks.

The next logical step of this work would be to implement a custom live-streaming system and test out both a) adaptive upload bitrate selection to mitigate rebuffer rates and b) new upload transport protocols to maintain high efficiency over cellular links.

Acknowledgements

We thank Ravi Netravali for his help brainstorming in the early stages of this research.

6. REFERENCES

[1] M. Siekkinin, E. Masala, and T. Kämäräinen. "A First Look at Quality of Mobile Live Streaming Experience: the Case of Periscope". In *Proceedings of IMC 2016*. pp. 477–483.
[2] SeleniumHQ: Browser Automation. `http://www.seleniumhq.org/`.
[3] PyVirtualDisplay. `https://github.com/ponty/pyvirtualdisplay`.
[4] Google Chrome. `https://www.google.com/chrome/`.
[5] Facebook Live: Live Video Streaming. `https://live.fb.com/`
[6] Live Videos from Publishing Tools. `https://www.facebook.com/facebookmedia/get-started/live`.
[7] Real-Time Messaging Protocol (RTMP) Specification. `https://www.adobe.com/devnet/rtmp.html`.
[8] FFmpeg. `https://ffmpeg.org/`
[9] Facebook for Developers: The Graph API. `https://developers.facebook.com/docs/graph-api`.
[10] Periscope. `https://www.periscope.tv/`
[11] Mahimahi. `http://mahimahi.mit.edu/`
[12] Wireshark. `https://www.wireshark.org/`
[13] Tesseract. `https://github.com/tesseract-ocr`
[14] Facebook Live Video Publishing tools. `https://www.facebook.com/facebookmedia/get-started/live`
[15] A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP `http://dl.acm.org/citation.cfm?id=2787486`