

Edge-based Transcoding for Adaptive Live Video Streaming

Pradeep Dogga*, Sandip Chakraborty†, Subrata Mitra‡, Ravi Netravali*

*UCLA, †IIT Kharagpur, ‡Adobe Research

Abstract

User-generated video content is imposing an increasing burden on live video service architectures such as Facebook Live. These services are responsible for ingesting large amounts of video, transcoding that video into different quality levels (i.e., bitrates), and adaptively streaming it to viewers. These tasks are expensive, both computationally and network-wise, often forcing service providers to sacrifice the “liveness” of delivered video. Given the steady increases in smartphone bandwidth and energy resources, coupled with the fact that commodity smartphones now include hardware-accelerated codecs, we propose that live video services augment their existing infrastructure with edge support for transcoding and transmission. We present measurements to motivate the feasibility of incorporating such edge-support into the live video ecosystem, present the design of a peer-to-peer adaptive live video streaming system, and discuss directions for future work to realize this vision in practice.

1 Introduction

Recent years have witnessed a rapid increase in the amount of live video content generated by user smartphones [14, 33]. Indeed, the prevalence of high quality video cameras on commodity smartphones, coupled with the rise of *live video* services like Periscope [32], Houseparty [4], Facebook Live [2], and YouTube Live [5], have made it easy for users to share real-time video streams with viewers. Through these services, users can share personalized versions of globally distributed events such as sports events, political debates, concerts, and more.

Figure 1 shows how existing live video services operate. Users wishing to share videos upload their streams to a transcoding server, typically hosted on a CDN. The server continually transcodes the incoming video into multiple bitrates (i.e., quality levels) and distributes the transcoded video chunks to edge caches. Finally, the edge caches serve that video to subscribed viewers via adaptive bitrate (ABR) streaming algorithms [24, 34] that aim to maximize streaming quality given environment conditions.

The highest order goals of live video streaming systems are to ensure that videos are available and as live as possible (delays around 100 ms) when delivered to viewers [17]. In other words, these systems aim to minimize the end-to-end delay between when a video frame is captured by a user’s smartphone, and when it is displayed to a viewer of the live stream. Unfortunately, several traffic patterns of the live video ecosystem complicate

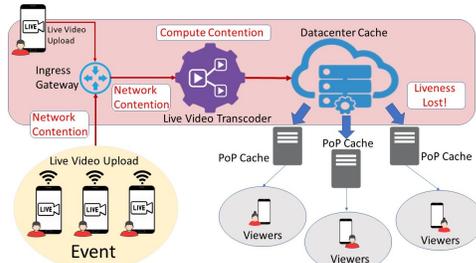


Figure 1: The existing live video architecture.

achieving these goals [6, 14, 33]. First, live video traffic is bursty, with videos rapidly growing and dropping in viewer popularity. Spikes in video popularity often result in the “thundering herd problem,” as millions of viewers may simultaneously subscribe to a video feed, causing delays and disconnection. Second, important events often generate multiple live streams from different users. For example, hundreds or thousands of individuals may simultaneously upload live video from a popular concert.

Collectively, these properties place a significant burden on live video infrastructure. In order to achieve low viewing latencies, transcoding servers (the first point of contact for a video uploader) must 1) maintain high ingress network bandwidth to support the abundance of incoming video streams, 2) have sufficient computational resources to transcode each incoming video stream into numerous bitrates, and 3) have access to sufficient egress network bandwidth to populate edge caches. While state-of-the-art live video services employ optimizations such as using low-latency streaming protocols and AI-based parallel transcoding, end-to-end delays continue to be negatively affected by high video loads [10, 15]. As a result, service providers must often resort to video buffering at sources to ease transcoding overheads at the expense of liveness.

In this paper, we propose that existing live video systems be augmented with edge-based video transcoding and peer-to-peer (P2P) video transmission. Rather than having each uploaded video stream pass through a transcoding server and edge cache prior to reaching its viewing destination, we propose that certain video streams be directly transcoded and distributed by viewer smartphones.¹ However, unlike prior P2P video systems [7, 8, 13, 30, 37], we aim to preserve *liveness and adaptive video streaming* so viewers can download video at the bitrates best suited for their networks and devices. To realize this vision, we seek to leverage prior work on multicast distribution trees, but extend it to support live

¹We couple transcoding and transmission because transcoding alone would increase bandwidth overheads (peers would have to upload every video version) and harm liveness (adding roundtrips to transcode).

videos and edge-based (i.e., phone-based) transcoding (§3). Using our proposed protocols, video sources and viewers dynamically arrange themselves into distributed, balanced trees to transcode and distribute video while adhering to device resource constraints (bandwidth and energy), ensuring fairness, and maximizing liveness.

Our proposal is motivated by two main observations. First, there have been steady increases in client-side bandwidth (uplink and downlink) [27] and energy resources [16]. Moreover, smartphone usage is increasingly WiFi-based, without strict data caps [1]. Second, commodity smartphones, a key player in the live video ecosystem [18], now include hardware-accelerated video codecs for fast and low-energy transcoding. Our results (§2) show that these edge resources have been *largely ignored* by live video systems even though transcoding and transmitting on phones only add 0.5% and 5% energy overheads, respectively, beyond viewing video (which smartphones must already do). Additionally, smartphone hardware codecs can easily support the 30-60 frames per second (FPS) that live video systems use. Thus, we believe that video producers and consumers are well-suited to take on increasing roles in live video streaming systems.

Introducing edge-based transcoding and P2P transmission into live video systems introduces many challenges. First, our system must balance resource utilization at peers with the goal of maximizing video liveness. Performing this online optimization is difficult given the flux in viewers of a given video stream. Second, our system must be robust to handling peer mobility and failures: viewers may appear and disappear at any time, but the video service must remain available. Third, how can viewers be incentivized to contribute resources to this P2P service? Fourth, how can the system be protected against malicious peers, e.g., those who tamper with video content? This paper primarily focuses on the first two challenges (§3), but we describe how prior incentivization [9, 29, 31] and data integrity [11, 36] strategies can be applied to this setting.

In summary, this paper is a call for live video services to incorporate edge-based transcoding resources into their pipelines. Though there still exist challenges to enabling performant and secure P2P live video transmission (§4), we believe that the potential load reductions on live video infrastructure could be significant.

2 Motivation

In this section, we present preliminary measurements to motivate the feasibility of edge-based live video transcoding. Our results highlight three key points:

1. Existing live video systems fail to incorporate edge-based computation to reduce backend overheads.
2. Transcoding and transmitting on smartphones consume little energy beyond that used to view video.

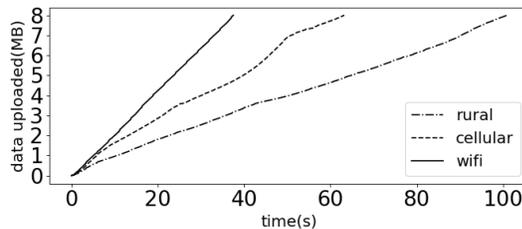


Figure 2: Bytes sent over time for Facebook Live uploads of the same video over three different network conditions.

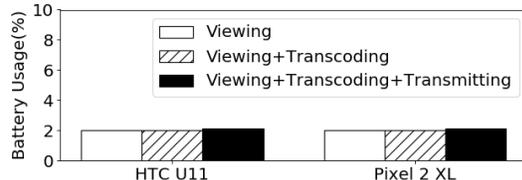


Figure 3: Battery expenditure for client tasks in live video systems, with and without edge-based transcoding.

3. Transcoding on smartphones is fast (30-60 FPS).

Existing live video systems fail to leverage edge computing. To better understand a video uploader’s role in today’s live video systems, we analyzed the network traffic generated by Facebook Live video uploads over a variety of configurations. Our experiments used the Mahimahi network emulator [26], and considered three different network conditions: cellular (Verizon LTE trace with 80 ms RTT), residential WiFi (24 Mbps with 20 ms RTT), and a slow rural or crowded network (1 Mbps with 80 ms RTT). On each network, we accessed Facebook Live via Google Chrome (v71) on a Dell XPS 13 laptop, and uploaded a nearly-identical video of an empty office.

Figure 2 illustrates that across these network conditions and video sessions, Facebook Live transmitted a nearly identical number of bytes (≈ 8 MB). As expected, such fixed transmission resulted in different lags across the tested network conditions. For instance, the cellular network experienced a lag of (≈ 60 s) for the final frame in the video, while the rural network’s lag reach (≈ 100 s) for the same frame. These results demonstrate that Facebook Live uploaders perform minimal (if any) edge-based optimizations to uploaded video.

Phone-based transcoding and transmission imposes low energy overheads. We sought to evaluate the energy overheads of mobile phones performing 1) local transcoding with hardware accelerators, and 2) video transmission. Our experiments used two Android phones: a Google Pixel 2 XL and an HTC U11, both with hardware-accelerated codecs. We streamed several versions of an HD video (10 minutes long) from one phone to the other: 1080p, 720p, and 480p. All transcoding was done on the fly using the devices’ hardware codecs, accessed via Android’s MediaCodec infrastructure. Device energy usage was measured with the Android BatteryManager.

Figure 3 reports the battery drain on each phone

	1080p → 720p	720p → 360p	1080p → 360p
HTC U11 (hardware)	27.65s (15ms)	12.89s (7ms)	17.14s (9ms)
Pixel 2 XL (hardware)	14.41s (8ms)	10.53s (6ms)	16.05s (9ms)
Laptop (software)	11.33s (6ms)	3.873s (2ms)	4.81s (2ms)

Table 1: **Transcoding time for an entire 1 minute HD video (and per frame) using hardware-accelerated transcoders and modern software transcoding (H264 with FFMPEG).**

when performing different combinations of transmitting, transcoding, and downloading video. As shown, transcoding and transmission add minimal battery overhead beyond the energy used to view video. For example, with the HTC U11, transcoding consumes an additional 5%, while transcoding and transmitting has a 10% overhead beyond viewing. The reason is that transcoding can leverage the video processing that is already being done to capture or view video. In particular, transcoding requires decoding video into raw frames, and then encoding those frames into new decoded frames at a different bitrate. Since viewing an incoming video requires a phone to decode incoming frames before rendering, transcoding only adds an additional hardware-accelerated encoding step. Similar lightweight encoding can be performed directly on the raw frames captured by a video uploader.

Phone-based transcoding is fast enough for live streaming. Edge-based transcoding in live video systems must not add additional delays to the end-to-end streaming process; otherwise, video may as well go directly to congested transcoding servers. To evaluate the latency overheads of edge-based transcoding, we transcoded a 1-minute video from 1080p to 720p and 480p. For reference, we also performed the same transcoding using a modern H264 software transcoder, FFMPEG [3]. Table 1 shows that transcoding the entire video takes no more than 27s for the hardware-based accelerators (6-15 ms per frame). To provide context for these delays, we connected two phones via USB cable, and streamed video at 30 and 60 FPS. In both cases, the received frame rate was not affected by the hardware-based transcoding on the sender.

3 Proposed Design

3.1 Overview

Figure 4 provides an overview of our proposed architecture. We classify participants into three primary groups:

- a **video source** who captures video using their mobile device, transcodes it to the highest requested bitrate, and transmits it to a peer for that bitrate,
- a **leader** who downloads video, transcodes it to the bitrate that it requires, sends that transcoded video to other peers viewing at the same bitrate and to another leader for subsequent transcoding,

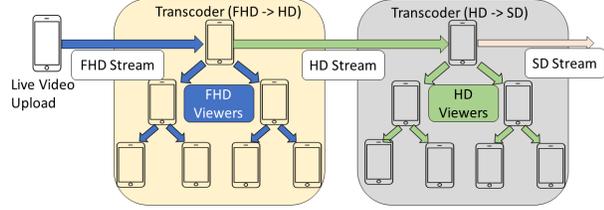


Figure 4: **Overview of our proposed P2P live video system.**

- and a **follower** who downloads video sent from a leader or another follower.

Inspired by multicast trees [25], we model each collection of viewers demanding a particular bitrate as a distributed balanced tree. All such trees have links to the root of the tree pertaining to the next highest bitrate; the tree pertaining to the highest bitrate being streamed is directly connected to the video source. To distribute the task of keeping track of tree structures, each node in a tree has knowledge of its immediate descendants and the depth of the subtree rooted by it. Further, all trees have a leader node who is the first to receive each video frame at the tree’s bitrate. Each leader is aware of the leader in the tree for the next lowest bitrate, and the source is aware of the leader of the tree for the highest viewed bitrate. The leader transcodes video frames that it receives to the desired bitrate, and distributes this transcoded video to its immediate children (i.e., followers) in the tree, which continually propagate the frames down the tree until they reach the tree’s leaves. The leader also sends transcoded frames to the leader of the tree for the next lowest bitrate.

3.2 Balancing Trees

The distribution trees for a given stream must be rebalanced when nodes enter or leave, or periodically for fairness. Rebalancing should balance device resource constraints (bandwidth, energy), fairness, liveness of delivered video, and control overhead (i.e., frequent tree restructuring). In particular, given a current topology X for a distribution tree, network latency estimates between every pair of nodes, and a preferred bitrate $\mathbf{BR}_{\text{pref}}(x)$, energy constraints $\mathbf{E}(x)$, and bandwidth constraints $\mathbf{B}(x)$ for each node $x \in X$, our goal is to optimize the following:

$$\begin{aligned}
 & \underset{x_i, x_j \in X}{\text{minimize}} && \sum |\mathbf{E}(x_i) - \mathbf{E}(x_j)| + |\mathbf{L}(x_i)| + |\mathbf{B}(x_i) - \mathbf{B}(x_j)| \\
 & && + |X' - X| \\
 & \text{subject to} && 0 \leq \mathbf{E}(x_i) \leq \mathbf{E}_{\text{available}(i)}, \\
 & && 0 \leq \mathbf{B}(x_i) \leq \mathbf{B}_{\text{max}(i)}, \\
 & && \mathbf{BR}(x_i) \leq \mathbf{BR}_{\text{pref}}(x_i)
 \end{aligned}$$

where $L()$ represents the liveness of video for a given node (based on computation time and P2P network latencies and transmission delays), and $X' - X$ represents the graphical difference between the prior topology and the one

being generated. We note that this optimization assumes global knowledge of the streaming ecosystem, and can be performed by a single rotating node that aggregates information from other nodes in its tree. A distributed version, which would alleviate the potential optimization bottleneck of aggregating tree information, will likely have to rely on heuristics, e.g., optimizing only after the viewer base has sufficiently changed, with intermediate modifications made using local tree balancing. In addition, in the event that expected P2P transmission times are unacceptably high, the live video service can revert to a centralized approach or a hybrid version, where only peers in close proximity exchange video to reduce infrastructural load.

3.3 Operational Details

Registering new nodes. When a new viewer wishes to join a live video stream, it first registers with the corresponding video source. If the new viewer is the first in the stream, or if it wants a higher bitrate than any other existing viewer (and thus, tree), a new tree is established for the requested bitrate and the new viewer is appointed the tree leader. The new viewer gets video directly streamed from the video source at the requested bitrate. If other trees exist at lower bitrates, the new viewer must inform the leader of the next highest bitrate tree (whose IP address is provided by the source) of its existence.

If viewers exist at higher or the same bitrate as the new viewer, then the new viewer must follow a protocol to determine where it belongs in the distribution ecosystem. The source will first inform the new viewer of the IP address of the leader for the highest bitrate tree. If the new viewer's preferred bitrate matches that tree's, then the new viewer joins the tree; the viewer's exact placement in tree is determined by executing the optimization described above. Otherwise, if the new viewer's preferred bitrate is lower than that leader's, the leader provides the new viewer with the IP of the leader in the next highest bitrate's tree. This process repeats until the new viewer finds the appropriate tree, or determines that it must start its own; if a new tree is made, adjacent leaders must be notified.

Handling departing of failing nodes. Given the semantics of live video streaming traffic, mobile phones can be expected to leave a stream at any point. Thus, we need a failure recovery mechanism to quickly rebalance the video distribution tree and minimize streaming disruptions. We consider three possible scenarios. If the video source fails, the leader of the highest bitrate's tree attempts to reconnect until a timeout is reached; on a timeout, the stream is deemed temporarily canceled, and this is propagated to all trees and nodes. If a follower A who streams video to a follower B fails, follower B then informs its followers that a recovery process is underway (this propagates down to the leaves), and contacts the source to follow the

same process as described above for a newly joining node. Finally, if a leader for a subtree fails, one of its direct children is appointed the temporary leader, and the subgraph rooted at the right of that child follows the follower failure protocol to reoptimize the tree. Coordination with leaders of adjacent trees is done by the source. Across all of these scenarios, failure detection can be performed in a distributed manner as each node is aware of its immediate neighbors in the tree. Thus, standard mechanisms such as heartbeat messages and timeouts can be used; timeout values would dictate how quickly failures are detected.

Swapping leaders. Unlike followers who simply transmit received video to their immediate descendents, leaders must also transcode video *and* transmit it to the neighboring tree. The additional transmission overheads can be alleviated by altering the tree structure such that the leader only has a single child, which then has two children; in this way, all nodes transmit to at most two other nodes. Further, though transcoding consumes minimal additional energy resources (§3), it is unfair to have the same single node always bear the transcoding burden for an entire tree. Thus, we may have to periodically reoptimize the tree.

4 Future Work and Discussion

4.1 Challenges

Peer Presence. With any P2P system, a big challenge is ensuring sufficient peer presence to scale and meet QoE goals. Though the overheads that our proposal imposes on peers are low (§2), and tit-for-tat strategies can incentivize participants, we cannot guarantee that all viewers would make good peers. P2P live streaming is only feasible when smartphones are connected to low-latency WiFi or unlimited cellular plans. Thus, additional work must be done to ensure (via real-time decisions) that P2P delivery is only used when the available peers can operate within the desired latency bounds.

Viewer Flux. Live video streams often experience significant flux in viewer numbers. Thus, viewer devices may appear or disappear at any time during a video stream. However, our solutions to maximize QoE given peer resource constraints from Section 3 require us to dynamically rebalance transmission trees each time a device appears or disappears (and periodically for fairness). Intelligent distributed protocols are required to ensure that each optimization converges to a balanced tree.

Storing Video. Live video services may wish to permanently store copies of uploaded video. In these cases, P2P support cannot entirely eliminate the bandwidth overheads of the CDN as video must eventually be uploaded. However, by having peers transcode and deliver video during real-time viewing, live video systems earn slack

as to when video must be uploaded. For example, video can be buffered on peers until bandwidth contention has subsided at the CDN. Further, transcoding can be done at the edge or the CDN. The former trades off bandwidth for compute resources, while the latter does the opposite.

Alternate Bottlenecks. Our proposal aims to alleviate the high bandwidth and computation demands that live video infrastructures face. However, in certain scenarios, edge network bandwidth may be the primary bottleneck in end-to-end live video transmission, e.g., at events with numerous uploaders and limited access links. We note that, in these cases, P2P video transmission will not provide benefits unless all viewers are colocated with the uploader such that transmissions over the access link are reduced.

Data Integrity. How can viewers verify that video downloaded from peers is semantically equivalent to that generated by the source? At first glance, standard integrity techniques may seem sufficient. However, each has limitations. Having peers perform remote attestation on frames (even probabilistically) will impose significant computational overheads, harming liveness. Comparing per-frame hashes is insufficient because video may have been transcoded multiple times from the video captured at the source, making hash comparisons uninformative. A viewer could randomly request frames from the source, and transcode locally to perform a comparison with frames received from peers. However, this may not generalize as transcoders across phones need not be identical.

Seeks and Delayed Viewing. Viewers can cache all frames during the duration of a stream. However, for long streams, this may not be possible due to storage limitations. In the event that a viewer wishes to view an old frame which is no longer locally cached, where should they go? Relatedly, viewers may watch a live video stream after the fact. Such requests should be directed to standard infrastructure (i.e., CDNs) as it mimics classic video streaming settings. But how should viewers populate those CDNs? A scheme must balance upload fairness with limiting load on existing infrastructure.

4.2 Other Application Use Cases

Real-time Video Analytics. In these systems, cameras upload live video to cloud clusters, which run machine learning pipelines to recognize objects and answer video-specific queries (e.g., is there traffic at the intersection?). Since classification is costly, much work has focused on tuning pipeline knobs (e.g., video resolution) to balance query accuracy and resource consumption [19, 35].

Edge-based transcoding could be used to reduce the bandwidth costs associated with uploading live video to analytics clusters. To realize this, we ask whether the edge (i.e., cameras) can be made aware of the query

knobs that are being used in the cloud cluster? Cameras, which are also increasingly equipped with hardware-based transcoders, could downgrade captured video to the minimal bitrate necessary for the cloud-based query execution, thereby saving upload bandwidth. Prior work has explored the potential bandwidth savings [28], but understanding how best to incorporate edge-based transcoding into this pipeline in a real-time manner remains an open question.

Private Live Video. Today, live video uploads must go through a third party service like Facebook Live or Periscope. These services often cache these videos on edge servers to disseminate them to viewers (§1). However, consider a scenario where a user wants to distribute private live video (e.g., an internal company conference) to only select viewers (e.g., employees). Edge-based transcoding and P2P transmission could support such a scenario without sharing private content with a third party.

5 Related Work

P2P video streaming: The key differences between our proposal and prior P2P content delivery approaches [7, 8, 13, 30, 37] are 1) we target smartphone video generation and consumption, which pose strict restrictions on network bandwidth and energy resources (§3), and 2) we aim to preserve the adaptive bitrate streaming that live video services currently offer.

Edge-based support for video streaming: With Crowd-Transcoding [12], video sources ship video to datacenter machines, which offload video transcoding to video viewers (and retrieve the transcoded versions). However, this does not alleviate the network contention associated with uploading live video to datacenters; in contrast, our proposal targets both server-side transcoding costs *and* server network contention. A separate line of work has proposed peer-based transcoding for P2P video delivery systems. For instance, rather than having video owners transcode video into all possible bitrates, PAT [21, 22] requires each viewer to transcode the video that they watch using transcoding metadata [20] shipped via an overlay network. In contrast, our proposal ensures that a video is globally transcoded at most once to each required bitrate.

Reducing server-side transcoding overheads: Several systems have enabled more efficient and cheaper transcoding for video servers. VideoCoreCluster [23] showed how high-scale video transcoding can be done in a cost-effective way using clusters of Raspberry Pi devices. More recently, ExCamera [10] described how encoding could adopt fine-grained parallelism using Lambda function services, cutting processing times by an order of magnitude. Our proposal is orthogonal to these techniques in that we are adding yet another knob to alleviate infrastructural overheads during bursty transmission periods.

References

- [1] Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022 white paper. <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html>.
- [2] Facebook live: Live video streaming. <https://live.fb.com/>.
- [3] Ffmpeg. <https://www.ffmpeg.org/>.
- [4] Houseparty: Face to face social network. <https://houseparty.com/>.
- [5] Live - youtube. <https://www.youtube.com/channel/UC4R8DWoMoI7CAwX8LjQHig>.
- [6] Under the hood broadcasting live video to millions at facebook. <https://code.fb.com/networking-traffic/under-the-hood-broadcasting-live-video-to-millions/>.
- [7] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03*, pages 298–313, New York, NY, USA, 2003. ACM.
- [8] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8):1489–1499, 2002.
- [9] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.
- [10] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein. Encoding, Fast and Slow: Low-latency Video Processing Using Thousands of Tiny Threads. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, NSDI'17*, pages 363–376. USENIX Association, 2017.
- [11] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang. A tree-based forward digest protocol to verify data integrity in distributed media streaming. *IEEE Trans. on Knowl. and Data Eng.*, 17(7):1010–1014, July 2005.
- [12] Q. He, C. Zhang, and J. Liu. Utilizing massive viewers for video transcoding in crowdsourced live streaming. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 116–123. IEEE, 2016.
- [13] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. Promise: peer-to-peer media streaming using collectcast. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 45–54. ACM, 2003.
- [14] High Scalability. How Facebook Live Streams To 800,000 Simultaneous Viewers. <http://highscalability.com/blog/2016/6/27/how-facebook-live-streams-to-800000-simultaneous-viewers.html>, 2016.
- [15] InfoQ. Sachin Kulkarni Describes the Architecture behind Facebook Live. <https://www.infoq.com/podcasts/sachin-kulkarni-facebook-live>, 2017.
- [16] InfoQ. Fact check: Is smartphone battery capacity growing or staying the same? <https://www.androidauthority.com/smartphone-battery-capacity-887305/>, 2018.
- [17] InfoQ. Handling Traffic Spikes from Global Events at Facebook Live. <https://www.infoq.com/news/2018/02/facebook-live-scaling>, 2018.
- [18] InfoQ. 62 Must-Know Live Video Statistics. <https://livestream.com/blog/62-must-know-stats-live-video-streaming>, 2019.
- [19] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18*, pages 253–266. ACM, 2018.
- [20] D. Liu, S. Chen, and B. Shen. Amtrac: Adaptive meta-caching for transcoding. In *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, page 6. ACM, 2006.
- [21] D. Liu, F. Li, B. Shen, and S. Chen. Building an efficient transcoding overlay for p2p streaming to heterogeneous devices. *ACM Trans. Multimedia Comput. Commun. Appl.*, 8(1S):10:1–10:22, Feb. 2012.

- [22] D. Liu, E. Setton, B. Shen, and S. Chen. Pat: Peer-assisted transcoding for overlay streaming to heterogeneous devices. In *Proc. Int. Workshop Netw. Oper. Syst. Support Dig. Audio Video*, 2007.
- [23] P. Liu, J. Yoon, L. Johnson, and S. Banerjee. Greening the video transcoding service with low-cost hardware transcoders. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, pages 407–419, Berkeley, CA, USA, 2016. USENIX Association.
- [24] H. Mao, R. Netravali, and M. Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, New York, NY, USA, 2017. ACM.
- [25] J. D. Mol, D. H. Epema, and H. J. Sips. The orchard algorithm: Building multicast trees for p2p video multicasting without free-riding. *IEEE Transactions on Multimedia*, 9(8):1593–1604, 2007.
- [26] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-replay for HTTP. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pages 417–429. USENIX Association, 2015.
- [27] Ookla. Speed test. <http://www.speedtest.net/reports/united-states/#fixed>. Sep 7, 2017.
- [28] C. Pakha, A. Chowdhery, and J. Jiang. Reinventing Video Streaming for Distributed Vision Analytics. In *Proceedings of the 10th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'18, Berkeley, CA, USA, 2018. USENIX Association.
- [29] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bit torrent. In *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.
- [30] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, pages 329–350, London, UK, UK, 2001. Springer-Verlag.
- [31] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki. Dandelion: Cooperative content distribution with robust incentives. In *Proceedings of the 2007 USENIX Annual Technical Conference*, ATC'07, Berkeley, CA, USA, 2007. USENIX Association.
- [32] 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*, CHI '16, pages 4770–4780, New York, NY, USA, 2016. ACM.
- [33] B. Wang, X. Zhang, G. Wang, H. Zheng, and B. Y. Zhao. Anatomy of a personalized livestreaming system. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 485–498. ACM, 2016.
- [34] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 325–338, New York, NY, USA, 2015. ACM.
- [35] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live Video Analytics at Scale with Approximation and Delay-tolerance. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 377–392. USENIX Association, 2017.
- [36] X. Zhang, S. Chen, and R. Sandhu. Enhancing data authenticity and integrity in p2p systems. *IEEE Internet computing*, 9(6):42–49, 2005.
- [37] X. Zhang, J. Liu, B. Li, and Y.-S. Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM*. IEEE, 2005.