Fog-Based Transcoding for Crowdsourced Video Livecast

Qiyun He, Cong Zhang, Xiaoqiang Ma, and Jiangchuan Liu

Recent years have witnessed the booming popularity of CLS platforms, through which numerous amateur broadcasters live stream their video contents to viewers around the world. The heterogeneous qualities and formats of the source streams, however, require massive computational resources to transcode them into multiple industrial standard quality versions to serve viewers with distinct configurations, and the delays to the viewers of different locations should be well synchronized to support community interactions.

Recent years have witnessed the booming popularity of CLS platforms, through which numerous amateur broadcasters live stream their video contents to viewers around the world. The heterogeneous qualities and formats of the source streams, however, require massive computational resources to transcode them into multiple industrial standard quality versions to serve viewers with distinct configurations, and the delays to the viewers of different locations should be well synchronized to support community interactions.

ABSTRACT

Recent years have witnessed the booming popularity of CLS platforms, through which numerous amateur broadcasters live stream their video contents to viewers around the world. The heterogeneous qualities and formats of the source streams, however, require massive computational resources to transcode them into multiple industrial standard quality versions to serve viewers with distinct configurations, and the delays to the viewers of different locations should be well synchronized to support community interactions. This article attempts to address these challenges and to explore the opportunities with new generation computation paradigms, in particular, fog computing. We present a novel fog-based transcoding framework for CLS platforms to offload the transcoding workload to the network edge (i.e., the massive number of viewers). We evaluate our design through our PlanetLab-based experiment and real-world viewer transcoding experiment.

INTRODUCTION

Recent years have witnessed the emergence of the crowdsourced livecast service (CLS) platforms, represented by Twitch TV (https://www.twitch.tv), YouTube Live (https://www.youtube.com/live), and so on. Given the rapid development of personal computing devices (e.g., smartphones) and broadband network access, most video sources in CLS come from amateur broadcasters rather than commercial/professional content providers, which remarkably stimulates content diversity. This new generation of livecast service has already achieved tremendous success. According to Twitch Retrospective 2015 (https://www.twitch.tv/year/2015), Twitch TV had 35,610 concurrent broadcasters and more than 2 million concurrent viewers during peak time in 2015, with the total stream time exceeding 241 billion minutes through the whole year.

While globally gaming is the major topic on Twitch TV and other mainstream platforms, we have also witnessed the prevalence of regional diversities. For example, the Chinese market is dominated by Douyu TV (https://www.douyu.com), Panda TV (http://www.panda.tv), and Inke (https://www.inke.cn), with a wide range of livecast contents such as singing, talk shows, and even traveling livecast.

Similar to traditional YouTube-like video streaming, the streams in CLS platforms normally have various qualities and formats to serve viewers with diverse network conditions. However, such heterogeneity in source content is more dramatic. For example, as we measured on Twitch TV, the live video contents are generated at over 150 different resolutions, which clearly demands unifying the source streams into industrial standard quality versions. Video on demand (VoD) providers such as YouTube and Netflix have widely adopted adaptive bit rate (ABR) [1] streaming and dynamic adaptive streaming over HTTP (DASH) [2], where the video contents are sliced and transcoded into multiple quality versions, which are finally served to distinct viewers based on their individual requirements. Unlike the VoD scenario, real-time video transcoding for live streaming requires a huge amount of concurrent computational resources, which can be expensive with dedicated servers and even the cloud. As a trade-off between cost and quality of service (QoS), popular CLS platforms such as Twitch TV only offer transcoding services to a small number of premium broadcasters, which make up only 1 to 1.5 percent of all broadcasters.

CLS platforms also involve a great number of viewers who constantly interact with broadcasters and fellow viewers. They have shown great willingness to support the platform and broadcasters, through forms such as donation and monthly subscription at a certain fee. Thanks to the rapid advancement in hardware and software, their home computers nowadays are powerful enough to transcode while playing high-quality live streaming simultaneously. As we observed, the viewer base in major CLS systems is always much larger than the channel base (the number of total channels) at any moment, indicating the existence of a huge amount of potential computational resources.

In this article, we seek novel solutions to offload the massive transcoding workloads in CLS systems to the edge of the network (i.e., the viewers). By analyzing the captured data and systematically studying the user behavior dynamics, we put forward a novel fog-based transcoding framework that crowdsourced the computational resources from viewers and smartly schedule stable viewers for real-time video transcoding. We also address the critical issue of cross-viewer synchronization so as to enhance community interactions.

Digital Object Identifier:
10.1109/MCOM.2017.1600826

The authors are with Simon Fraser University. Xiaoqiang Ma is also with Huazhong University of Science and Technology.

0163-6804/17/$25.00 © 2017 IEEE

Crowdsourced Livecast: Challenges and Opportunities

We first briefly introduce the background and related research on crowdsourced livecast. We then illustrate the unique features, challenges, and opportunities of this new generation of video service through our measurements.

The form of crowdsourced livecast started to get popular in 2012, with some leading platforms guickly expanding in the market. This new generation of user-generated video streaming has also attracted much attention from academia. First, in terms of the social component of these platforms, Kaytoue *et al*. [3] proposed the first characterization of the emerging online Web-based community on Twitch TV, and Hamilton *et al*. [4] further studied its emergence, socialization, and participation. Second, regarding the video streaming performance and user experience, Zhang *et al*. [5] explored the architecture of Twitch TV and investigated the impact of interaction delay on user experience. Aparicio-Pardo *et al*. [6] dug into the Twitch datasets and presented an optimal model for streaming quality to improve viewers' satisfaction. Third, in the field of system architecture, Chen *et al*. [7] proposed a generic cloud renting strategy to optimize the cloud site allocation for video transcoding and delivery in CLS platforms. He et al. [8] further put forward a cloud-based solution that jointly considers user satisfaction and service availability/pricing for video transcoding CLS platforms. However, given the considerable cost of deploying cloud servers and the fact that the CLS platforms charge viewers nothing as a free system by nature, cloud-based transcoding solutions, which are currently adopted by Twitch TV, can only provide transcoding service to very popular streams. To better explore the characteristics and challenges in crowdsourced livecast, we have conducted three measurements.

First, it has been reported that community interaction plays an important role in crowdsourced livecast services. For example, the chat lines of all broadcast channels is found to constantly exceed 400 per second. (http://twitchstatus.com/index.html) Intuitively, it is desirable that fellow viewers are relatively synchronized such that the in-channel community interaction would not cause negative user experience. But according to our measurement, out-of-synchronization chats are not uncommon due to heterogeneous broadcast latency among fellow viewers. We set up two computers: one serves as the broadcaster to encode a source video at 1500 kb/s bit rate; the other hosts two identical virtual machines (VMs). We created a Twitch channel, and the two VMs watched the video stream as viewers. We first set the bandwidth limit to 2000 kb/s for both VMs. We then added the propagation delay of one VM (VM *A*) from 100 to 400 ms, while keeping the other VM (VM *B*) unchanged. The broadcast latency difference between the two viewers under different network conditions is shown in Fig. 1. We can see that, with added propagation delay, the broadcast latency difference becomes significantly larger. A broadcast latency difference of 20 s is almost intolerable for real-time interaction between the viewers. On the other hand, when we changed the bandwidth of VM *A* to

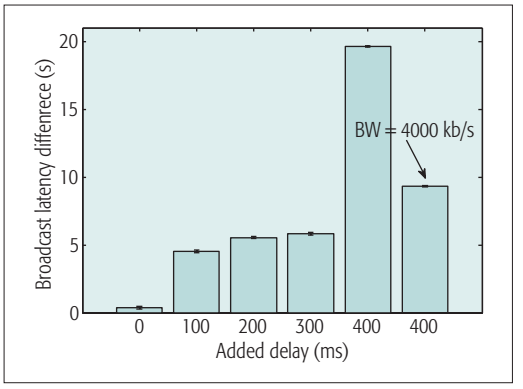

Figure 1. Broadcast delay difference under different network conditions.

4000 kb/s and kept the added propagation delay at 400 ms, the broadcast latency difference is dramatically reduced by half. Our first measurement reveals that divergent end-to-end delays can cause intolerable broadcast latency difference for fellow viewers. Such physical constraints as propagation delay and bandwidth limit, however, are not easy to be lifted for viewers, and we instead should seek for solutions within the broadcast platform.

In our second measurement, we focused on the channels/broadcasters. From February 2015 to June 2015, we captured the data of the broadcasters from Twitch TV every five minutes, using Twitch's public application programming interface (API) (http://dev.twitch.tv). Two outstanding characteristics attracted our attention. First, the resolution of source streams is highly varying over time. For example, at one moment we witnessed 177 different resolutions ranging from 116p to 1600p. Even for the source streams with the same resolution, there are very different bit rates. Second, at any moment the viewer base is dramatically larger than the channel base, and the result is even more remarkable if we only consider the top 10 percent of channels, which attract more than 98 percent of viewers.

The third measurement specifically targets viewer behavior. We captured viewers' online traces of five popular Twitch TV channels from January 25 to February 27, 2015. The measurement captured the JOIN message when any registered viewer joins the channel and the PART message when the viewer leaves the channel. In total, we collected 11,314,442 JOIN records and 11,334,140 PART records. We first see that the overall viewer online duration time can be closely fit to a scaled Pareto distribution function with $\alpha = 0.7$ and $x_m = 2$. Additionally, we can conclude that the longer a viewer is online, the more likely this viewer will continue to be online. The viewers' online duration behavior is also quite consistent, as we see around 80 percent of viewers have a standard deviation less than 20 min.

In summary, in such large CLS platforms as Twitch TV with a massive viewer base, a considerable portion of online viewers are potential resources for stable video transcoding. When exploring these resources, however, inherent challenges lie in the viewers' heterogeneity in terms of their stability and networking/system performance, which calls for effective strategies to distinguish viewers and appropriately make use of their resource. To support rich community inter-

Our first measurement reveals that divergent end-to-end delays can cause intolerable broadcast latency difference for fellow viewers. Such physical constraints as propagation delay and bandwidth limit however are not easy to be lifted for viewers, and we instead should seek for solutions within the broadcast platform.

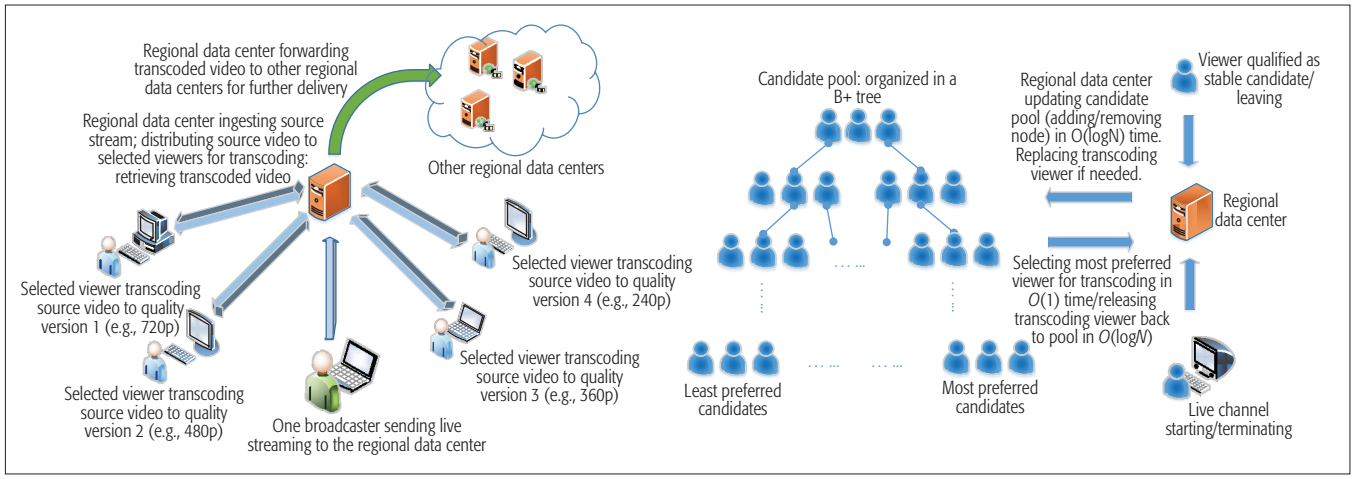

Figure 2. Fog-based transcoding framework: a) system overview; b) scheduling process.

actions, the delays to viewers at different locations should be not only minimized, but also well synchronized. We address these challenges by a novel solution inspired by fog computing [9].

Fog-Based Transcoding: Architectural View

We now illustrate the overall architecture of our design. At the top level, globally the system is divided into multiple regions. Each region is maintained by its own regional data center (also referred to as "regional server"), which is responsible for ingesting source videos from its region, assigning transcoding viewers, recollecting transcoded video, and forwarding the processed streams for further delivery (Fig. 2a). If the regional server cannot find enough transcoding viewers inside the same region, it forwards an assignment request to its neighbor regions, and such assignment is called *cross-region assignment*. If a transcoding viewer becomes offline prior to the termination of the assigned channel, another candidate needs to be scheduled to continue on the transcoding work, which is defined as *reassignment*. At the bottom level, each individual viewer/broadcaster has its own behavior, for example, online/offline time, which is not controlled by the system.

In terms of the scheduling process, our objective is to minimize the number of cross-region assignments and reassignments, as the former introduce extra streaming delay, while the latter cause additional system overhead for re-selecting candidates as well as short absence of the target quality version during reassignment. We therefore want the selected viewers to be as stable as possible, while having a candidate pool of reasonable size so that it will not trigger too many cross-region assignments. Note that we assume selected viewers are cooperative, while in real-world scenarios more incentive mechanisms (e.g., auction for crowdsource [10]) can be used. In the following three subsections, we illustrate how the system selects stable viewers, and then conducts the scheduling process.

Extracting Qualified Stable Candidates

Our observations above indicate that generally the stability of a viewer is proportional to the existing time he/she has already spent in the channel. We therefore set a *waiting threshold* $T(t)$, after passing which the viewer can be regarded as stable. Notably, a longer waiting threshold leads to more stable candidates, but also results in fewer qualified candidates and more time wasted for waiting. We therefore want to maximize the total accumulative transcoding time of stable viewers leaving before and after the channel terminates. To do that, we formulate the mathematical expression of the sum of these two time components and set its derivative to zero, such that the transcoding time is maximized. Given the viewer online duration distribution, the above mentioned scenario is achieved when the waiting threshold is around 30.4 percent of the remaining livecast time. In reality, due to the dynamics of viewers, this result can vary from 25 to 34 percent of the remaining time.

Estimating Individual Stability

Now that we have the optimal online time threshold to filter stable viewers in general, we also expect to estimate the stability of an individual viewer. Some heuristics, such as viewer age, gender, and video quality [11, 12], can be used to further estimate the individual stability in a fine-grained manner. Here we use a simple but effective heuristic that measures the average online duration and standard deviation of a viewer's online record. We use a linear combination of them to further estimate the individual stability. This heuristic particularly reflects the fact that a longer average online duration indicates the viewer tends to stay longer, and a smaller standard deviation means such behavior is more consistent.

Scheduling while Handling Viewer Dynamics

To minimize the reassignment count, we expect to choose the most stable candidate first. However, we are reluctant to change any assignment once made, unless either the directly related viewer or channel state has changed.

To maintain the candidate pool, we can simply re-rank all candidates at any update, but this takes $O(N)$ time every update and $O(1)$ time at assignment time. It incurs significant calculation when N is large with frequent updates. Since there are many more viewer updates than channel updates and reassignments, and re-ranking is only conducted at the latter, we can instead order candidates at the assignment time (taking $O(N$-

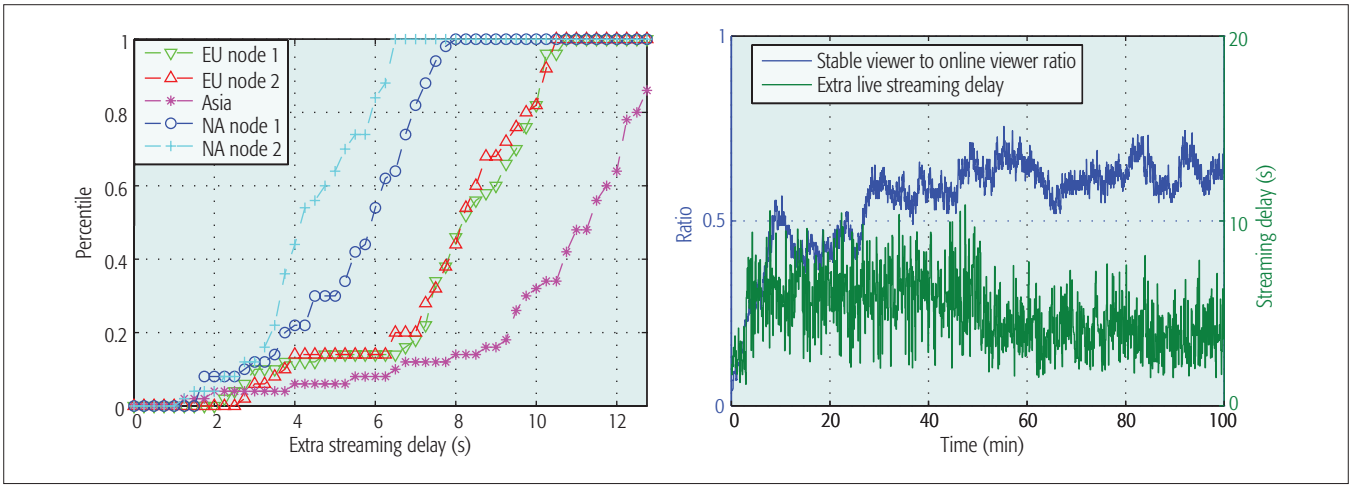

Figure 3. PlanetLab-based experiment result.

log*N*) only). Nonetheless, any extra time at the scheduling moment is undesirable, as it increases the transcoding delay and consequently the live-cast delay. Therefore, we seek to combine both strategies, that is, better organization of the candidates with minimized operational cost per update. Many advanced organization structures can be applied in this context. Using the simple and mature B+ tree as an example, we can deploy a B+ tree in every region to organize all transcoding candidates, where the ordering key is the preference (individual stability) of the candidate. Figure 2b illustrates the major events for its deployment and maintenance, where the the single update operation time is reduced to $O(N\log N)$.

When the system is running, any action will generate a message recording the time, viewer/channel ID, and its type. According to the message, qualified viewers will be inserted into the B+ tree; the starting channel will be assigned with transcoding candidates; the terminating channel will release its transcoding viewers back to the pool; a leaving candidate will be removed from the tree, or be replaced if already assigned.

Prototype and Experimental Study

We implemented a prototype of our fog-based transcoding system on PlanetLab, with 5 nodes as servers, and 208 as viewers and broadcasters. Each viewer node will join the system and stay for a while according to the scaled Pareto distribution. The proposed scheduler will select the most preferred candidates, who then use ffmpeg to transcode a 3.5Mb/s 1080p video into lower-quality versions.

In terms of scheduling results, over time the stable viewer ratio oscillates around 50 to 60 percent, as shown by the blue line in Fig. 3b. We therefore focus more on the streaming performance. We measure the streaming delay, which is one critical metric for video streaming. Figure 3a shows the result measured on the five servers, from which we see huge delay variance in different regions. For example, the delay in North America is much smaller than that in Asia. To further understand the impact of delay variance, we then focus on one server instance in the NA region. As shown in Fig. 3b, the green line indicates the live streaming delay measured by recording the transmission and transcoding time of every 1 s video slice. Clearly, the delay time changes at time 3.5 min and 51 min, which is caused by two reassignment events. The experimental result indicates the existence of delay variance before the transcoded video content is streamed from servers. Although with the optimization of interaction delay the actual delay variance perceived by end viewers is mitigated, it is still worthwhile to make the scheduler take the delay performance into consideration when assigning transcoding workload for the same channel.

To see the transcoding ability of real end viewers instead of PlanetLab virtual machines, we conducted another experiment with eight devices of different popular CPU types. We use *VLC* to do H.264 transcoding for a 1080p video (3.5 Mb/s) to lower-quality versions while the device is playing a live streaming at *Source* quality from a channel on Twitch TV. For comparison, we also measured the transcoding time of the 720p video quality when the device was idle (denoted as 720p*). Table 1 shows the experiment results in the form of transcoding time to video playback time ratio. We see almost all devices can handle quality versions equal to and lower than 480p. Transcoding for the 720p version, however, is more computationally intense, as only the top three devices manage to proceed in real time. Notably, the viewing QoE of all devices is almost not affected, and lack of computational ability is mainly revealed by the long transcoding time. For comparison, we also measured resource usage of a similar workload on an Amazon AWS m3.large server. Typically, transcoding a source video from 1080p to 720p, 480p, 360p, and 240p takes around 73, 54, 42, and 35 percent CPU usage, respectively, which is around the same level of these personal devices.

In short, our experimental result confirms the real-time transcoding ability of modern CPUs, and also suggests distinguishing different unqualified viewers, as some of them can handle lower-quality versions.

Optimizing the Interaction Delay

We now examine the critical issue of cross-viewer synchronization in the CLS platform. We consider a *community* as the broadcaster and the view-

The optimal rate allocation will be obtained in a certain number of iterations. Compared to a centralized solver that directly obtains the optimal solution to the primal problem at the streaming server, our proposed InterSync algorithm is much easier to implement in practical systems.

T CPU type	720p* 2.5 Mb/s	720p 2.5 Mb/s	480p 1.2 Mb/s	360p 0.8 Mb/s	240p 0.5 Mb/s
Intel i7-3770 @3.40 GHz x4	33.7%	59.6%	25.0%	17.5%	14.5%
Intel i7-3630QM @2.4 GHz x4	45.5%	58.2%	33.1%	24.7%	19.7%
Intel i5-2400 3.1 GHz x2	53.5%	66.7%	38.4%	27.8%	19.2%
Intel i5-3210M 2.50 GHz x2	90.6%	113%	68.6%	43.1%	34%
Intel i5-4250U 1.3 GHz x2	116.3%}	191.5%	92%	70.8%	51.2%
AMD a10-4600M 2.3 GHz x2	104.3%	143.3%	77.5%	59.4%	48.2%
Intel i3-2310M 2.10 GHz x2	130.0%	155.8%	90.0%	60.3%	44.4%
Intel Core 2 Duo 2.53 GHz x2	86.7%	190.5%	171.1%	112.3%	76.9%

Table 1. Transcoding time to video playback time ratio of different devices while playing live streaming at Source quality.

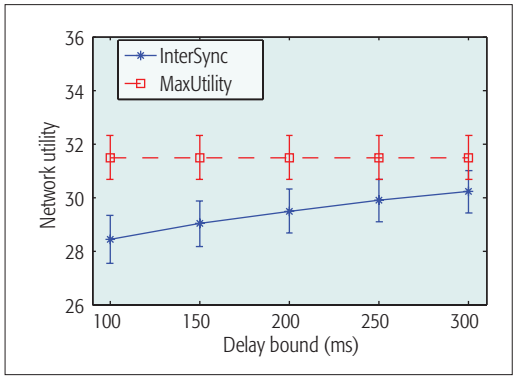

Figure 4. Comparison of network utility.

ers watching the same broadcaster at the same time who are willing to participate in community interaction through chatting. The literal interaction should be relatively synchronized such that viewers' watching experience will not be severely affected. Intuitively, this issue could be solved by dividing viewers into smaller chat channels based on viewers' delay, but this would limit the user interaction in the community. We instead address this issue by adaptively tuning the video rates at viewers to achieve cross-viewer synchronization.

We start from a streaming session consisting of one broadcaster and a set of viewers. A streaming server with certain bandwidth capacity serves the viewers in this session, and the exact video rate allocated to each viewer is adjustable through transcoding, as long as it does not exceed the viewer's downloading bandwidth.

As in previous studies [13, 14], we consider the viewing experience of a viewer is given by a utility function of the allocated video rate, which is strictly concave, increasing, and continuously differentiable. For the streaming session, our objective is then to optimize the viewers' experience through rate adaptation (i.e., tuning streaming rate of each viewer), and meanwhile ensure that the difference between any pair of viewers' network delay is bounded by an empirical threshold. Furthermore, the total streaming rates of all the viewers should not exceed the server's capacity. This leads to a typical network utility maximization (NUM) problem.

Since the objective function is differentiable and strictly concave, and the feasible region is compact, the optimal solution exists (although it may not be unique). This convex problem can be directly solved in a centralized way via the classic simplex and interior-point-based algorithms, provided that the streaming server has the information of each viewer, that is, the end-to-end throughput and the end-to-end delay. It is, however, worth noting that a centralized solver can be very time-consuming for large sessions of thousands of current viewers, not to mention viewers' dynamic join and leave activities.

To this end, we develop a distributed algorithm, *InterSync*, based on dual decomposition, which allows viewers to select appropriate playback bit rates without knowing others' information. The basic idea is that we can solve the dual of the original NUM problem at two levels. At the lower level, each viewer computes its video rate by maximizing its own surplus (i.e., utility minus payment) based on the aggregate price of bandwidth and feeds this value back to the streaming server; at the upper level, the streaming server updates the dual variables according to the feedback information of individual viewers. The optimal rate allocation will be obtained in a certain number of iterations. Compared to a centralized solver that directly obtains the optimal solution to the primal problem at the streaming server, our proposed InterSync algorithm is much easier to implement in practical systems.

We conducted a simulation to compare InterSync with a baseline *MaxUtility*. In MaxUtility, the network utility is maximized subject only to the bandwidth capacity of the streaming server, while the cross-viewer synchronization constraint is not considered. The bandwidth capacity of the streaming server is 100 Mb/s, and the number of viewers is 50. Each viewer's bandwidth is uniformly distributed between 0.5 and 5 Mb/s, and the network delay is uniformly distributed between 50 and 500 ms. The source encoding rate is 5 Mb/s. We vary the end-to-end delay bound δ from 50 to 300 ms.

We report the average and the standard deviation of network utility of both algorithms under different delay bounds, shown in Fig. 4. Since the delay bound is not considered in MaxUtility, the obtained network utility does not change. The network utility obtained by InterSync becomes higher as the delay

bound increases, since the feasible region of the optimization problem (1) also expands with larger delay bound. Although MaxUtility outperforms InterSync in terms of network utility, the real-time community interaction becomes intolerable. In our simulation, the maximum end-to-end delay difference of MaxUtility ranges from 826.0 to 914.4 ms, with an average of 861.4 ms. As evidenced by the measurement above, this level of delay difference would easily lead to tens of seconds broadcast delay among viewers.

Conclusion

In this article, we present a novel fog-based transcoding framework to smartly assign qualified stable viewers to channels for transcoding assignment in CLS systems. We evaluate our design through a PlanetLab-based experiment as well as an end-viewer transcoding experiment. Several future improvements could be studied for our framework. First, it would be interesting to see hybridization of the proposed system with dedicated cloud servers, to have better streaming performance at low cost. Second, more dedicated reward mechanisms can be studied to trigger viewers' incentive of participation. Third, the dynamic pattern inside each region can be studied in a fine-grained manner, after which we can further present strategies for adapting viewer qualifying criteria. Moreover, given that the sources and viewers are shifting to mobile platforms (e.g., Facebook Live and Twitter Periscope for mobile livecast), effectively using their computation resources without significantly increasing their communication costs and energy consumption becomes a challenge too.

Acknowledgment

This publication was made possible by NPRP grant #[8-519-1-108] from the Qatar National Research Fund (a member of Qatar Foundation), and by a Discovery Grant and a Steacie Memorial Fellowship from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] B. Li *et al.*, "Two Decades of Internet Video Streaming: A Retrospective View," *ACM Trans. Multimedia Computing Commun. Applications*, vol. 9, no. 1s, article no. 33, Oct. 2013.

[2] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," *Proc. ACM MMSys 2011*.

[3] M. Kaytoue *et al.*, "Watch Me Playing, I Am a Professional: A First Study on Video Game Live Streaming," *Proc. ACM WWW 2012*.

[4] W. A. Hamilton, O. Garretson, and A. Kerne, "Streaming on Twitch: Fostering Participatory Communities of Play within Live Mixed Media," *Proc. ACM CHI 2014*.

[5] C. Zhang and J. Liu, "On Crowdsourced Interactive Live Streaming: A Twitch TV-Based Measurement Study," *Proc. ACM NOSSDAV 2015*.

[6] R. Aparicio-Pardo *et al.*, "Transcoding Live Adaptive Video Streams at a Massive Scale on the Cloud," *Proc. ACM MMSys 2015*.

[7] F. Chen *et al.*, "Crowdsourced Live Streaming over the Cloud," *Proc. IEEE INFOCOM 2015*.

[8] Q. He *et al.*, "Coping with Heterogeneous Video Contributors and Viewers in Crowdsourced Live Streaming: A Cloud-Based Approach," *IEEE Trans. Multimedia*, vol. 18, no. 5, May 2016, pp. 916–28.

[9] F. Bonomi *et al.*, "Fog Computing and Its Role in the Internet of Things," *Proc. ACM MCC 2012*.

[10] Y. Singer and M. Mittal, "Pricing Mechanisms for Crowdsourcing Markets," *Proc. ACM WWW 2013*.

[11] F. Dobrian *et al.*, "Understanding the Impact of Video Quality on User Engagement," *Proc. ACM SIGCOMM Comp. Commun. Review*, vol. 41, no. 4, Aug. 2011, pp. 362–73.

[12] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs," *IEEE/ACM Trans. Networking*, vol. 21, no. 6, Feb. 2013, pp. 2001–14.

[13] J. Huang *et al.*, "Joint Source Adaptation and Resource Allocation for Multi-User Wireless Video Streaming," *IEEE Trans. Circuits Systems Video Technology*, vol. 18, no. 5, May 2008, pp. 582–95.

[14] M. Chen *et al.*, "Utility Maximization in Peer-To-Peer Systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, Dec. 2008, pp. 169–80.

Biographies

Qiyun He (qiyunh@cs.sfu.ca) received a B.Eng. degree from Zhejiang University, China, and a B.Sc. degree from Simon Fraser University, British Columbia, Canada, both in 2015. He is currently an M.Sc. student at the School of Computing Science, Simon Fraser University. His research interests include cloud computing, crowdsourcing, multimedia systems, and networks.

Cong Zhang (congz@cs.sfu.ca) received his M.S. degree in information engineering from Zhengzhou University, China, in 2012, and is currently working toward a Ph.D. degree in computing science at Simon Fraser University. He is currently working with the Network Modeling Research Group, Simon Fraser University. His research interests include multimedia communications, cloud computing, and crowdsourced live streaming.

Xiaoqiang Ma (xma10@cs.sfu.ca) received his B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, in 2010, and his M.Sc. and Ph.D. degrees from Simon Fraser University in 2012 and 2015, respectively. He is currently an assistant professor with the School of Electronic Information and Communication, Huazhong University of Science and Technology. His research interests include wireless networking, multimedia, cloud, and big data.

Jiangchuan Liu [F] (jcliu@cs.sfu.ca) is currently a full professor (with a University Professorship) in the School of Computing Science at Simon Fraser University. He is an NSERC E.W.R. Steacie Memorial Fellow. He is a Steering Committee Member of *IEEE Transactions on Mobile Computing*, and an Associate Editor of *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Big Data*, and *IEEE Transactions on Multimedia*.

Given that the sources and viewers are shifting to mobile platforms (e.g., Facebook Live and Twitter Periscope for mobile livecast), effectively using their computation resources without significantly increasing their communication costs and energy consumption becomes a challenge too.