CrowdTranscoding: Online Video Transcoding With Massive Viewers

Qiyun He, *Member, IEEE*, Cong Zhang, *Student Member, IEEE*, and Jiangchuan Liu, *Fellow, IEEE*

***Abstract*—Driven by the advances in personal computing devices and the prevalence of high-speed network accesses, crowdsourced livecast platforms have emerged in recent years, through which numerous broadcasters lively stream their video content to fellow viewers. Compared to professional video producers and broadcasters, these new generation broadcasters are highly heterogeneous in terms of the network/system configurations and, therefore, the generated video quality, which calls for massive encoding and transcoding in order to unify the video sources and serve multiple quality versions to viewers with different configurations. On the other hand, with the rapid evolution in the hardware industry, high-performance processors become mainstream in personal computer market. More end devices can easily transcode high-quality videos in realtime. We witness huge computational resource among the massive fellow viewers that could potentially be used for transcoding. In this paper, we propose CrowdTranscoding, a novel framework for crowdsourced livecast systems that offloads the transcoding assignment to the massive viewers. We identify that the key challenges in CrowdTranscoding are to detect qualified stable viewers and to properly assign them to the source channels. We put forward a viewer crowdsourcing transcode scheduler to smartly schedule the workload assignment. Our solution has been evaluated under diverse viewer/channel conditions as well as different parameter settings. The trace-driven simulation confirms the superiority of CrowdTranscoder, while our PlanetLab-based and real world end-viewer experiments show the practical performance of our approach, which also give hint to the further enhancement.**

***Index Terms*—Video transcoding, livecast, crowdsourcing.**

I. Introduction

CROWDSOURCING was first introduced in 2005 by Merriam-Webster as a way to obtain resources by collecting contributions from crowds of people, instead of employees or suppliers [1], [2]. With the advances of personal computing devices and the prevalence of high speed Internet accesses, *crowdsourced livecasting* [3]–[5] has emerged in the market, which has seen explosive growth recently. In such systems, numerous amateur broadcasters lively stream their video contents to viewers around the world. Fellow viewers watching the same channel constantly interact with each other and the channel broadcaster through chatting messages. One of the most successful examples, Twitch TV, has 35,610 concurrent broadcasters and more than 2 million concurrent viewers during peak time in 2015. The total stream time was more than 241 billion minutes through the whole year [6]. The viewers are also more active in such interactive livecast platforms. For example, the average monthly watch time spent by each viewer was 421.6 minutes on Twitch TV, while it was only 291.0 minutes for Youtube [6]. While Twitch TV is dominating the global market overall, we have also seen some regional diversities, such as Douyu TV,[1] Panda TV,[2] Inke[3] in China, and Afreeca TV,[4] Azubu TV[5] in South Korea. On the other hand, livecast channel-based online communities with social relationships have also emerged [4], [7], and active viewers have shown strong willingness to support the platforms and broadcasters, through donation and monthly subscription.

Given that most of the broadcasters are non-professionals, the content formats and quality can be much more heterogeneous than the early generation of video streaming systems, which typically use P2P (Peer-to-Peer) or CDN (content distribution Networks) [8]–[10] to distribute professionally produced and readily available video content. There is a strong need to unify video contents into industrial standard representations. Facing the dramatic scalability challenges, cloud computing becomes a preferred solution for crowdsourced livecast platforms to handle video ingesting and transcoding [3], [11], [12]. However, video transcoding is computation-intensive [13]. As we measured using an Amazon AWS m3.large server, transcoding a source video from 1080p to 720p, 480p, 360p, 240p takes around 73%, 54%, 42%, 35% CPU usage, respectively. This means a general cloud instance can only conduct at most two transcoding tasks simultaneously. And thus given the large number of concurrent live channels on these platforms, a pure cloud-based approach for video transcoding is expensive. For that reason, real-world platforms only provide transcoding service to a small portion of broadcasters. For example, Twitch TV only offers transcoding service to its premium broadcasters, which make up only 1% to 1.5% of all broadcasters. Moreover, the requirement on streaming latency in such interactive environment is stringent, as a long latency severely affects the viewer-broadcaster interactive

Manuscript received July 6, 2016; revised November 22, 2016; accepted December 1, 2016. Date of publication January 11, 2017; date of current version May 13, 2017. This work was supported by the NPRP under Grant 8-519-1-108 from the Qatar National Research Fund (a member of Qatar Foundation). The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Lingfen Sun.

The authors are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: qiyunh@cs.sfu.ca; congz@cs.sfu.ca; jcliu@cs.sfu.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2017.2652061

[1][Online]. Available: https://www.douyu.com
[2][Online]. Available: http://www.panda.tv
[3][Online]. Available: https://www.inke.cn
[4][Online]. Available: http://www.afreeca.tv
[5][Online]. Available: http://www.azubu.tv

experience [14]. In a pure cloud-based system, due to the service and budget limitation, the actual physical server however may be far away from the live source, which inevitably causes higher streaming latency. We therefore want to seek for more budget-efficient solutions to provide transcoding and streaming service with shorter latency.

In the livecast platforms, we also see huge potential crowdsourcing computational resources on the massive viewer side. As we measured on Twitch TV, the concurrent viewer number is always 20–50 times more than the channel number, and only top 10% of the channels need to be considered as they attract more than 98% of the total viewers. Additionally, for major crowdsourced livecast platforms such as Twitch TV, Azubu TV and Dailymotion Games, their channel contents are mostly game scenes or live gaming events, and most of their viewers are gaming players, a great portion of whom possibly have high-end system configuration that can easily handle the transcoding jobs. Indeed, as most mainstream CPUs nowadays have high performance [15], even a general purposed computer is able to transcode while playing the streaming. Since major crowdsourced livecast platforms provide free service, it is hard for them to deploy expensive dedicated servers for transcoding a large number of channels. The potential of edge users in crowdsourced livecast systems was however ignored in the past, which is indeed strong in such crowdsourced environment. We thus put forward CrowdTranscoding, a novel framework to offload the massive transcoding workload to the crowd of viewers, enabling ABR/DASH for more livecast channels. Notably, from the architectural perspective, we base our system on fog computing, which extends the cloud computing paradigm to the edge of the network [16]. It is also worth mentioning that the nature of our proposed system is twofold, that is, from the perspective of computational resources collaboration, it is a crowdsourced system, while from the network perspective, it follows the fog computing paradigm, as the workload is pushed to the edge of the network.

The remainder of this paper proceeds as follows: Section II analyzes the insights observed in our measurement and discusses the motivation. Section III presents an overview of using viewers to transcode video content in crowdsourced livecast systems, and further implies the unique challenges. In Section IV, we first formulate the problem of viewer to channel assignment, and then put forward an efficient solution VCTS (Viewer Crowdsourcing Transcode Scheduler) which minimizes the undesirable impacts on system performance. In Section V, we evaluate our design through trace-driven simulations. To better understand the practical performance of our system, we conduct a PlanetLab-based experiment, and a realworld transcoding experiment in Section VI. We further discuss issues implied by our experiments and related enhancement in Section VII. Finally, Section VIII concludes the paper and discusses potential future directions.

II. Observations and Motivations

We have conducted two separate measurement studies on the popular crowdsourced livecast platform Twitch TV, focusing on channel-related information and viewer behavior, respectively. Our analysis on our latest data highlights the distinct characteristics of such a new generation of crowdsourced livecast services.

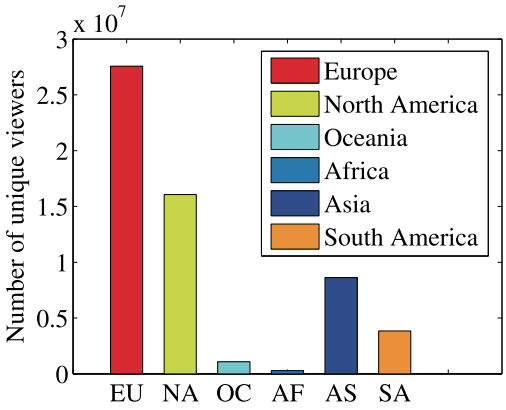


Fig. 1. Viewer distribution in six continents during Oct. 21 to Nov. 21, 2015.

A. Observations on Channels/Broadcasters

In the first measurement, we focus on the channels/broadcasters side. 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).[6] This public API provides the game name, viewer number, stream resolution, average fps, broadcaster language, premium partner status (Yes/No), and some other related information of every broadcast channel. It however does not provide certain detailed information about the viewers, such as their network conditions, choices of video resolution, or physical distributions. The geographical and bandwidth information of the broadcasters is not provided by this API, either. Therefore, we rely on another dataset in [17] when estimating the network conditions of viewers/broadcasters, and, for the geographic distribution of viewers, we use the data from [18] to represent the global streaming traffic and demographic statistics.

Many of our observations from the Twitch TV data are consistent with those from conventional video streaming systems, e.g., heterogeneous and dynamic viewers [19], [20]. Yet the heterogeneity becomes much stronger, not only on the viewers' side, but also on the broadcasters' side. There is a variety of source stream resolutions. From the data we captured on March 7, 2015 at 15:00 PST, we find 177 different resolutions ranging from 116p to 1600p. Even for the source streams with the same resolution, there are quite different bitrates. There is clearly a need to unify these source streams into industrial standard representations.

We also use the API provided by *quantcast* [18] to measure the number of unique viewers located in 221 countries/regions from Oct. 21 to Nov. 21, 2015, as shown in Fig. 1. It is clear to see that despite of the unbalanced distribution in different regions, the viewer base is huge, with abundant potential computational resource available. To better explore such viewer resource,

[6][Online]. Available: http://dev.twitch.tv

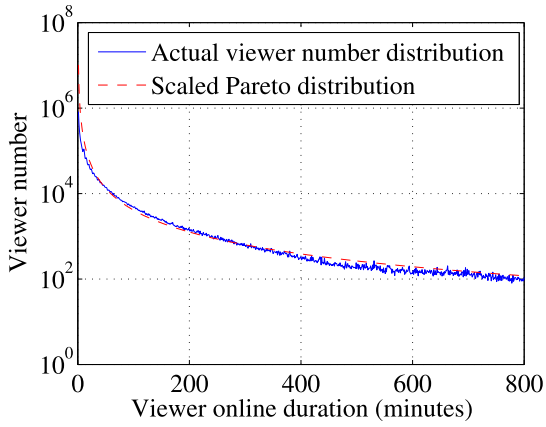


Fig. 2. Fitting viewer distribution with Pareto distribution.

we further study the detailed viewer behavior in the following subsection.

B. Observations on Viewer Behavior and Insights

To better understand the viewer behavior, we also had another measurement study conducted capturing viewers' online traces of five popular Twitch channels (i.e., $hearthstonefr$, $monstercat$, $rocketbeanstv$, $twitchplayspokemon$ and $versuta$), from January 25, to February 27, 2015. During this measurement, we 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.

The blue line in Fig. 2 shows the distribution of viewer online duration of all observed viewers in the five channels. We see that the majority of viewers tend to leave the channel in a short time. More specifically, 90% of the viewers leave the channel within 100 minutes. However, the longer a viewer is online, the more likely this viewer will continue to be online. From time to time, there have been lots of studies in the literature indicating the consumption of web and media contents follows the classic Pareto distribution, for both the media content popularity/lifespan [20], [21], and client behavior [22]. Inspired by these empirical evidences, we also try to use a customized Pareto distribution function to fit the viewer online duration where $\alpha = 0.7$ and $x_m = 2$. The fitting function is shown as the red imaginary line in Fig. 2.

Each channel also has slightly different viewer online duration distribution. As shown in Fig. 3, the channel $versuta$ has a larger portion of its viewers with one-time online duration less than 100 minutes than other four channels, while channel $monstercat$ has the least. In general, we see around 20% of the viewers will stay online for more than one hour, though this number varies from channel to channel. This indicates a considerable portion of viewers are stable for non-stop video transcoding. Some other statistics also imply the viability of viewer-based transcoding. Fig. 4 shows the number of online times of viewers during the measured period of 33 days. We see around 5% of the viewers get online equal to or more than once per day on average. Finally Fig. 5 shows the standard deviation

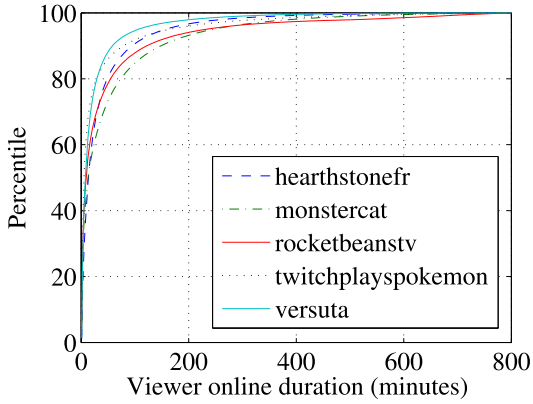


Fig. 3. CDF of viewer one-time online duration of five channels.

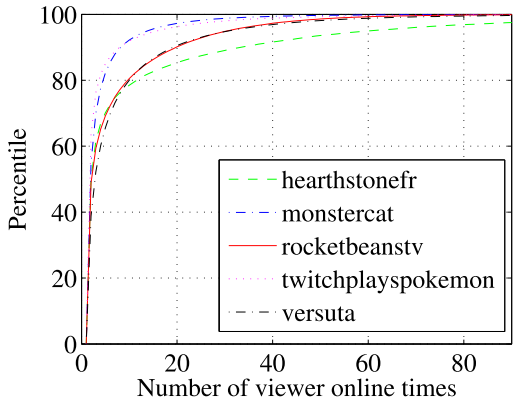


Fig. 4. CDF of viewer online frequency during measured period.

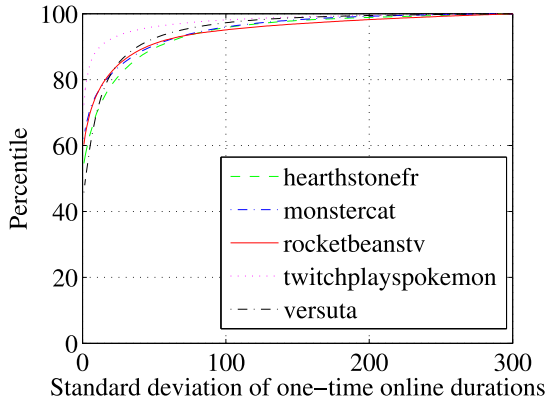


Fig. 5. CDF of standard deviation of viewer one-time online duration.

of the viewer online time. Overall around 80% of the viewers have quite consistent online duration every time.

In short, in large crowdsourced livecast systems such as Twitch TV with massive viewer base, a considerable portion of concurrent viewers are potential resources for stable video transcoding. On the other hand, the 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 to appropriately make use of their resource.

III. CrowdTranscoding: Architecture and Challenges

We now illustrate the overall architecture of our design. Globally, the system is divided into multiple regions, where each region has its own regional datacenter (also referred to as "regional server") for ingesting source videos, assigning transcoding viewers, recollecting transcoded video and forwarding the processed streams for further delivery. For a single broadcaster, his/her livecast content will first be encoded locally, and then collected by the regional server through protocols such as RTMP (Real Time Messaging Protocol) [14]. The parameter settings for video encoding during this phase are decided by the broadcasters and therefore may vary from person to person. However, major service providers usually have several recommended settings for encoding resolution and bitrate, for broadcasters to choose based on their hardware configuration and upload bandwidth availability. For example, Twitch TV has a detailed guide[7] for video encoding and other broadcast setup. When the video contents reach the regional server, several viewers will then be selected for video transcoding according to certain criteria, or requests will be sent to neighbor regions if such transcoding viewers cannot be found locally. Leaving assigned viewers will cause reassignment. At the selected viewers' side, video contents of different qualities are generated and sent back to the regional datacenter. Finally, the transcoded video is forwarded to other regional datacenters to serve all viewers.

Intuitively, the system is always balanced given there are much more viewers than broadcasters overall. However, in reality the viewer and broadcaster numbers are highly dynamic over time, for both a single region and the whole global system, as in such free systems users come and leave by their will. On the other hand, viewers' computational ability and networking conditions are dramatically various. Different viewers also tend to stay in the system for a different amount of time, and causal selection may cause a large number of reassignments, leading to high system overhead for recalculation and short absence of the target quality version during the reassigning time. Reducing cross-region assignment is our objective, too, as it increases the streaming latency.

We assume all viewers are cooperative for the transcoding assignment, with motivations such as exemption for advertisements. In fact, other more advanced methods, such as auction models, can also be introduced for the contributing viewers to share advertising revenue with the service providers.

IV. Problem Formalization and Solutions

In this section we first formulate the problem, and then provide the key component of CrowdTranscoding – VCTS (Viewer Crowdsourcing Transcode Scheduler) which can smartly assign qualified stable viewers to live channels for transcoding. Note that, as latency plays a critical role in livecast, all transcoding candidates will be evaluated through a video transmission and transcoding test, and candidates who do not meet the preset latency criterion will be disqualified prior to any other selection.

[7][Online]. Available: https://help.twitch.tv/customer/portal/articles/1262922-open-broadcaster-software

TABLE I
Notations in the Formalized Problem

Name	Description
t	given time point
L	live channel termination time
$T(t)$	waiting threshold
$p(x)$	Pareto distribution of viewers
s	individual stability
$\bar{d}$	average online duration
σ	standard deviation of online duration
C	set of live video broadcasters
V	set of viewers
$\mathbb{R}$	set of regions
P	candidate pool

Before illustrating the formalization in detail, we summarize important notations in Table I.

A. Extracting Stable Transcoding Candidates

We want the selected viewers to be as stable as possible, so that they can provide long time non-stop transcoding to minimize the number of reassignment. Our observations in Section II indicate that the stability of a viewer in general is proportional to the existing time he/she has already spent in the channel. Thus, we can set a *waiting threshold* $T(t)$, after passing which the viewer can be regarded as stable. Obviously the longer this waiting threshold is, the more stable those left viewers are. However, a longer threshold also means fewer viewers can be qualified by this criterion, and more potential transcoding resource are wasted during the waiting process. Therefore, we want to maximize the mathematical expectation of the non-stop transcoding time of all viewers, given the viewer online duration distribution. Based on the data we captured, the probability function of viewer distribution over online time can be estimated as

$$p(x) = \frac{\alpha x_m^{\alpha}}{x^{\alpha+1}} \text{ when } x \geq x_m, \text{ typically } \alpha = 0.7 \text{ and } x_m = 2.$$

Thus, if a live channel terminates at time L, for any viewer enters the channel at given time t, we want to find the optimal waiting threshold $T(t)$ that maximizes the mathematical expectation of serving time of the viewer

$$\underset{T(t)}{\operatorname{argmax}} \left[\frac{\int_{T(t)}^{L-t} xp(x)\,dx + \int_{L-t}^{+\infty} (L-t)p(x)\,dx}{\int_{T(t)}^{+\infty} p(x)dx} - T(t) \right] =$$

$$\underset{T(t)}{\operatorname{argmax}} \left[\frac{T(t)^{\alpha}}{(L-t)^{\alpha-1}(1-\alpha)} + \frac{T(t)}{\alpha - 1} \right] = (L-t)\left(\frac{1}{\alpha}\right)^{\frac{1}{1-\alpha}}.$$

The answer is found by making the derivative of $T(t)$ inside $[\cdot]$ to 0, which is

$$\frac{1}{1-\alpha} - \frac{\alpha}{\alpha - 1}\left(\frac{T(t)}{L-t}\right)^{\alpha-1} = 0.$$

In the above formula, $\int_{T(t)}^{L-t} xp(x)\,dx$ and $\int_{L-t}^{+\infty}(L-t)p(x)\,dx$ are the accumulative transcoding time of stable viewer leaving before and after the channel terminates, respectively. And the

whole part inside bracket $[\cdot]$ is the expectation of total transcoding time of filtered stable viewers with waiting threshold $T(t)$. Typically, when $\alpha = 0.7$, $T(t)$ is approximately $0.304(L-t)$. In reality, due to the dynamics of viewers, the value of α varies from 0.5 to 0.9, and thus the optimal waiting threshold is from $0.25(L-t)$ to $0.34(L-t)$. In general, any viewer watching a channel longer than 34% of the left time of that channel can be regarded as qualified stable viewer candidate for transcoding.

B. Stability of Individual Viewer

Now that we have the optimal minimum online time threshold to filter stable viewers in general, we also want to estimate the stability of certain individual viewer. Some heuristics, such as viewer age, gender and video quality [23], [24], can be used to further estimate the individual stability in a fine-grained manner. Here we use a simple heuristic that measures the average online duration and standard deviation of a viewer's online record, denoted as $\bar{d}$ and σ, respectively. We use a linear combination of them to further estimates the individual stability of each viewer, which is

$$s = \lambda \cdot \bar{d} - (1-\lambda) \cdot \sigma; \lambda \in (0,1).$$

The default λ is 0.8 in evaluations, which is acquired by analyzing our captured viewer trace. This heuristic in particular is inspired by 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. We leave it for further study to find more sophisticated heuristic.

C. Selecting Stable Viewers for Transcoding: Initial Setup

We denote the live video broadcasters as a set $C = \{c_1, c_2, \ldots, c_m\}$, and viewers as $V = \{v_1, v_2, \ldots, v_n\}$. Let $\mathbb{R} = \{R_1, R_2, \ldots, R_n\}$ be the set of regions. For each participated viewer, we evaluate his/her realtime transcoding ability and networking performance, which is conducted when the viewer is already watching a live channel at *Source* quality. If any item in the evaluation is lower than the corresponding threshold, this viewer will be unqualified. For each region, all qualified viewers are put into a candidate pool, ranked in decreasing order of their individual stability if known. Candidates with the highest individual stability are the most preferred.

During this initial setup, due to the lack of knowledge of detailed system condition, the scheduling is conducted in a conservative manner. That is, we guarantee the assignment for the most popular live channels, to maximize the overall viewer QoE (Quality of Experience). Ideally, if we need $q+1$ quality versions for each channel, q qualified viewers are assigned to every channel within the same region. For existing crowdsourced livecast systems, this ideal situation is achievable overall as the viewer base is much larger than the channel number all the time. But due to the dynamics and randomness of the viewer behavior, unbalanced viewer distribution across regions may happen, in which case some transcoding viewers need to be "borrowed" from the neighbor region. Algorithm 1 (VCTS I) shows such detailed strategy for the initial setup. First, all channels are ranked in decreasing order of their popularity (viewer number). In each region, candidate viewers are ranked in decreasing order of individual stability. For each channel, q most preferred candidates is chosen in the same region for transcoding. If the regional candidate pool is empty, the scheduler tries to make the assignment from the neighbor regions.

Algorithm 1: CrowdTranscoding – VCTS I

```
Sort C in decreasing order of their Popularity
Sort V in decreasing order of preference level in each region
for i from 1 to |C| do
    let region R be the region c_i in
    if there are enough viewer left in R then
        assign first q viewers to c_i for transcoding
    else
        rank all neighbor regions in increasing distance with c_i's region
        for each region R in the ranked region list do
            try to assign q viewers to c_i
            if assignment succeeds then
                break
            else if all regions are tried then
                terminate scheduler
            end if
        end for
    end if
end for
return the schedule
```

D. Maintaining the System and Handling Viewer Dynamics

The VCTS I indicates the initial setup. However, as both viewers and channels are highly dynamic, rescheduling using VCTS I at any change causes a large number of reassignments as well as huge computational overhead of scheduling calculation. Therefore, when maintaining the system, we are reluctant to change any assignment once it is made, unless either the directly related viewer or channel has state change.

To maintain the candidate pool, we can simply re-rank all candidates at any update, which however takes $O(N)$ time every update and $O(1)$ time at the assignment time. It incurs significant calculation when N is large with frequent updates. Since there are much 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 $O(NlogN)$ only). Nonetheless, any extra time at the scheduling moment is undesirable, as it increases the transcoding delay and consequently the live streaming delay. Therefore, we seek to combine both strategies, i.e., a 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 index) of the candidate. Thus a single update operation time is reduced to $O(logN)$.

Algorithm 2: CrowdTranscoding – VCTS II

```
Input: a message indicating 1) a viewer joins/leaves a
  channel, or is qualified to be a stable viewer 2) a
  channel starts/terminates
if message.action is channelStart then
    assign top q viewers with highest preference from
    candidate pool P
    if candidate pool P does not have q viewers then
        make assignment with left candidates in P;
        send message' to nearest neighbor region with
        unfinished assignments
    end if
else if message.action is channelEnd then
    release transcoding viewers for message.
    channelID back to candidate pool P
else if message.action is qualified then
    add message.viewerID into candidate pool P
else if message.action is part then
    if viewer viewerID is assigned to any channel c_i and
    doing the transcoding then
        if candidate pool P is not empty then
            choose most preferred viewer in P to replace the
            leaving viewer
        else
            send message' to nearest neighbor region with
            this assignment
        end if
    end if
    remove message.viewerID from the candidate pool
    P as the viewer leaves the system
end if
```

Algorithm 2 shows the system maintaining algorithm VCTS II. When the system is running, any action will generate a message recording the time, viewer/channel ID, and the action type ($join$, $qualified$, $part$, $channelStart$ or $channelEnd$). According to the message, qualified viewers will be inserted into the B+ tree; Starting channel will be assigned with transcoding candidates; Terminating channel (i.e., $message.action$ is $channelEnd$) will release its transcoding viewers back to the pool; Leaving candidate (i.e., $message.action$ is $part$) will be remove from the tree, or be replaced if already assigned.

V. Performance Evaluation

To evaluate our framework, we have conducted extensive simulations using large scale data captured by Twitch API. We first briefly introduce the selected datasets, and then continue on to present the methodology as well as the evaluation results.

A. Data Sets and Traces

Our channel-based viewer trace data captured with Twitch API contains the join/leave record of viewers in certain channel from January 25 to February 27, 2015. Each record message includes the viewer ID, time of the action and the action type (join or leave). We also record the instant time-based information of all channels, once every five minutes, from February 2015 to June 2015. In each time-based record we have the detailed information of all live channels at capturing time. The total channel number and viewer number are therefore logged together over time, which is used to estimate the average channel to viewer ratio in our simulation. On the other hand, the skewness of the channel popularity is extremely high, and top 10% channels attract around 98% of the total viewers. We therefore only regard those top 10% channels as *qualified channels*. Fig. 6 shows viewer number, channel number and the viewer to qualified channel ratio (referred as viewer-to-channel ratio, or V/C ratio), from March 1 to March 7, 2015. We see in general the V/C ratio is between 200 to 500, which means at least 200 viewers, or around 84 qualified viewers can serve one channel for transcoding all the time, considering 30% mobile and 40% unqualified viewers.

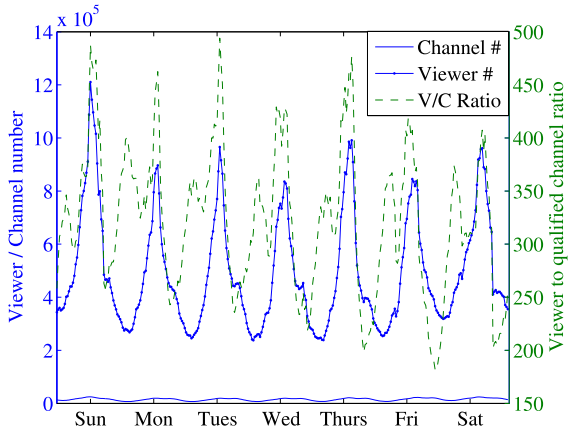


Fig. 6. Viewer number to qualified channel number ratio (V/C Ratio) over time.

B. Methodology

For comparison, we implemented four strategies: Top-N with dedicated servers/cloud, and VCTS using *online* viewers / *qualified* stable viewers / most *preferred qualified* stable viewers. Top-N, the current strategy of Twitch TV, provides full video representation set to the top N most popular broadcasters, but only the original video quality for the rest of the broadcasters. We set N = 300, estimated according to the Twitch partner program[8] requirements and the trace data we collected. For each transcoded channel, we assume there will be equal number of viewers watching each video quality version, while for channels streaming only at source quality, we assume their viewers watch the original quality version even without enough bandwidth, in which case video stall will happen. We regard three hours as the average channel duration and randomly generate channel records over viewer records with different average viewer-to-channel ratio. We also set the default waiting threshold to the optimal value calculated in Section IV which is 60 minutes in this case. Given the sufficient viewer number, for simplicity we set the threshold fixed, instead of exactly 33% of the channel remaining time. We use *Reassignment count*,

[8][Online]. Available: http://www.twitch.tv/p/partners

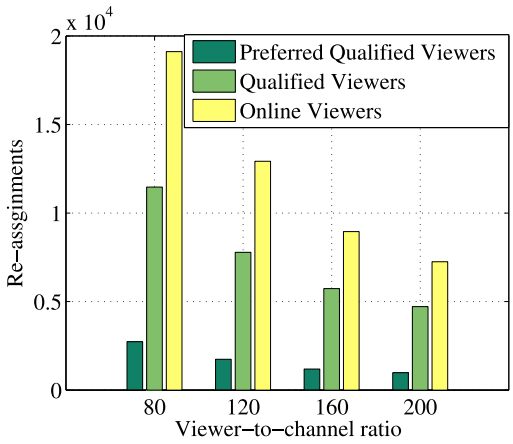


Fig. 7. Reassignment count of VCTS with different viewer selection strategies.

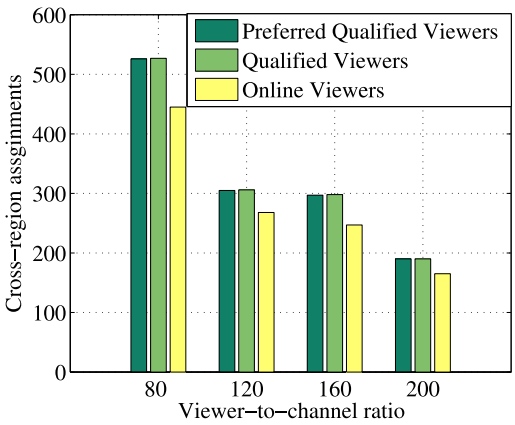


Fig. 8. Cross-region assignment count of VCTS with different strategies.

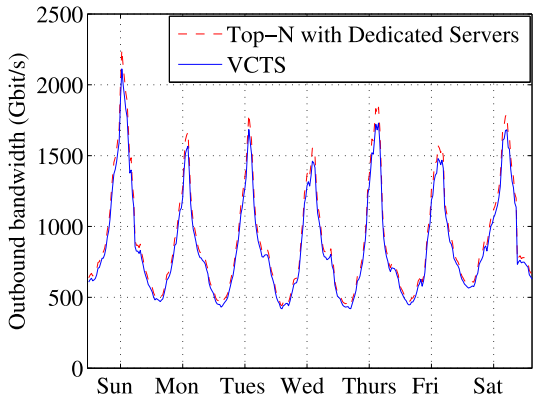


Fig. 9. Bandwidth comparison of Top-N and VCTS.

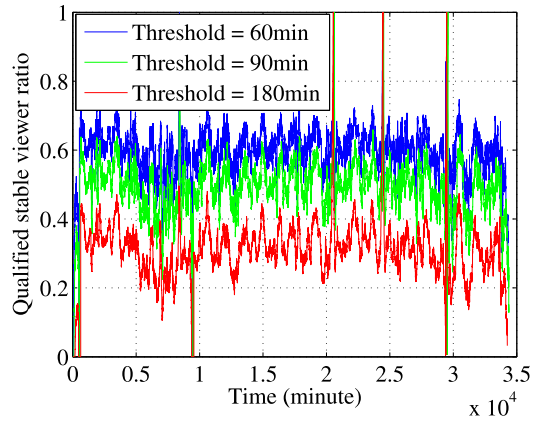


Fig. 10. Qualified stable viewer ratio under different waiting thresholds.

Cross-region assignment count as metrics to measure our simulation results.

C. Assignment Quality and Cost

We first conduct simulations to investigate the impacts of different viewer selection strategies for VCTS under default settings. Fig. 7 shows the reassignment counts under different V/C ratios (X axis). In our simulation, the case with 120 V/C ratio is the real world scenario on average, and we also have several other cases to reflect the system dynamics. We see strategy "online viewers" has the highest reassignment count, which is around 50% more than that of strategy "qualified viewers". The strategy "preferred qualified viewers" performs the best, indicating the great improvement of using preference regarding to the individual stability.

In the meanwhile, we also measure the number of cross-region assignments. Fig. 8 shows the difference of three strategies for VCTS. Clearly, the strategy "online viewers" has the fewest cross-region assignments. Such result is because this approach has the largest candidate pool and thus less likely to overrun the availability limit. Strategies "qualified viewers" and "preferred qualified viewers" have similar results, with around 20% more than that of "online viewers". Their similarity is because a better organization of the viewer pool does not mitigate the availability overrun situation; Their small difference is caused by their different scheduling results.

As the Top-N approach is implemented by the cloud or dedicated servers, there is no cross-region assignment and reassignment for comparison. However, in terms of the outbound bandwidth, Top-N has around 6.16% and 5.82% more bandwidth consumption than VCTS at peak and valley time, respectively (as shown in Fig. 9). This is because in the Top-N approach, a portion of viewers without enough bandwidth are forced to watch at the *Source* bitrate. They end up with playback stalls but still consume more bandwidth than watching a lower quality version. When VCTS is deployed, they switch to lower quality versions, which lowers the total bandwidth consumption. Note that the real-world situation for bandwidth however may vary dramatically from region to region, while the result in our simulation provides a more generic view.

D. Impacts of Waiting Threshold

We further conduct simulation to evaluate influence of waiting thresholds. We first check the ratio of qualified stable viewers of total viewers with different waiting thresholds, as shown in Fig. 10. We see from time to time there are around 60%, 50%, and 30% qualified stable viewers with waiting threshold 60, 90, 180 minutes, respectively. This indicates we have more freedom

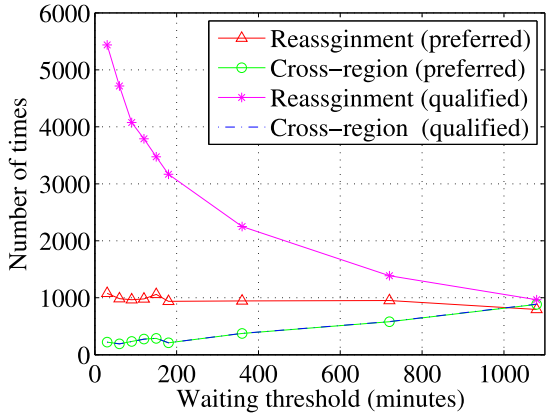


Fig. 11. Reassignment count and cross-region assignment count (V/C ratio = 200).

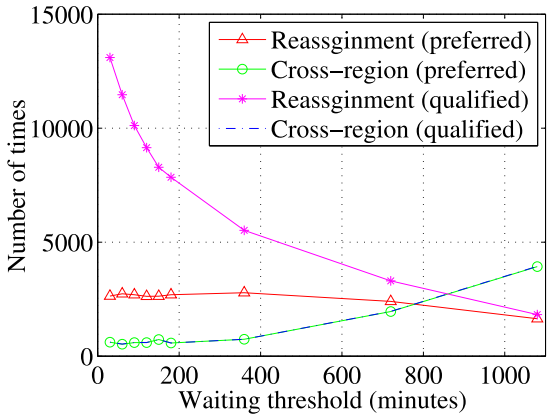


Fig. 12. Reassignment count and cross-region assignment count (V/C ratio = 80).

to choose a larger waiting threshold for real world systems like Twitch TV with massive viewer base.

To further verify this, we measure the reassignment count and cross-region assignment count of VCTS with strategies "qualified viewers" and "preferred qualified viewers" under two V/C ratios, as shown in Figs. 11 and 12. These two V/C ratios (200 and 80) are average maximum and minimum values observed from the Twitch dataset. Under both ratios, strategy "qualified viewers" has its reassignment count dropping fast with the increasing threshold, while strategy "preferred qualified viewers" has more stable results. This confirms that preferred viewers are the most stable ones. In terms of the cross-region assignment, both strategies have similar results, which is consistent with the former simulation. When the waiting threshold is larger than 200 minutes, any reduction in reassignment count is at huge expense of increasing cross-region assignment count. This is because the system overruns the candidate availability more frequently, though the selected viewers are more stable. Consistently, cases with lower V/C ratio has much higher reassignment count and cross-region assignment count when other parameters are the same.

In conclusion, the large viewer base and V/C ratio allow us to have a larger waiting threshold up to 200 minutes, although the benefit of careful candidate pool organization cannot be achieved by simply increasing the waiting threshold.

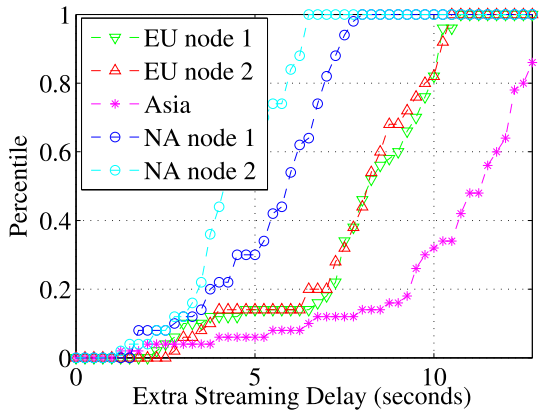


Fig. 13. Extra streaming delay measured at each regional server (CDF).

VI. Prototype and Experimental Study

We further conduct a PlanetLab-based experiment and a real-world experiment to see the practical performance of our design and the actual end-viewer transcoding performance.

A. Prototype Implementation and Measurement

We implemented a prototype of our system on the PlanetLab. In our prototype, we use 5 PlanetLab nodes with similar network conditions as regional servers, 2 in North America (NA), 1 in Asia and 2 in Europe (EU), and other nodes as viewers. In total 213 nodes are used. During the experiment, each viewer node imitates an actual viewer behavior by joining the nearest regional server at a random time, staying for an duration according to the Pareto distribution, and leaving. The above operation is independent from other viewer nodes, and will be conducted repeatedly. The regional servers maintains the stable candidate pool for its region. The most preferred candidate is selected to use *ffmpeg* to transcode a high quality video sent from the regional server using TCP, into a low quality version, which is then sent back to the server. We use a 3.5 Mbps 1080p video as our source video and set 2.5 Mbps 720p as the target quality. Each video slice is of 1 second. Note that there are actually more parameter settings for transcoding besides resolution and bitrate, among which the "speed" property affects the transcoded video quality the most. In *ffmpeg* the "speed" can be set from "veryslow" to "ultrafast". During our experiment, we choose a conservative strategy to use the default "median" option for speed to have a relative good video quality without sacrificing too much time. In reality however, this parameter should be decided by the nature and focus of live channel, i.e., whether the clear frame is more important, or the short interactive latency.

We first measured the extra streaming delay of all the nodes under the corresponding regional server, shown in Fig. 13. We see the delay difference varies from region to region. All Video sources from NA have the extra delay less than 8 seconds, while sources from other regions have higher extra streaming delay. The variance indicates we need to have a pre-set delay to cope

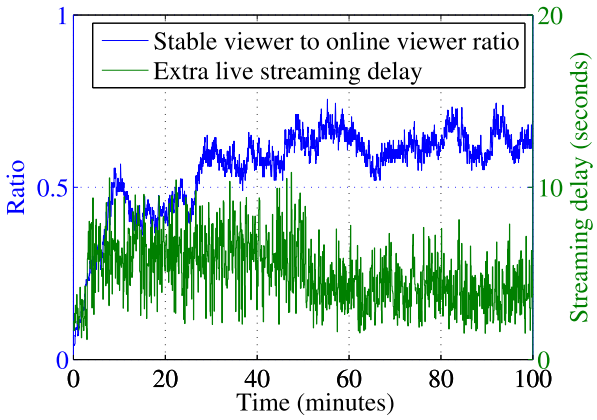


Fig. 14. Stable viewer to online viewer ratio and extra streaming delay over time.

with the delay change caused by the reassignment. We then focus on an experiment conducted on a NA regional server. As shown in Fig. 14, the blue line is the stable viewer to all online viewer ratio, oscillating around 50% to 60%, which is consistent with our observation on Twitch data. The green line indicates the extra live streaming delay measured by recording the transmission and transcoding time of every 1-second video slice. Clearly, the extra delay time changes dramatically at time 3.5 minutes and 51 minutes, which is caused by two reassignment events.

The experiment shows our system can run smoothly, but it also indicates the existence of sudden change of streaming delay due to reassignment events, which calls for having a pre-set delay to prevent playback stall cause by such events.

B. End-Viewer Transcoding Experiment

To better understand the transcoding ability of real end-viewers instead of PlanetLab virtual machines, we conducted another experiment with 8 devices of different CPU types. These 8 CPUs are popular representatives that power up most home computers and personal PCs in the current market. We use *VLC* to do H.264 transcoding for a 1080p video (3.5 Mbit/s) to lower quality versions while the device is playing a live streaming at *Source* quality from a channel on Twitch TV. Similar to the previous prototype experiment, we also use the default option for speed and other transcoding parameter settings. For comparison, we also measured the transcoding time of the 720p video quality when the device was idle (denoted as 720p*). Some devices were equipped with dedicated GPUs and we note them as "[*]" after the CPU name.

Table II shows the experiment results in form of transcoding time to video playback time ratio. The device is capable of doing corresponding real-time transcoding only if the ratio is below 100%. All processors except Intel Core 2 Duo are able to handle real-time transcoding for 480p or lower quality versions. Transcoding for 720p version however is more computationally intense, as only top 3 devices manage to proceed in real time. Notably, during the experiment, the viewing QoE of all devices is almost not affected, and the lack of computational ability is mainly revealed by the long transcoding time. The experimental results confirm the real-time transcoding ability of modern CPUs, and it also suggests to distinguish different unqualified viewers, some of whom can be helpful for transcoding lower quality versions.

VII. Further Enhancements

Regarding to the practical performance of our framework, we next present three enhancements that further improve the system performance under diverse conditions.

A. Categorizing Median Viewers and Enabling Collaborative Transcoding

In VCTS, we set strict threshold on viewer computational ability and networking performance, and therefore a considerable portion of median viewers are not qualified as candidates. These median viewers refer to those who are capable of watching at source resolution and still have idle computational and networking resource, though the extra resource is not enough to handle one complete transcoding assignment for a high target quality version, such as 720p. In fact, as we see from our realworld experiment, some viewers are capable of transcoding lower quality versions except for the 720p. Though being unqualified, these median viewers can contribute for most lower quality versions, or even high quality versions if they collaboratively use their idle resource. On the other hand, due to the dynamics of the system, it is possible that in certain region the availability of qualified stable viewers suffers overrun, causing the increase of cross-region assignments or even failed assignments.

To deal with such system dynamics, we can categorize these median viewers into different levels in terms of their idle resource, and use additional secondary B+ trees to organize median viewers from each level. The VCTS thus can be modified such that if the size of the primary B+ tree (candidate pool P) is below certain level, median viewers in secondary B+ trees can be assigned for transcoding assignment of lower quality versions; If the primary B+ tree suffers overrun, N of median viewers will be chosen to collaboratively transcode one target version, where N relates to the category level of selected viewers. In the latter case, each selected viewer works on a smaller video slice independently and returns the transcoded slice to the regional server, where the video slices are reordered and forwarded to viewers/CDNs all over the world.

B. Hybridization With Dedicated Servers to Reduce Streaming Latency

The PlanetLab-based experiment shows the additional streaming latency caused by deploying VCTS. Although the extra latency is small and acceptable, due to the critical requirement of livecast, any further optimization on video latency is beneficial. Our framework can be hybridized with dedicated transcoding servers. The original video stream is then divided into $slice_A$ and $slice_B$ iteratively instead of single slices of same length. Once the $slice_A$ is created, it is forwarded to the assigned viewers for transcoding, and after a while, when the $slice_B$ is generated, it is sent to the dedicated servers for

TABLE II
TRANSCODING TIME TO VIDEO PLAYBACK TIME RATIO OF DIFFERENT DEVICES WHILE PLAYING LIVE STREAMING AT *Source* QUALITY

CPU type	720p* 2.5 Mbit/s	720p 2.5 Mbit/s	480p 1.2 Mbit/s	360p 0.8 Mbit/s	240p 0.5 Mbit/s
Intel i7-3770 @3.40 GHz × 4	33.7%	59.6%	25.0%	17.5%	14.5%
Intel i7-3630QM @2.4 GHz × 4 [*]	45.5%	58.2%	33.1%	24.7%	19.7%
Intel i5-2400 3.1 GHz × 2	53.5%	66.7%	38.4%	27.8%	19.2%
Intel i5-3210M 2.50 GHz × 2	90.6%	113%	68.6%	43.1%	34%
Intel i5-4250U 1.3 GHz × 2	116.3%	191.5%	92%	70.8%	51.2%
AMD a10-4600M 2.3 GHz × 2 [*]	104.3%	143.3%	77.5%	59.4%	48.2%
Intel i3-2310M 2.10 Ghz × 2 [*]	130.0%	155.8%	90.0%	60.3%	44.4%
Intel Core 2 Duo 2.53 GHz × 2	86.7%	190.5%	171.1%	112.3%	76.9%

processing. When both pieces are processed and returned, they are combined back as one single video slice and delivered to end-viewers. Since the data transfer time and video transcoding time are usually longer for the transcoding viewers than the dedicated servers, the ideal situation is the processed $slice'_A$ and $slice'_B$ return at the same time, in which case the extra streaming delay of the hybridized VCTS is exactly the same as the transcoding system with dedicated servers. If $slice_B$ is much smaller than $slice_A$, then the dedicated resource required in this hybridized system is very small. In addition, such hybridization is only needed for higher quality target versions (such as 720p, 480p), since streaming latency of lower quality versions transcoded by viewers is shorter and even less than the optimized latency of high quality versions. This further decreases the dedicated resourced needed.

Hybridization with dedicated transcoding servers can efficiently decrease the livecast latency, at only small expense of dedicated resource, though we do need to carefully consider the decision on length selection for $slice_A$ and $slice_B$, and the overhead caused by any extra video dividing/combining.

C. Community Interaction and Synchronization

As mentioned, a new trend in the crowdsourced livecast systems is that the channels-based communities of fellow viewers have emerged as a platform for communicating with each other, as well as interacting with the broadcaster. The communication in these communities are mostly done through text messages, which can be easily transferred, processed, and posted in realtime. The latency of the live streaming itself however varies from viewer to viewer due to the difference of their connection to the regional servers/CDNs. With VCTS being deployed, the variation of end-to-end latency becomes even more dramatic, as the processing time of each target version varies given the heterogeneous transcoding viewers. Often the case, a much more delayed viewer comments on a scene watched by others a while ago. This issue becomes more severe when it comes to some channels in which the game is collaboratively played by all the viewers. For example, in the channel Twitch Plays Pokemon,[9] fellow viewers play the game Pokemon together by typing commands into chat. If the delays of the video streams vary largely, viewers will have unsynchronized game information and thus give misleading command.

[9][Online]. Available: http://www.twitch.tv/twitchplayspokemon/profile

The existing Twitch implementation does not address the out-of-sync issue, which substantially affect the viewers' iterative experience [14]. We observe that except the synchronization of different quality versions, the video rate also plays a key role in controlling the live streaming latency, since the initial latency time is mainly decided by the buffering time, and every time the video discontinues due to an out-of-buffer event, the re-buffering time adds to the latency. We want to adapt the video rate so that the live streaming is synchronized at all end viewers. However, since each end viewer has different local time, and even viewers located in the same timezone have different system time due to the heterogeneity, we unfortunately cannot use the viewer's time to synchronize the video frame. We observe the time stamp on each chatting message can be regarded as a fair reference due to its low latency. To this end, we propose a revised DASH mechanism with text messaging based synchronization. In this approach, both the video slice and chat message will be marked with the instant time of the regional server. A reasonable delay time d is set such that the video slice with time stamp t should be played after d second of getting chat message with time stamp t. The client thus adapts the video quality version based on such synchronization mechanism. The delay time d should also be dynamically adjusted according to the viewer feedback of their network conditions.

As such, despite of the difference in video processing time and network configurations, heterogeneous end-viewers can better synchronize their live streaming with other peer-viewers.

VIII. CONCLUSION

In this paper, we examined the emerging crowdsourced livecast systems, in which both the broadcasters and the viewers are in massive number and can be highly heterogeneous. Through our measurement on the major crowdsourced livecast system, on the one hand, we see the need to unify the source video and to provide multiple industrial standard quality versions for diverse viewers, on the other, we also find huge potential on the massive viewer side for video transcoding. We presented a generic framework CrowdTranscoding with VCTS that can smartly assign qualified stable viewers to channels for transcoding assignment. Our trace-driven simulation proved the superiority of our design, while our PlanetLab-based experiment and end-viewer transcoding experiment further revealed a series of potential issues and suggested corresponding enhancements toward practical deployment.

In the future work, we plan to conduct experiments in larger scales. We are also interested in exploring the hybridization of CrowdTranscoder with the dedicated transcoding servers to improve the streaming performance, and in particular to explore the optimal solution to dynamically fragment video slices in order to have shortest end-to-end streaming latency. There is also need to study the dynamic pattern of viewers and broadcasters inside each region, after which we can further study the strategy for adapting viewer qualifying criteria, given the viewer-to-channel ratio and workload change dynamics.

ACKNOWLEDGMENT

The findings achieved herein are solely the responsibility of the authors.

REFERENCES

[1] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers?: Shifting demographics in mechanical Turk," in *Proc. CHI'10 Extended Abstracts Human Factors Comput. Syst.*, 2010, pp. 2863–2872.

[2] T. Hoßfeld *et al.*, "Best practices for QoE crowdtesting: QoE assessment with crowdsourcing," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 541–558, Feb. 2014.

[3] F. Chen, C. Zhang, F. Wang, J. Liu, X. Wang, and Y. Liu, "Cloud-assisted live streaming for crowdsourced multimedia content," *IEEE Trans. Multimedia*, vol. 17, no. 9, pp. 1471–1483, Sep. 2015.

[4] M. Kaytoue, A. Silva, L. Cerf, W. Meira, Jr., and C. Raïssi, "Watch me playing, i am a professional: A first study on video game live streaming," in *Proc. 21st Int. Conf. Companion World Wide Web*, 2012, pp. 1181–1188.

[5] 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 Trans. Multimedia*, vol. 18, no. 5, pp. 916–928, May 2016.

[6] "2015 Retrospective—Twitch.," Accessed on Nov. 11, 2016. [Online]. Available: https://www.twitch.tv/year/2015

[7] W. A. Hamilton, O. Garretson, and A. Kerne, "Streaming on Twitch: Fostering participatory communities of play within live mixed media," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 1315–1324.

[8] S. Traverso *et al.*, "Unravelling the impact of temporal and geographical locality in content caching systems," *IEEE Trans. Multimedia*, vol. 17, no. 10, pp. 1839–1854, Oct. 2015.

[9] M. Mu *et al.*, "P2P-based IPTV services: Design, deployment, and QoE measurement," *IEEE Trans. Multimedia*, vol. 14, no. 6, pp. 1515–1527, Dec. 2012.

[10] G. Zhang, W. Liu, X. Hei, and W. Cheng, "Unreeling Xunlei Kankan: Understanding hybrid CDN-P2P video-on-demand streaming," *IEEE Trans. Multimedia*, vol. 17, no. 2, pp. 229–242, Feb. 2015.

[11] F. Wang, J. Liu, and M. Chen, "CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities," in *Proc. INFOCOM*, 2012, pp. 199–207.

[12] X. Wang, M. Chen, T. T. Kwon, L. Yang, and V. C. Leung, "AMES-cloud: A framework of adaptive mobile video streaming and efficient social video sharing in the clouds," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 811–820, Jun. 2013.

[13] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," *IEEE Trans. Multimedia*, vol. 15, no. 4, pp. 870–883, Jun. 2013.

[14] C. Zhang and J. Liu, "On crowdsourced interactive live streaming: A twitch. TV-based measurement study," in *Proc. 25th ACM Workshop Netw. Operating Syst. Support Digital Audio Video*, 2015, pp. 55–60.

[15] "CPU Popularity," Accessed on Oct. 1, 2016. [Online]. Available: http://www.cpubenchmark.net/share30.html

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput*, 2012, pp. 13–16.

[17] S. Basso, A. Servetti, E. Masala, and J. C. De Martin, "Measuring dash streaming performance from the end users perspective using neubot," in *Proc. 5th ACM Multimedia Syst. Conf.*, 2014, pp. 1–6.

[18] "Twitch traffic and demographic statistics," Accessed on Aug. 20, 2016. [Online]. Available: https://www.quantcast.com/twitch.tv?country=MY#!countries

[19] B. Li, Z. Wang, J. Liu, and W. Zhu, "Two decades of internet video streaming: A retrospective view," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1s, 2013, Art. no. 33.

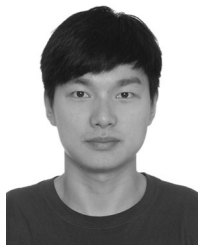
[20] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube traffic characterization: A view from the edge," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas.*, 2007, pp. 15–28.

[21] X. Cheng, J. Liu, and C. Dale, "Understanding the characteristics of Internet short video sharing: A YouTube-based measurement study," *IEEE Trans. Multimedia*, vol. 15, no. 5, pp. 1184–1194, Aug. 2013.

[22] K. Sripanidkulchai, B. Maggs, and H. Zhang, "An analysis of live streaming workloads on the internet," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, 2004, pp. 41–54.

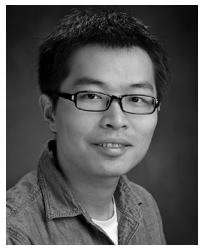
[23] F. Dobrian *et al.*, "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2011, pp. 362–373.

[24] S. S. Krishnan and R. K. Sitaraman, "Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 2001–2014, Dec. 2013.



Qiyun He (S'16–M'16) received the B.Eng. degree from Zhejiang University, Zhejiang, China, and the B.Sc. degree from Simon Fraser University, Burnaby, BC, Canada, both in 2015, and is currently working toward the M.Sc. degree in Computing Science at Simon Fraser University.

His research interests include cloud computing, social media, multimedia systems, and networks.



Cong Zhang (S'14) received the M.S. degree in information engineering from Zhengzhou University, Zhengzhou, China, in 2012, and is currently working toward the Ph.D. degree in computing science at Simon Fraser University, Burnaby, BC, Canada.

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.

Jiangchuan Liu (S'01–M'03–SM'08–F'17) received the B.Eng. (*cum laude*) degree from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree from The Hong Kong University of Science and Technology, Hong Kong, China, in 2003, both in computer science.

He is a University Professor with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada.

Prof. Liu is an NSERC E.W.R. Steacie Memorial Fellow. He is also an EMC-Endowed Visiting Chair Professor of Tsinghua University. He is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, the IEEE TRANSACTIONS ON BIG DATA, and the IEEE TRANSACTIONS ON MULTIMEDIA. He was the recipient of the inaugural Test of Time Paper Award of the IEEE INFOCOM in 2015, the *ACM SIGMM TOMCCAP* Nicolas D. Georganas Best Paper Award in 2013, and the ACM Multimedia Best Paper Award in 2012.