

Crowdsourced Live Streaming over the Cloud

Fei Chen^{*†}, Cong Zhang^{*}, Feng Wang[‡], Jiangchuan Liu^{*}

^{*}School of Computing Science, Simon Fraser University, Canada

[†]School of Digital Media, Jiangnan University, China

[‡]Department of Computer and Information Science, The University of Mississippi, USA

Email: chenf@jiangnan.edu.cn, congz@sfu.ca, fwang@cs.olemiss.edu, jcliu@cs.sfu.ca

Abstract—Empowered by today’s rich tools for media generation and distribution, and the convenient Internet access, *crowdsourced streaming* generalizes the single-source streaming paradigm by including massive contributors for a video channel. It calls a joint optimization along the path from crowdsourcers, through streaming servers, to the end-users to minimize the overall latency. The dynamics of the video sources, together with the globalized request demands and the high computation demand from each sourcer, make crowdsourced live streaming challenging even with powerful support from modern cloud computing. In this paper, we present a generic framework that facilitates a cost-effective cloud service for crowdsourced live streaming. Through adaptively leasing, the cloud servers can be provisioned in a fine granularity to accommodate geo-distributed video crowdsourcers. We present an optimal solution to deal with service migration among cloud instances of diverse lease prices. It also addresses the location impact to the streaming quality. To understand the performance of the proposed strategies in the realworld, we have built a prototype system running over the planetlab and the Amazon/Microsoft Cloud. Our extensive experiments demonstrate that the effectiveness of our solution in terms of deployment cost and streaming quality.

I. INTRODUCTION

The Internet has witnessed a significant increase in the popularity of media streaming with multi-source channels. In traditional video broadcast, the content of a channel generally comes from a single source, though it could be replicated and then streamed from different servers in a Content Distribution Network (CDN). A *multi-source* system, however, not only serves massive audience worldwide, but its content also comes from multiple contributing sources. For example, since Feb. 17, 2012, NASA Television’s Public and Media channels began to transmit their respective content in high definition (HD), with live feeds from such space centers as the NASA Headquarters, the Johnson Space Center, and the Goddard Space Flight Center ¹. With their respective content sources, they collectively serve the users interested in the stories and the latest news from NASA. In the very recent 2014 Sochi Winter Olympics, NBC had a total of 41 live feeds distributed both in Solchi and in the USA ², and in FIFA World Cup 2014, when a goal was scored, CBC synchronized the live scenes of the cheering fans in the public squares from a number of cities worldwide in its live streaming channel. The evolution is driven further by today’s advanced mobile/tablet devices that can readily capture high quality video anywhere and anytime

(e.g. iPhone 6 supports both 60 fps 1080p video recording, and 720 fps slow-motion recording for 720p videos), and such mainstream video sharing platforms as YouTube and Veedme have already enabled multi-party collaborative video content production. All these together are shifting the video service paradigm from the conventional single source, to multi-source, to many source, and now toward *crowd source*, in which the available video sources for the content of interest become highly diverse and scalable.

Global streaming imposes high demand on end device capabilities and network connections. The situation is further complicated in a crowdsourced streaming system. First, crowdsourced videos are geo-distributed: they come from all over the world, and then spread all over the world. Not only the scale of the consumers is enormous, but also is that of the contributors; Second, the crowdsourcers are often much more dynamic than dedicated content providers, as they can start or terminate a video contribution as their own will. This is particularly true when non-professionals using their smartphones/tablets for video production; Third, for collective content production, massive server capacity is necessary to deal with online video synchronization, processing, and transcoding for highly heterogeneous video contributors and consumers. For example, Twitch TV, the world’s leading video platform and community for gamers ³, allows any of its users to broadcast their live streaming videos online through their PCs or PS3/XBOX consoles. It attracts over 44 million visitors per month, and every second its servers are loaded with thousands of live channels. For such a large system, significant effort is needed to collect the highly dynamic and distributed video streams online, and to process and distribute the live channels to subscribers all over the world.

Elastic resource provisioning and computation offloading are where *cloud computing* platforms excel [10]. We have seen many new generation of cloud-based multimedia services that emerged in recent years, e.g., Netflix, which are rapidly changing the operation and business models in the market. Facing similar scale challenges, crowdsourced live streaming would benefit from the cloud services, too. Yet the distributed and highly dynamic sources, as well as the much more stringent delay constraints imposed by live streaming, make the problem more involving, which remains to be explored with novel and distinct solutions.

¹<http://www.nasa.gov/multimedia/index.html>

²<http://www.istreamplanet.com/sochi-2014/>

³<http://www.twitch.tv/>

In this paper, using realworld measurement, we identify the potential benefits as well as the key challenges when crowdsourced video meets cloud. We present a generic framework for a cost-effective cloud service that provisions cloud resources in a fine granularity to work with geo-distributed video crowdsourcers. Using adaptive and collaborative leasing strategies, our design well accommodates the diverse capacities and prices of cloud instances, and addresses the location impact to the streaming quality. We have built a prototype system running over the Internet and the Amazon EC2/Microsoft Azure cloud, and the experimental results experiments demonstrate the effectiveness of our solution in terms of both deployment cost and streaming quality.

The remainder of this paper proceeds as follows. Section 2 discusses the background and related work. Section 3 presents an overview of the crowdsourced live streaming system, and analyzes its unique challenges using realworld data traces. In Section 4, we first investigate the inherent problem of cloud leasing strategy. An optimal solution is then developed to deal with geo-distributed crowdsourcers in Section 5. In Section 6, we present a prototype platform with the measurement results and the trace-driven simulation. Finally, Section 7 concludes the paper and discusses potential future directions.

II. BACKGROUND AND RELATED WORK

In the past two decades, video streaming over the Internet has quickly risen to become a mainstream "killer" application [2]. For large scale distribution, many existing systems rely on content distribution networks (CDNs) [11] or peer-to-peer (P2P) [1], or hybrid solutions [8]. More recently, with the flexible and elastic resource provisioning, cloud computing has been proven to be an efficient solution toward highly scalable video distribution. A prominent example is Netflix, a major on-demand Internet video provider. Netflix migrated its entire infrastructure to the powerful Amazon AWS cloud in 2012, using EC2 for transcoding master video copies to 50 different versions for heterogeneous end users and S3 for content storage [11]. In total, Netflix has over 1 petabyte of media data stored in Amazon's cloud. It leases the computation, bandwidth and storage resources with much lower long-term costs than those with over-provisioned self-owned servers, and reacts better and faster to user demand with the dramatically increasing scale. There have been pioneer studies on migrating video services to the cloud to accommodate worldwide-distributed and time-varying video demands [10] [2]. Aggarwal et al. [13] showed that the cost of IPTV services can be noticeably reduced through a cloud infrastructure, and Wu et al. [10] utilized a geo-distributed cloud to support large scale social media streaming applications. Wang et al. [8] presented CALMS (Cloud-Assisted Live Media Streaming) to lease and adjust cloud server resources in a fine granularity, meeting with the temporal and spatial dynamics of demands from online users.

Empowered by today's rich tools for media generation and collaborative production, and the convenient Internet access, *crowdsourcing* further extends the single-source paradigm. It

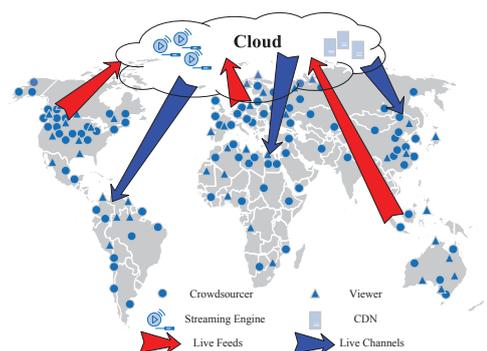


Figure 1: A generic crowdsourced live streaming system over cloud

combines the efforts of multiple self-identified contributors, known as *crowdsourcers*, for a greater result, and has seen success in many areas [3]. For example, LiFS (Locating in Fingerprint Space) was developed for wireless indoor localization with smartphones based crowdsourcing [4]. Ou et al. [5] used crowdsourcing approach to optimize mobile devices' energy efficiency by utilizing signal strength traces shared by other devices in cellular networks. For video applications, a scalable system that allows users to perform content-based searches on continuous collection of crowdsourced video was proposed in [7]. Biel et al. [6] investigated the the crowdsourcing of personal and social traits in online social video or social media content in general. Recently, Youtube has integrated with Google Moderator, a crowdsourcing and feedback production, to increase the engagement between viewers and content creators. Such other video sharing sites as Poptent and VeedMe have also opened interfaces for crowdsourcers with user generated content. Crowdsourced live streaming services have emerged in the market as well, especially for streaming sports online broadcast. Examples include Stream2Watch.me and sportLEMON.tv.

Our study is motivated by these pioneer works. Yet crowdsourced live streaming demands efficient content collection, processing, and distribution with stringent delay constraints, which remain to be explored. This paper highlights these unique challenges, particularly when crowdsourced live streaming meets cloud, and presents our initial attempts toward addressing these challenges.

III. CROWDSOURCED LIVING STREAMING: SYSTEM OVERVIEW AND CHALLENGES

We illustrate a generic crowdsourced live streaming system with geo-distributed *crowdsourcers* and *viewers* in Fig. 1. A set of crowdsourcers (or *sourcers* in short) upload their individual video contents in realtime, which, through a video production engine, collectively produce a single video stream. The stream is then lively distributed to viewers of interest. Both the sourcers and viewers can be heterogenous, in terms of their network bandwidth, and their hardware/software configurations for video capture and playback. As such, realtime transcoding is necessary during both uploading and downloading, so as to unify the diverse video bitrates/formats from

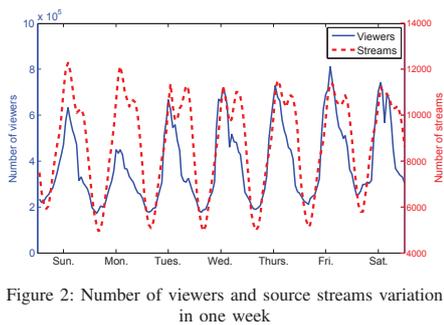


Figure 2: Number of viewers and source streams variation in one week

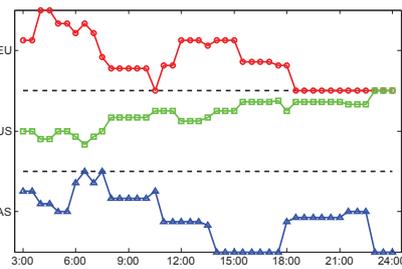


Figure 3: Source stream distribution in one day

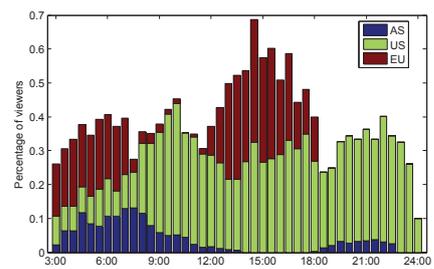


Figure 4: Viewer demand for the distributed source streams in one day

Table I: Top 5 sourcers from Twitch.tv on July, 12th

Sourcers ID	Time (Pacific Time)	Location
<i>riotgames</i>	11:10 AM-15:40 PM	Cologne, Germany
<i>dota2ti_ru</i>	7:10 AM-18:10 PM	Seattle, USA
<i>srkevo1</i>	6:00 AM-23:40 PM	Las Vegas, USA
<i>riotgamesturkish</i>	1:30 AM-7:10 AM	Istanbul, Turkey
<i>ongamenet</i>	3:00 AM-13:30 PM 18:20 PM-22:40 PM	Seoul, South Korea

different sourcers for content production, and to replicate the output video stream to serve the heterogeneous viewers, possibly through a CDN with such adaptation mechanisms as DASH (Dynamic Adaptive Streaming over HTTP) [12].

This generic architecture reflects that of state-of-the-art realworld systems. For example, NBC's video content from the 41 feeds in Sochi Winter Olympics were encoded by Windows Azure Media Services to the 1080P format, and dynamically transcoded into HLS and HDS formats. These streams were then pulled from Azure to the Akamai's CDN and distributed to audiences on targeted devices, resulting in over 3000 hours of live Olympics streaming contents.

Given the large system scale and the high bandwidth, storage, and computation demands involved, cloud services with elastic resource provisioning is expected. We again consider a generic geo-distributed cloud infrastructure, which consists of multiple *cloud sites* distributed in different geographical locations (e.g., US East (N. Virginia) and EU (Ireland) in Amazon EC2 Cloud)[10]. Each cloud site resides in a data center, and contains a collection of interconnected and virtualized servers. The server resources will be provisioned for crowdsourced live streaming, e.g., computation resources for collective production and transcoding.

Optimization for conventional single-source video streaming is generally *viewer-driven*; the resource provisioning depends on the distribution of the viewers. In crowdsourced video, however, the sourcers themselves come from all over the world, whose distribution must be as well taken into account during resource provisioning. This is further aggravated given that the collaborative production escalates the demands on both bandwidth and computation. The crowdsourced streaming workflow is also much more dynamic, as individual sourcers can start/terminate based on their own schedules.

To better understand the inherent challenges of deploying such a system, we have crawled one-week trace from July

6 to July 12, 2014 in Twitch.tv website, which has 14 geo-distributed ingest servers, 1 from Asia area (AS for short), 6 from European area (EU for short), and 7 from United States area (US for short) to broadcast live game streams to viewers in a global scale. For simplicity, we consider that one live stream is contributed by only one sourcer. Fig. 2 shows the number variation of viewers and streams in a week, from July 6 to July 12, 2014. First, it is obvious that the number of viewer is highly dynamic, which is prevalent in current large scale systems [2]. Due to the differences in time zones and languages, the distribution of viewers can be time-varying, which has been discussed in previous works [9] [8]. Similar to the number of viewers, we can see that the number of source streams also has great time variations in one-day time, from about 5000 streams in the early morning to almost 12000 streams in the afternoon. To further investigate the time-varying distribution of the source streams, we have measured the top 15 streams with the highest viewer population from 3:00 AM to 24:00 PM (PST) on July 12, 2014, and list the five most popular streams in Table 1. We can see that not only the time periods but also the locations of the stream sourcers are highly dynamic. In Fig. 3, we divide the locations as AS, EU, and US, and record the percentage of source streams from each region for every 30 minutes between 3:00 AM to 24:00 PM. It can be easily observed that most of the streams from Asia and Europe are during the morning and afternoon, and the number of streams from the United States keeps growing when night falls. We further measure the viewer population for the distributed source streams from each region in Fig. 4. We can see that in the early morning between 3:00 AM and 7:00 AM, most of the popular streams come from Europe or Asia. We conjecture that it is because the local times in Europe or Asia are in afternoon or evening, and there are more online sourcers from these regions during that time. Meanwhile, the viewer demand from these areas can also be more active during this period. And most of the viewers may prefer the streams with native language speaking sourcers. Similar reasons can also explain the increase of viewer demand for the source streams from the United States after 15:00 PM.

In summary, in a crowdsourced live streaming system, both the number and the distribution of the crowdsourcers can be highly dynamic. Together with time-varying viewer demand, the conventional server allocation design faces more

challenging in a large scale. We will utilize the cloud service to coordinate the crowdsourcers and viewers. The cloud server instances (e.g. EC2 in Amazon Cloud) are provisioned to collect and process the live feeds of the crowdsourcers, and the cloud CDNs (e.g. CloudFront in Amazon Cloud) are deployed to handle the viewer dynamics. Through dynamic cloud leasing, we will present a cost-effective solution with streaming quality guarantee.

IV. CLOUD-ASSISTANCE FOR CROWDSOURCED LIVE STREAMING

In this section, we first model the global cloud service leasing strategy with quality guarantee, and transform it into an equivalent problem in a directed graph. We will then present an optimal algorithm and an efficient online heuristic solution based on the equivalent problem.

A. Problem Formulation

We use \mathbb{A} to denote the global areas, which can be divided into n different regions as $\mathbb{A} = \{A_1, A_2, \dots, A_n\}$. Assume that there are m cloud sites all over the world, represented as $\mathbb{S} = \{s_1, s_2, \dots, s_m\}$. As most cloud providers have a minimum unit time for the duration of leasing a server (e.g. 1 hour for Amazon EC2), we use T to denote this duration. We define a *time slice* as an integer multiple κ ($\kappa \in \mathbb{N}^+$) of T and at the beginning of each time slice κT , our cloud leasing strategy makes decisions on whether to provision or terminate the cloud servers in the distributed regions. We assume that the schedules of crowdsourced streams are predictable and can be known beforehand, where the rationale is of two folds. First, in practice a large portion of crowdsourced streams are driven by well-scheduled events (e.g. as one of the top 5 sourcers from Twitch.tv in Table I, the channel of *srkevo1* has a strict schedule about the Evolution 2014 Tournament⁴). Moreover, many self-motivated crowdsourcers prefer a regular broadcast schedule everyday to attract more viewers. We can accordingly forecast the numbers and distributions of both crowdsourcers and viewers for the next time slice, e.g., using techniques from [8][14].

For a given time t , we denote the set of source streams from the crowdsourcers as $\mathbb{L}(t)$. According to the location distribution of crowdsourcers, we can specify the set as $\mathbb{L}^A(t) = \{l_{A_1}(t), l_{A_2}(t), \dots, l_{A_n}(t)\}$ for the n different regions, respectively. As all these live streams are served by the provisioned cloud instances, we further consider the set according to the dedicated cloud sites as $\mathbb{L}^s(t) = \{l_{s_1}(t), l_{s_2}(t), \dots, l_{s_m}(t)\}$, where $l_{s_j}(t)$ represents the live streaming sources loaded in cloud site s_j . For example, if $l_{s_j}(t) = \emptyset$, no crowdsourced stream is served by cloud site s_j , i.e., cloud site s_j does not need to be leased at time t . Otherwise, if the live streams from area A_2 , A_3 , and A_5 are served by cloud site s_j , we have $l_{s_j}(t) = l_{A_2}(t) \cup l_{A_3}(t) \cup l_{A_5}(t)$.

We denote the server provisioning cost at time t as $C^p(t) = \sum_{j=1}^m c_j^p(l_{s_j}(t))$, where c_j^p is the price of the leased instances

in cloud site s_j . We assume that there is always a bootstrapping server s_0 redirecting the global live sources to the distributed streaming servers with the cost c_0 . To offload the bandwidth support for the diverse viewer demands from the cloud servers, a globalized CDN strategy (e.g., CloudFront in Amazon) is deployed to distribute the live streams all over the world. The cost of out-bound traffic from the cloud servers to the CDN can be calculated by the number of channels loaded in the cloud servers, and denoted as $C^b = \sum_{j=1}^m c_j^b(l_{s_j}(t))$. As the cost of the bandwidth support from the CDN to the global viewers is proportional to the viewer demands $D(t) = \sum_{i=1}^n D_{l_{A_i}}(t)$, where $D_{l_{A_i}}(t)$ represents the viewer demands for the crowdsourced streams from region A_i , we can denote the total cost of the CDN as $C^d = c^d(D(t))$ with c^d as the cost to support one unit of the viewer demand. The total cost of the crowdsourced live streaming system can thus be calculated as follows:

$$\begin{aligned} Cost_{total} &= c_0 + C^p + C^b + C^d \\ &= c_0 + \sum_{j=1}^m [c_j^p(l_{s_j}(t)) + c_j^b(l_{s_j}(t))] + c^d(D(t)) \\ &= c_0 + \underbrace{\sum_{j=1}^m c_j(l_{s_j}(t))}_{Cost_{lease}} + c^d(D(t)) \end{aligned}$$

where $c_j(\cdot)$ can be determined by the price policy of instance leasing and data traffic in cloud site s_j . As the first and last costs on the right side of the equation can not be reduced, we focus on minimizing the middle part of the total cost, i.e., the cloud leasing cost, which we denote as $Cost_{lease}$.

We assume that the live crowdsourcers in each region $l_{A_i}(t)$ have a preference value on a given cloud site s_j , which we denote as $P(l_{A_i}(t), s_j)$. Generally, the preference value can be quantified according to the RTT, jitter or packet loss values of the connections between the crowdsourcers and the given cloud site, such as defined as a concave decreasing function of the estimated latency or a concave increasing function of the estimated connection speed in a geo-distributed service [9]. To guarantee the streaming quality of the crowdsourced streams in region A_i , we only consider allocating these streams to the cloud sites with the top k preference values, and define the set of these cloud sites as $Index(l_{A_i}(t), k)$ for the crowdsourced streams $l_{A_i}(t)$. As a real world example, Twitch/Justin.tv provides an ingest server ranker program to feedback the list with top 3 servers for each crowdsourcer.

The cloud service leasing problem in our geo-distributed crowdsourced live streaming system can thus be formulated as to find a cloud site leasing strategy \mathbb{L}^s , subjecting to the following constraints:

(1) Cloud site service constraint:

$$\begin{aligned} \forall A_i \in \mathbb{A}, \exists l_{s_j} \in \mathbb{L}^s, l_{A_i} \subseteq l_{s_j} \\ \forall l_{s_j}, l_{s_j} \in \mathbb{L}^s, \text{if } j \neq \hat{j}, l_{s_j} \cap l_{s_{\hat{j}}} = \emptyset \end{aligned}$$

(2) Crowdsourcer preference constraint:

⁴<http://evo2014.s3.amazonaws.com/brackets/index.html>

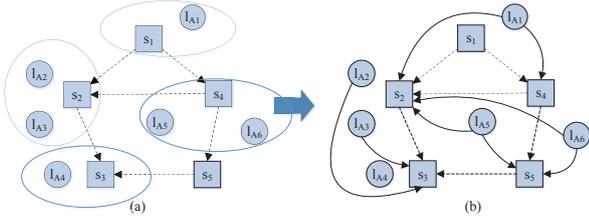


Figure 5: An illustrative example of (a) distribution graph; (b) service migration vectors

$$\forall A_i \in \mathbb{A}, s_j \in \mathbb{S}, \text{ if } l_{A_i} \subseteq l_{s_j} \\ s_j \in \text{Index}(l_{A_i}(t), k)$$

(3) Total budget constraint:

$$\text{Cost}_{lease} + c_0 + C^d \leq \text{Cost}_{max}$$

The cloud site service constraint states that the crowdsourced live streams in a given region are served by only one cloud site. The preference constraint guarantees that the crowdsourced live streams in each region are collected by one of the cloud sites with the corresponding top k preference values. The total budget constraint demands that the total cost including the bootstrapping server, the provisioned cloud sites and the CDN utilization must not exceed the total budget Cost_{max} . Our objective is to maximize the *global relative preference* of the crowdsourcers, which is defined as:

$$P_{global} = \frac{\sum_{\forall s_j \in \mathbb{S}, l_{A_i} \subseteq l_{s_j}} |D_{l_{A_i}}(t)| \cdot P(l_{A_i}(t), s_j)}{\sum_{\forall A_i \in \mathbb{A}} |D_{l_{A_i}}(t)| P(l_{A_i}(t), \text{Index}(l_{A_i}(t), 1))}$$

where for ease of exposition, we also use $\text{Index}(l_{A_i}(t), 1)$ to denote the top 1 preferred cloud site for the live crowdsourced streams $l_{A_i}(t)$. We use $|D_{l_{A_i}}(t)|$ to represent the size of viewer demands for crowdsourced streams $l_{A_i}(t)$, and P_{global} is thus a relative ratio ranged between (0, 1] in the global scale.

To make our solution cost-effective, we also need a second objective, i.e., to minimize the cloud leasing cost Cost_{lease} . It is easy to see that these two objectives (i.e., P_{global} and Cost_{lease}) may contradict with each other, since always leasing the top preferred cloud server can increase the leasing cost. Therefore, we adopt the following linear combination form to align them together by different weights:

$$\frac{p \cdot \text{Cost}_{lease}}{\text{Cost}_{max} - c_0 - C^d} + q \cdot (1 - P_{global})$$

where p and q are two parameters that can assign different weights to the two goals. As P_{global} is a relative ratio of the preference values of all the crowdsourcers in the system (i.e. if $P_{global} = 1$, all the crowdsourced live streams are allocated in their most preferred cloud sites), $(1 - P_{global})$ should be minimized as Cost_{lease} . To make the leasing cost part also be a ratio ranged between (0, 1], we further divide Cost_{lease} by $(\text{Cost}_{max} - c_0 - C^d)$ and then use parameters p and q to linearly combine the two parts together. In the next subsection,

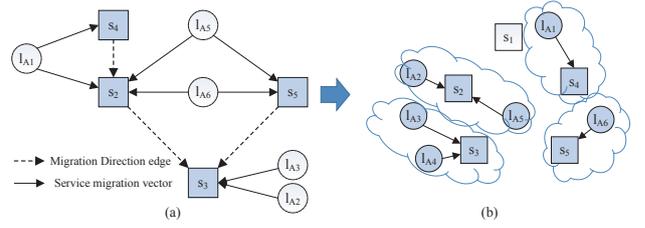


Figure 6: An illustrative example of (a) a constructed service migration graph; (b) migrated cloud service for geo-distributed crowdsourcers

we will transform this problem to an equivalent graph problem and then propose an optimal solution.

B. Equivalent Problem

For ease of exposition, we assume the given time is t for the remainder of this section and thus omit (t) in all such notations as $l_{A_i}(t)$, $l_{s_j}(t)$, $D_{l_{A_i}}(t)$, etc. Given the geo-distributed crowdsourcers and cloud sites, we can construct a *distribution graph*. Fig. 5(a) shows an example of 5 cloud sites and global crowdsourcers located in 6 regions. There are two types of vertices in the distribution graph, namely, the live crowdsourcers (e.g. l_{A_1}, \dots, l_{A_6} in Fig. 5(a)), which are represented by circles, and the cloud sites (e.g. s_1, \dots, s_5), which are represented by squares. Initially, all the live source steams are attached to their most preferred cloud sites and we denote the corresponding leasing cost as

$$\text{Cost}_{initial} = \sum_{\forall A_i \in \mathbb{A}, s_j = \text{Index}(l_{A_i}, 1)} c_j(l_{A_i})$$

According to the price strategy $c_j(\cdot)$ of different cloud site s_j , we have the *direction edges* between these distributed cloud sites. We use $\vec{d}(i, j)$ to denote a direction edge from the cloud site i with higher price to the cloud site j with lower price (e.g. in Fig. 5 (a), $\vec{d}(4, 2)$ means that $c_2(x) < c_4(x)$ for the same crowdsourcer x), which indicates that the service is migrating towards a more cost-effective solution.

With the distribution graph and direction edges, we then generate *service migration vectors* to indicate the available cloud sites for more cost-effective service migration. We use $\vec{m}(i, j)$ to denote a service migration vector that represents the live crowdsourcers l_{A_i} are migrated and served by the cloud site s_j , rather than the cloud site $\text{Index}(l_{A_i}, 1)$. For example, in Fig. 5(b), the cloud site s_4 is preferred by the live crowdsourcers l_{A_5} and l_{A_6} , i.e., $s_4 = \text{Index}(l_{A_i}, 1)$ for $i \in \{5, 6\}$. According to the direction edges $\vec{d}(4, 2)$ and $\vec{d}(4, 5)$, we can have the service migration vectors $\vec{m}(5, 2)$ and $\vec{m}(5, 5)$ for the live crowdsourcers l_{A_5} , and $\vec{m}(6, 2)$ and $\vec{m}(6, 5)$ for the live crowdsourcers l_{A_6} . Define M as the set of all service migration vectors that are generated from the given distribution graph. For each service migration vector $\vec{m}(i, j) \in M$, the *relative preference degradation* for live crowdsources l_{A_i} to be served by the cloud site j can be calculated as follows:

$$\text{Deg}(i, j) = \frac{|D_{l_{A_i}}| (P(l_{A_i}, \text{Index}(l_{A_i}, 1)) - P(l_{A_i}, s_j))}{\sum_{\forall A_i \in \mathbb{A}} |D_{l_{A_i}}| P(l_{A_i}, \text{Index}(l_{A_i}, 1))}$$

Also, for each $\vec{m}(i, j)$, we have the *cost saving* as follows:

$$Save(i, j) = c_j(l_{A_i}) - c_j(l_{A_i})$$

where c_j is the pricing policy of cloud site $s_j = Index(l_{A_i}, 1)$.

Traversing all the service migration vectors $\vec{m}(i, j) \in M$, we can have a *service migration graph* $G(V, E)$. Fig. 6(a) shows an example of Fig. 5(b). We connect the cloud sites with at least one service migration vector through migration direction edges. Note that there may be more than one migration direction edges leaving from the same cloud sites. For example, in Fig. 5(b) there are two migration direction edges $\vec{d}(4, 2)$ and $\vec{d}(4, 5)$ leaving from cloud site s_4 . Since the set of service migration vectors M has already been generated from the migration direction edges, we can put any one of these directed edges into the constructed service migration graph (which is only for the connectivity purpose that will be further explained in the next subsection). Finally, we connect the crowdsourcers to the cloud sites by the service migration vectors. In the constructed service migration graph $G(V, E)$, we can further define the *optimal service migration (OSM)* problem as to find a set of migration vectors $O \subseteq M$, subjecting to the following constraints:

(1) Service migration vector constraint:

$$\forall \vec{m}(i, j), \vec{m}(i, \hat{j}) \in M \text{ and } j \neq \hat{j}, \\ \text{if } \vec{m}(i, j) \in O, \text{ then } \vec{m}(i, \hat{j}) \notin O$$

(2) Preference degradation constraint:

$$\forall \vec{m}(i, j) \in O, s_j \in Index(l_{A_i}, k)$$

(3) Cost saving constraint:

$$Cost_{initial} - \sum_{\forall \vec{m}(i, j) \in O} Save(i, j) + c_0 + C^d \leq Cost_{max}$$

The service migration vector constraint represents that there is at most one migration vector leaving from a live crowdsourcer vertex, which corresponds to the cloud site service constraint in the cloud leasing problem. The preference degradation constraint is related to the crowdsourcer preference constraint of the cloud leasing problem. The cost saving constraint refers to the total cost not exceeding $Cost_{max}$ in the original problem. Our objective is to minimize the linear combination of cost saving and the relative preference degradation as follows:

$$\begin{aligned} & \frac{p}{Lease_{max}} (Cost_{initial} - \sum_{\forall \vec{m}(i, j) \in O} Save(i, j)) \\ & + q(1 - (1 - \sum_{\forall \vec{m}(i, j) \in O} Deg(i, j))) \\ = & \frac{p \cdot Cost_{initial}}{Lease_{max}} + \sum_{\forall \vec{m}(i, j) \in O} (q \cdot Deg(i, j) \\ & - \frac{p \cdot Save(i, j)}{Lease_{max}}) \end{aligned}$$

where $Lease_{max} = Cost_{max} - c_0 - C^d$. As $Cost_{initial}$ cannot be further reduced, our objective can thus be simplified as to minimize

$$\sum_{\forall \vec{m}(i, j) \in O} (q \cdot Deg(i, j) - \frac{p \cdot Save(i, j)}{Lease_{max}})$$

The OSM problem in graph $G(V, E)$ can be naturally related to the cloud site leasing problem: the optimal solution O indicates the service allocation for the crowdsourcers in each region toward the distributed cloud sites. Fig. 6(b) shows an example with $O = \{\vec{m}(1, 4), \vec{m}(3, 3), \vec{m}(5, 2), \vec{m}(6, 5)\}$. Therefore, we have the set of live crowdsourcers served in each cloud site as follows: $l_{s_1} = \emptyset$, $l_{s_2} = l_{A_2} \cup l_{A_5}$, $l_{s_3} = l_{A_3} \cup l_{A_4}$, $l_{s_4} = l_{A_1}$, and $l_{s_5} = l_{A_6}$.

V. OPTIMAL CLOUD LEASING STRATEGY

The optimal solution of the equivalent problem can be computed according to the spanning trees in the service migration graph. Clearly, a spanning tree is a subgraph of the directed graph $G(V, E)$. Let T denote the number of spanning trees in a service migration graph $G(V, E)$. We define the set of service migration vectors in the i -th spanning tree ($i \in \{1, \dots, T\}$) as M_i , and the optimal solution of M_i as O_i . We then have the following theorem:

Theorem 1. *There must exist an optimal solution O of the service migration vectors M in the service migration graph $G(V, E)$, such that $O \in \{O_1, \dots, O_T\}$.*

We can prove this using contradiction by assuming that there exists an optimal solution set of the service migration vectors O with edges in a circle. Then there are two scenarios if the edges in directed graph contain a circle: (1) The directed edges are sequenced in a line one after another, with the end vertex sending toward the head vertex. (2) There is more than one directed edge leaving from the same vertex. As there is no edge sending toward the live crowdsourcers vertex in directed graph $G(V, E)$, there would be cloud sites sequenced in a circle, and we have the confliction $c_1(l) > c_2(l) > \dots > c_{end}(l) > c_1(l)$. Also we can eliminate the scenario 2 according to the definition of service migration graph. Due to space limitation, here we omit the details of the proof, which can be found in our technical report presented.

According to Theorem 1, each spanning tree can provide a local optimal solution, and the global optimal solution can be achieved by exploring all the spanning trees in $G(V, E)$. There are extensive studies on enumerating all the spanning trees in a directed graph [15][16]. E.g., a well-known algorithm in [15] uses backtracking and a method for detecting bridges based on the depth-first search with the time complexity $O(|V| + |E| + |E| \cdot |T|)$ and the space complexity $O(|V| + |E|)$. For a spanning tree i in the service migration graph $G(V, E)$, the service migration vectors M_i (and each of its subsets) are feasible solutions under the service migration vector constraint. By enforcing the preference degradation constraint, a number of spanning trees can be further screened out. Thus, for a remained spanning tree i , we need to calculate

Algorithm 1: Optimal service migration()

```

1  $O = \emptyset$ 
2 for each enumerated spanning tree  $\lambda$  on  $G(V, E)$  do
3   if tree  $\lambda$  fulfils the preference degradation constraint then
4     if  $\sum_{\bar{m}(i,j) \in M_\lambda} Save(i, j) \geq Save_{min}$  then
5        $\bar{O} = \mathbb{F}(M_\lambda, \sum_{\bar{m}(i,j) \in M_\lambda} Save(i, j) - Save_{min})$ 
6        $O_\lambda = M_\lambda - \bar{O}$ 
7       if  $objective(O_\lambda) < objective(O)$  then
8          $O = O_\lambda$ 
9       end
10    end
11  end
12 end
13 return  $O$  as the global optimal solution for  $G(V, E)$ 

```

the local optimal migration vector set O_i to minimize the combinational objective with the cost saving constraint, which can be solved by the classic 0-1 knapsack problem. In particular, let $\mathbb{F}(ItemSet, TotalWeight)$ denote the standard 0-1 knapsack problem. The *ItemSet* is M_i in our problem and the *TotalWeight* is equal to $(\sum_{\bar{m}(i,j) \in M_\lambda} Save(i, j) - Save_{min})$,

where $Save_{min} = Cost_{initial} + c_0 + C^d - Cost_{max}$. We thus need to select a set of items \bar{M} (service migration vectors) in the *ItemSet* (M_i) with the total weight $\sum_{\bar{m}(i,j) \in \bar{M}} Save(i, j) \leq \sum_{\bar{m}(i,j) \in M_\lambda} Save(i, j) - Save_{min}$ so as to maximize the total value

$$\sum_{\forall \bar{m}(i,j) \in \bar{M}} (q \cdot Deg(i, j) - \frac{p \cdot Save(i, j)}{Lease_{max}})$$

From the optimal solution \bar{O} of \mathbb{F} , we can thus calculate the optimal solution O_i of M_i on the spanning tree i as $O_i = M_i - \bar{O}$. Then the global optimal solution can be found through enumerating all the spanning trees on the service migration graph $G(V, E)$. We summarize this optimal solution in Algorithm 1.

It is worth noting that finding the optimal solution for the standard 0-1 knapsack problem can become a time-consuming task as the crowdsourcers are distributed in a large scale, which can cause the optimal solution proposed in Algorithm 1 less suitable in practice, especially for an online system with highly dynamic crowdsourcer distribution and viewer demand. To this end, we further propose a simplified heuristic algorithm in Algorithm 2, which can work efficiently and still return the global optimal solution under certain situations. We then have the following theorem:

Theorem 2. *Algorithm 2 can return the global optimal solution when $Cost_{initial} + c_0 + C^d \leq Cost_{max}$ for each enumerated spanning tree.*

Note that, if we can prove that the local optimal solution in each spanning tree can be achieved by Algorithm 2 when $Cost_{initial} + c_0 + C^d \leq Cost_{max}$, we can then prove that Algorithm 2 can return the global optimal solution by Theorem 1. We can prove this using contradiction by assuming that there is a spanning tree λ with $Cost_{initial} + c_0 + C^d \leq Cost_{max}$ but has an optimal solution $O_\lambda \subseteq M_\lambda$, which is better than the solution O_λ found by Algorithm 2. As

Algorithm 2: Efficient online service migration()

```

1  $O = \emptyset$ 
2 for each enumerated spanning tree  $\lambda$  on  $G(V, E)$  do
3   if tree  $\lambda$  fulfils the preference degradation constraint then
4      $O_\lambda = \emptyset$ 
5      $Total_{save} = 0$ 
6     sort  $\bar{m}(i, j) \in M_\lambda$  with  $\frac{Deg(i, j)}{Save(i, j)}$  in increasing order
7     for  $\bar{m}(i, j) \in M_\lambda$  do
8       if  $(q \cdot Deg(i, j) < \frac{p}{Lease_{max}} \cdot Save(i, j))$  or
9         ( $Total_{save} < Save_{min}$ ) then
10        put  $\bar{m}(i, j)$  into  $O_\lambda$   $Total_{save} = Total_{save} + Save(i, j)$ 
11      end
12    end
13    if  $objective(O_\lambda) < objective(O)$  then
14       $O = O_\lambda$ 
15    end
16  end
17 return  $O$  as the online solution for graph  $G(V, E)$ 

```

$Save_{min} = Cost_{initial} + c_0 + C^d - Cost_{max} \leq 0$, we always have $Total_{save} \geq Save_{min}$. Thus, for all $\bar{m}(i, j) \in O_\lambda$, we have $q \cdot Deg(i, j) < \frac{p}{Lease_{max}} \cdot Save(i, j)$. The contradiction can thus be achieved by first identifying the difference between \bar{O}_λ and O_λ , and then showing that making changes to \bar{O}_λ according to O_λ can further improve \bar{O}_λ . Due to space limitation, here we omit the details of the proof, which can be found in our technical report presented.

VI. PERFORMANCE EVALUATION

We have implemented the crowdsourced live streaming system as a prototype based on PlanetLab, Amazon Cloud, Microsoft Azure Cloud, and the opensource VLC/VLM coder, and have conducted realworld experiments to understand its performance. We have also performed trace-driven simulations to further evaluate the system performance in large scale.

A. Prototype experimental results

In our prototype implementation, both the live crowdsourcers and end users are deployed in 398 Planetlab nodes, which are set up with VLC media player 0.8.7 Janus on each node. We deploy the federation of cloud service from Microsoft Azure Cloud and Amazon Cloud in our prototype platform. These two cloud service providers can offer totally 21 cloud sites distributed all over the world. In each cloud site, the General Purpose instances are provisioned with Medium (A2) from Microsoft Azure Cloud and m3.medium from Amazon Cloud. Each provisioned instance is set up with Ubuntu 14.04 LTS and installed with VLM to manage multiple live streaming channels. Further, we deploy the CloudFront CDN service in All Edge Locations for the globalized content delivery to the geo-distributed viewers. In order to evaluate the streaming quality, the live feeds are generated through videos uploaded from the distributed Planetlab nodes. We use a series of test videos with different resolutions and bitrates⁵. Each dedicated sourcer stores one of these videos as its own live feed. We deploy 18 cloud sites in different regions from Amazon Cloud and Microsoft Azure, 9 from America area, 3 from Europe area, and 6 from Asia Pacific, respectively. To explore the distribution of the 398 planetlab nodes, we measure the RTT latency between the nodes and the cloud sites, and use the cloud site with the minimal latency to approximate their

⁵<http://www.cs.sfu.ca/%7Ejcliu/infocom15/crowdsourcing/videos>

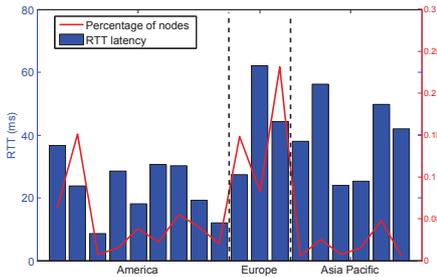


Figure 7: RTT latency

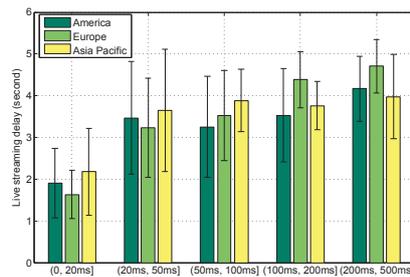


Figure 8: Different regions

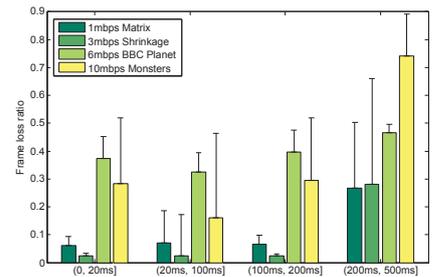


Figure 9: Videos with different bitrates

Table II: Three cloud leasing strategies for crowdsourced live streaming from 7 areas

	Van (10)	CA (19)	VA (20)	SA (5)	K. and J. (20)	CHN (16)	S. and A. (4)	Cost
TOP preferred first strategy	$m3 \times 3$ (Oregon)		$m3 \times 2$ (Virginia)	$m1 \times 1$ (Sao Paulo)	$m3 \times 2$ (Tokyo)	$m3 \times 1 + m1 \times 1$ (Singapore)	$m1 \times 1$ (Sydney)	\$5.584 per Hour
Centralized provisioning strategy			$m3 \times 5$ (Virginia)			$m3 \times 4$ (Singapore)		\$4.77 per Hour
Optimal migration	$m3 \times 3$ (Oregon)		$m3 \times 2 + m1 \times 1$ (Virginia)		$m3 \times 2$ (Tokyo)	$m3 \times 2$ (Singapore)		\$5.118 per Hour

locations. In Fig. 7, we present the nodes population and the average RTT latency from their top 1 preferred cloud sites. With the latency results, each sourcer can construct a preference list of the cloud sites. In order to measure the delay, we implement a live streaming of a timer video⁶ from the planetlab node to the cloud server. We also use `ffmpeg` to measure the frame loss ratio during the live streaming through recording the number of duplicated frames (i.e. because the current frame is not received by the playback deadline, the former frame is duplicated) and dropped frames (i.e. the frame is received but corrupted). These planetlab nodes are divided into groups according to the RTT latency in Fig. 7. We present the streaming delay in different areas in Fig. 8, and frame loss ratio with different videos in Fig. 9. Generally, we can see the streaming delay increase more than 80% if the latency is above 20ms. On the other hand, the frame loss ratio is stable when the latency is under 200ms.

We will further investigate the server provisioning cost and the video streaming quality of the cloud-based strategies through the implementation on the prototype platform. Besides our proposed optimal migration (OM) strategy, two other cloud-based strategies are implemented for comparison. The top preferred first (TOP) strategy deploys all the available cloud sites to allocate the service for sourcers in their most preferred cloud site. Meanwhile, in centralized provisioning (CP) strategy the cloud servers are allocated in the regions with the most sourcers. Here we select Virginia and Singapore as the central regions, and consider CP as the benchmark strategy. The implementation details of the cloud leasing strategy are presented in Tab. II. For example, $m3 \times 1 + m1 \times 1$ (Singapore) means one `m3.xlarge` instance and one `m1.large` instance are provisioned in Singapore region to serve 16 sourcers. We also calculate the server provisioning cost per hour according to the prices of Amazon EC2. CloudFront is deployed as CDN for the global distribution, and we record the average frame loss ratio from 20 distributed users. Generally the frame loss ratios can be reduced by about 10% for TOP and OM strategies. Especially, for the planetlab nodes in China, the improvement can reach almost 30% with the proposed

strategy. Comparing with TOP strategy, our proposed solution saves 8.34% cost, and improves 9.1% video quality on average.

B. Trace-driven simulation results

To further evaluate the performance of the proposed strategy in larger scale, we simulate the system with the real world trace data from Twitch.tv and the measurement results from the prototype system. The diverse prices of distributed cloud sites are referred to Amazon Cloud and Microsoft Azure Cloud. We consider a conventional centralized dedicated server (CDS) strategy as the benchmark, in which the single server is allocated in the central region to service the global requests. The price cost should cover the peak user demand, and we will take this cost as the budget constraint in our proposed OM strategy. We also set $p/q = 0.1$ and the preference value is inversely proportional to the RTT latency. Another two cloud based strategies are deployed for comparison. All these cloud-based strategies can scale their provisioning capacity adaptively to the user demand.

Fig. 11 shows the streaming delay reduction of the three cloud-based strategies comparing with the benchmark CDS strategy. Generally, TOP and OM strategies, which deploy the geo-distributed cloud service, can reduce almost 50% streaming delay of the benchmark strategy. The CP strategy can have an improvement only when most of viewers concentrate on several sourcers from the same region (e.g. 3:00AM-8:00AM in Asia and 13:00PM-16:00PM in Europe). Different from the streaming delay reduction, the frame loss reduction is more dynamic with time variations in Fig. 12. Before 8:00 AM, most of popular sourcers are from Europe and Asia, the CDS strategy would suffer from the long transmission, despite the total number of streams is not large, and there is still extra available bandwidth capacity for the rented server. After 9:00AM, sourcers from north America attract more viewer demand. Then dedicated server can provide an acceptable service with less frame loss ratio. In Fig. 13, we present the cost ratio between the three cloud-based strategies and the benchmark strategy. As the server instances are allocated in the distributed cloud sites with diverse prices, the Top1 strategy can lead to a higher cost when the peak demand comes. Because of the budget constraint, the provisioning

⁶<http://www.cs.sfu.ca/%7Ejcliu/infocom15/crowdsourcing/timer.mkv>

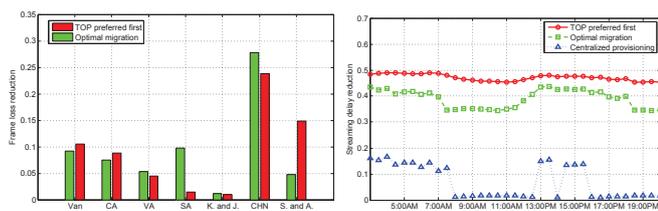


Figure 10: Implementation results

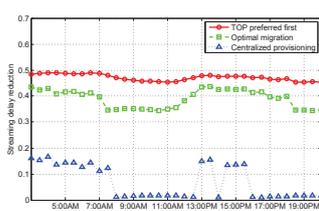


Figure 11: Reduction of streaming delay

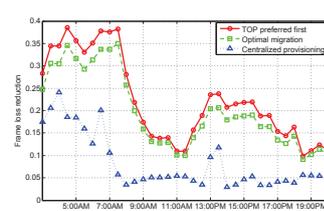


Figure 12: Reduction of frame loss

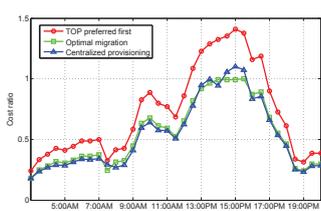


Figure 13: Reduction of provisioning cost

cost in our proposed strategy is limited under the cost of the benchmark. Yet, comparing with the TOP strategy, the gap of streaming delay and frame loss ratio can still be kept within 5%, and almost 30% of the provisioning cost is saved through the service migration during peak demand.

VII. CONCLUSION AND FUTURE WORK

In this paper, we explored the emerging crowdsourced live streaming systems, in which both the number and distribution of the crowdsourcers can be highly dynamic. It further motivated the design of cloud leasing strategy to optimize the cloud site allocation for geo-distributed live crowdsourcers. A prototype of crowdsourced live streaming platform was built with Amazon Cloud/Microsoft Azure and Planetlab nodes. The performance of the proposed strategy was evaluated through extensive experiments.

Our work is an initial study, and there are still many open issues to be further explored. We plan to continue enhancing our design by conducting more evaluations on our prototype with larger scale experiments. Our ongoing work includes tailoring our method for some specific crowdsourced live streaming applications, such as synchronizing multiple collaborative crowdsourced live videos for 3D immersive environment reconstruction or real-time interaction. We are also interested in extending our current deployment strategy to a more general scenario, in which the distributed server instances can cooperate with CDNs for a larger service coverage with a lower cost. In addition, we believe that the dynamic geo-distributed crowdsourcers are predictable, in which there are two major types of live sources, namely, scheduled sources and non-scheduled sources. The scheduled sources mean the crowdsourcers follow some social event during a certain time, such as a presidential election, or a football match, which is easy to predict. As to the non-scheduled sources, the crowdsourcers can start their live streaming arbitrarily. The time-varying live sources usually relate to the dynamic viewers demand, since the crowdsourcers are motivated to get more subscribers as a reward. They tend to broadcast in a fixed time every day, or choose a period when a peak number of viewers can be achieved. This behavior of crowdsourcers is evident in some modern crowdsourced live streaming platform, such as Twitch.tv. Our solution could be enhanced with crowdsourcer prediction through user behavior analysis from real-world measurement results.

ACKNOWLEDGMENT

This research is supported by Natural Sciences and Engineering Research Council of Canada (NSERC), a Start-up

Grant from the University of Mississippi, a Start-up Grant from the Jiangnan University, the National Natural Science Foundation of China (No. 61103223) and the Natural Science Foundation of Jiangsu Province (No. BK2011003).

REFERENCES

- [1] J. Liu, S. G. Rao, B. Li, and H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Proceedings of the IEEE*, 96(1):11-24, 2008.
- [2] B. Li, Z. Wang, J. Liu, and W. Zhu, "Two Decades of Internet Video Streaming: A Retrospective View," *ACM Transactions on Multimedia Computing, Communications and Applications*, Special Issue on 20th Anniversary of ACM SIGMM Multimedia, 9(1):1-20, October 2013.
- [3] H. Ma, D. Zhao, and P. Yuan, "Opportunities in Mobile Crowd Sensing," *IEEE Communications Magazine*, 52(8):29-35, August 2014.
- [4] C. Wu, Z. Yang, and Y. Liu, "Smartphones Based Crowdsourcing for Indoor Localization," *IEEE Transactions on Mobile Computing*, 14(2):444-457, February 2015.
- [5] Z. Ou, J. Dong, S. Dong, J. Wu, A. Ylä-Jääski, P. Hui, R. Wang, and A. Min, "Utilize Signal Traces from Others? A Crowdsourcing Perspective of Energy Saving in Cellular Data Communication," *IEEE Transactions on Mobile Computing*, 14(1):194-207, 2015.
- [6] J.-I. Biel, and D. Gatica-Perez, "Mining Crowdsourced First Impressions in Online Social Video," *IEEE Transactions on Multimedia*, 16(7):2062 - 2074, 2014.
- [7] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, M. Satyanarayanan, "Scalable Crowd-Sourcing of Video from Mobile Devices," *In Proc. ACM MobiSys*, 2013.
- [8] F. Wang, J. Liu, and M. Chen, "CALMS: Migration towards Cloud-Assisted Live Media Streaming," *In Proc. IEEE INFOCOM*, 2012.
- [9] Z. Wang, L. Sun, C. Wu, W. Zhu, and S. Yang, "Joint Online Transcoding and Geo-distributed Delivery for Dynamic Adaptive Streaming," *In Proc. IEEE INFOCOM 2014*.
- [10] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F.C.M. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," *In Proc. IEEE INFOCOM*, 2012.
- [11] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner and Z. Zhang, "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery," *In Proc. IEEE INFOCOM*, 2012.
- [12] C. Liu, I. Bouazizi, and M. Gabbouj, "Rate Adaptation for Adaptive HTTP Streaming," *In Proc. ACM Multimedia Systems*, 2011.
- [13] V. Aggarwal, V. Gopalakrishnan, R. Jana, K. K. Ramakrishnan, and V. A. Vaishampayan, "Optimizing Cloud Resources for Delivering IPTV Services Through Virtualization," *IEEE Transactions on Multimedia*, 15(4):789-801, 2013.
- [14] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand Forecast and Performance Prediction in Peer-Assisted On-Demand Streaming Systems," *In Proc. IEEE INFOCOM Mini-Conference*, 2011.
- [15] H. N. Gabow, and E. W. Myers, "Finding All Spanning Trees of Directed and Undirected Graphs," *SIAM Journal on Computing*, 7(3):280-287, 1978.
- [16] S. Kapoor, and H. Ramesh, "An Algorithm for Enumerating All Spanning Trees of a Directed Graph," *Springer Algorithmica*, 27(2):120-130, June 2000.