# Towards Hybrid Cloud-assisted Crowdsourced Live Streaming: Measurement and Analysis

Cong Zhang*, Jiangchuan Liu*, Haiyang Wang†
*Computing Science School, Simon Fraser University, BC, Canada
†Department of Computer Science, University of Minnesota Duluth, USA
{congz, jcliu}@cs.sfu.ca, haiyang@d.umn.edu

## ABSTRACT

Crowdsourced Live Streaming (CLS), most notably *Twitch.tv*, has seen explosive growth in its popularity in the past few years. In such systems, any user can lively broadcast video content of interest to others, e.g., from a game player to many online viewers. To fulfill the demands from both massive and heterogeneous broadcasters and viewers, expensive server clusters have been deployed to provide video ingesting and transcoding services. Despite the existence of highly popular channels, a significant portion of the channels is indeed unpopular. Yet as our measurement shows, these broadcasters are consuming considerable system resources; in particular, 25% (*resp.* 30%) of bandwidth (*resp.* computation) resources are used by the broadcasters who do not have any viewers at all. In this paper, we closely examine the challenge of handling unpopular live-broadcasting channels in CLS systems and present a comprehensive solution for service partitioning on hybrid cloud. The trace-driven evaluation shows that our hybrid cloud-assisted design can smartly assign ingesting and transcoding tasks to the elastic cloud virtual machines, providing flexible system deployment cost-effectively.

## CCS Concepts

**•Information systems → Multimedia streaming;**

## Keywords

Crowdsourced Live Streaming, Hybrid Cloud, Workload Migration, Twitch.tv

## 1. INTRODUCTION

Crowdsourced live streaming (CLS) has emerged as powerful, real-time means of video broadcasting over the Internet. Such commercial systems as *Twitch.tv*[1] (or Twitch for short) and *YouTube Gaming*[2] enable a new form of user-generated live streaming service, attracting an increasing number of viewers all over the globe. Using CLS-based eSports broadcasting as an example, it is known that the number of eSports gaming audiences has trebled to 89 million in the past three years [4]. The time spent watching eSports broadcasting has increased to 3.7 billion hours in 2014. To fulfill the elevating user demands, CLS service providers are aggressively expending their data center infrastructures. For example, Twitch has already deployed 25 service zones, hosting their dedicated streaming datacenters across five continents.

In this paper, we find that a significant fraction of these expensive data center resources is however consumed by the broadcasters who have very few or even no viewers. In particular, over 25% (*resp.* 30%) of the server bandwidth (*resp.* computation) resources are used to host broadcasters who do not have any viewers at all, not to mention other unpopular broadcasters who only have 1 or 2 viewers. To better examine these unpopular broadcasters in CLS system. We closely monitored 1.5 million broadcasters and 9 million streaming channels within a month. Different from the unpopular channel problem in traditional streaming systems [7], we find that the unpopular broadcasters are not only in greater numbers but also harder to manage in CLS systems. In particular, their online behavior is highly dynamic with short online duration but frequently arrival pattern. These highly dynamic broadcasters (video sources) are not yet considered in the optimization of existing streaming systems. Moreover, CLS highlights the event-related live streams with different broadcasters. One representative scenario is that multiple players (i.e., broadcasters) broadcast their game sessions from specific perspectives or languages. In each CLS event, streaming contents have an event-based correlation, but show broadcaster-based differences. All these make the workload optimization of CLS quite challenging.

Given the dynamic online pattern of these unpopular broadcasters, an intuitive and cost-efficient solution is to migrate their video ingesting and transcoding services to elastic cloud platforms. Our previous EC2-based measurement [14] also indicates that public cloud can provide comparable transcoding and communication latency if we can smartly assign cloud virtual machines (VMs) to the broadcasters. Based on this observation, we present the design of a hybrid cloud-assisted CLS framework (HyCLS) enhancing the utilization of existing dedicated

[1] www.twitch.tv, owned by Amazon.com in September, 2014.

[2] gaming.youtube.com



*NOSSDAV'16, May 13 2016, Klagenfurt, Austria*



DOI: http://dx.doi.org/10.1145/2910642.2910644

datacenters. We first propose the stable index (SI) to estimate the stability of broadcasters who already have historical activities. We further design effective algorithms to offload workload of new broadcasters. The evaluation shows that our proposed solutions can migrate up to 59.9% of the workload from dedicated datacenter to public cloud and cost-effectively reduce about 20% of lease cost in hybrid cloud-assisted design.

## 2. BACKGROUND

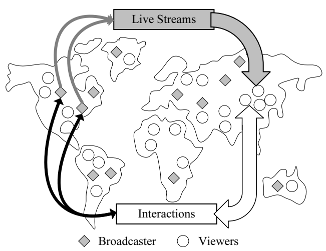


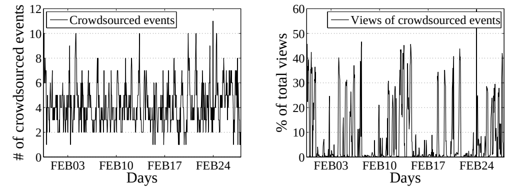

Figure 1: A generic diagram of CLS systems.

The idea of service crowdsourcing has attracted a substantial amount of attentions from both industry and academia [12, 8]. This service model refers to the process of getting contributions from a crowd of people (crowdsourcers) [8]. In multimedia-related crowdsourcing studies, various researchers proposed different frameworks to evaluate and improve users' Quality-of-Experience (QoE) for images and videos processing [2, 9].

Figure 1 briefly depicts a generic system diagram of crowdsourced live streaming platforms with streaming and interactive pipelines that jointly serve geo-distributed broadcasters and viewers. All sources are managed by amateur broadcasters (i.e., crowdsourcers) and driven by massive viewers/broadcasters in real-time using live messages (e.g., TwitchPlaysPokemon). Use Twitch as a case study, our previous work has illustrated the basic architecture of CLS platform [15]. Several recent studies already focused on crowdsourced live streaming services [6, 1]. Different from the existing studies, our work examines the online behavior of the crowdsourcers and explores the effective utilization of resources. Our work differs from these recent studies in the following aspects: first, we target on crowdsourced live streaming systems which represent lots of unique features. For example, geo-distributed broadcasters determine the service quality from the "first-mile" of live streaming distribution. The scalability of streams in this process basically determines the viewers' QoE. Second, any improvement and optimization must carefully design the strategy to generate low-latency live streaming. Therefore, we propose an optimal solution that cost-effectively schedules the broadcasters' workloads to public clouds in the crowdsourced live content generation.

## 3. MEASUREMENT OF CROWDSOURCED LIVE STREAMING: TWITCH AS A CASE STUDY

In our measurements, we try to answer the following fundamental questions: how many unpopular broadcasters exist in real crowdsourced live streaming systems? And, what is the underlying impacts of them? As such, we deeply measure the workloads and corresponding resource consumption based on the crawled dataset from Twitch.




(a) Daily patterns (b) Effects of crowdsourced live events

Figure 2: Characteristics of crowdsourced live events.

Our investigation is based on crawled data, which are continually collected from Twitch every five minutes in a one-month period (Feb.1st-28th, 2015). Through the Twitch APIs [3], our multi-thread crawler obtained information from each broadcaster and whole system dashboard. We retrieved broadcaster datasets and stream datasets through polishing aforementioned data. A brief explanation is as follows:

- in broadcaster datasets: each trace collects the total number of views and other statistics such as playback bitrate, resolution, and partner status, for a total of 1.5 million broadcasters (2% outliers are eliminated).

- in stream datasets: each trace records the number of viewers every five minutes and other properties including start time, duration, game name, etc., for a total of 9 million streams (0.3% outliers are removed).

*Effects of Crowdsourced Live Events*: CLS highlights the event-related live streams with different broadcasters. One representative scenario is that multiple players (i.e., broadcasters) broadcast their game sessions from specific perspectives or languages. In each CLS event, streaming contents have an event-based correlation, but show broadcaster-based differences. To illustrate this distinct feature, we explore the crowdsourced live events based on the broadcaster's channel name and game type in each trace of stream datasets. Figure 2a plots the number of crowdsourced events during one month [4]. We observe that crowdsourced live events exist in all data traces. Although the highest number is twelve, these live events attract up to 52% of total views in our measurement, as shown in Figure 2b. This distinct feature will be considered in the problem formulation (Section 5).

*Popularity of Crowdsourced Streaming Channels*: We then focus on the distribution of broadcaster's popularity, which has played a key role in previous studies for multimedia systems, and is also critical to answer our first question. We plot the highest number of concurrent views against the rank of the broadcasters (in terms of the popularity) in log-log scale in Figure 3. From this figure, we observe that the popularity of those broadcasters exhibits perfect Zipf's pattern[5]. We further find that there exists such a high skewness that the top-3% popular broadcasters account for about 80% of the peak requests. Another interesting result shows that 90% of broadcasters only attract less than 8 viewers (labeled on the small figure in Figure 3 ) even

[3]http://dev.twitch.tv/

[4]Due to space limitation, only four date labels are displayed.

[5]We use the coefficient of determination, denoted $R^2$, to illustrate how well our measured data fit the Zipf's law.

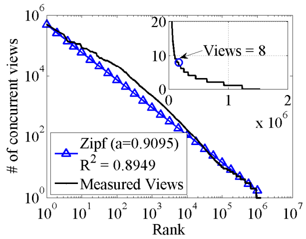
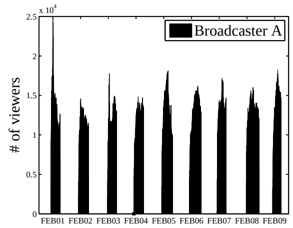
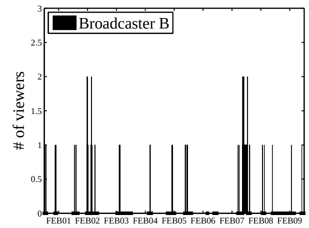



Figure 3: Broadcasters rank ordered by popularity.




(a) Popular broadcaster A




(b) Unpopular broadcaster B

Figure 4: Two samples of broadcasters.

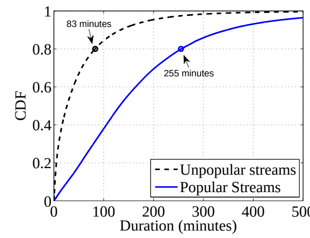
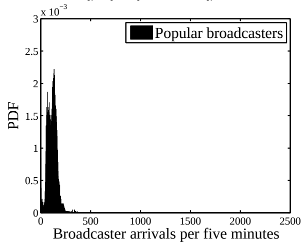
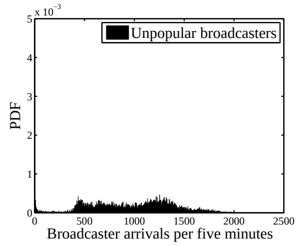



Figure 5: The distribution of duration.

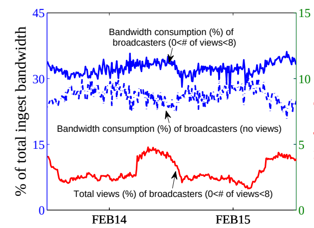



(a) Popular broadcasters

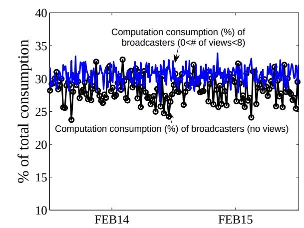



(b) Unpopular broadcasters

Figure 6: Broadcaster arrivals per five minutes.




(a) Bandwidth consumption




(b) Computation consumption

Figure 7: The effectiveness of resource consumption.

at their peak time in our one-month broadcaster datasets. Based on these findings, if the highest number of concurrent views number of a broadcaster is less than 8, we assume that s/he is not a popular broadcaster. How long are these broadcasters' live streams? Is there any difference between popular and unpopular broadcasters in terms of live duration? We next investigate the characteristics of broadcasters' streams based on stream datasets.

*Dynamics of Crowdsourced Live Broadcasters*: If the concurrent number of views in one stream is less than 8, the popularity of this stream is quite low. To further investigate the characteristics of them, we compare the distribution of their duration with popular streams in Figure 5. This figure shows that the durations of 80% of unpopular streams are less than 83 minutes, implying the workloads of them are highly dynamic. Because the number of unpopular streams is quite large (about 8.13 million), massive unpopular streams could occupy the datacenter resources frequently and dynamically. We also calculate the total duration of all unpopular streams in one month to be nearly 830 years, while the total duration of popular streams is only 310 years. Therefore, a huge amount of resources cannot be utilized effectively. To illustrate the different characteristics of two types of broadcasters, we plot their activities during ten days in Figure 4a and 4b. Figure 4a shows that broadcaster A has a regular live schedule with the stable live duration, attracting a large number of viewers. While the broadcaster B in Figure 4b not only has irregular schedules, but also consumes dedicated resources during the dynamic live duration. We also plot The Probability Distribution Function (PDF) of the broadcaster arrival rate every five minutes in Figure 6. This figure shows that the arrivals of popular broadcasters are clearly lower than 300, while the unpopular broadcaster's arrival rate has a considerable range from 400 to 1800. Due to the frequent arrivals and huge resource consumptions, it is necessary to enhance current CLS systems with optimizing the dynamic workloads of these unpopular broadcasters.

*Challenge of Hosting Unpopular Broadcasters*: To evaluate the underlying challenges of these unpopular broadcasters, we use the playback bitrate and resolution of each live stream to estimate the consumptions of bandwidth and computation resources based on the measurement works in [1]. Figure 7 shows the proportion of bandwidth/computation consumption of two types of broadcasters when they stream live content to ingest servers on Feb 14th/15th, 2015. The broadcasters who do not have any viewers consume about 25% (*resp.* 28%) of bandwidth (*resp.* computation) resources. At the meantime, about 33% (*resp.* 31%) of bandwidth (*resp.* computation) resources are consumed by the broadcasters who only have less than 8 concurrent viewers. This figure also shows that these broadcasters only attract less than 5% concurrent viewers, which means that CLS service providers have to carry out a large number of ingest servers to allocate these unpopular broadcasters dedicated bandwidth/computation resources continually.

# 4. HYCLS ARCHITECTURE

Based on our Twitch measurement, we have demonstrated that the characteristics of broadcasters/streams and illustrated that the dedicated resources are not to be consumed effectively. Our previous EC2-based measurements also illustrate public cloud can support crowdsourced live streaming effectively [14]. As such, we present the architecture of our hybrid cloud-assisted crowdsourced live streaming system HyCLS in this section.

In CLS system, broadcasters constantly utilize the streaming pipeline, considering the latency-intensive features, any service interruption will generate a series of degradation of viewer's QoE. Therefore, the main challenge of our design is to optimize the broadcast latency. Besides, crowdsourced live events, wherein several broadcasters simultaneously start live-broadcast, have a more stringent requirement on the disparity of broadcast latencies between various broadcasters. To address these problems, our design focuses on stream pipeline and optimizes the following three steps: (1) *Initial Offloading*, for the broadcasters who have historical information about live streams, the system makes an offloading decision between public cloud and dedicated datacenter when they start to connect ingest servers. (2) *Ingesting Redirection*, according to the broadcasters' performs, the system assigns one alternative ingest area and redirect her/his source streaming; (3) *Transcoding Schedule*, each offloading also consider the transcoding capacities of various service areas in HTTP Live Streaming scenario. In fact, step 2 and 3 have to be designed together, the reason is that once the workload of a broadcaster is offloaded to a certain ingest area, transcoding workload has to be processed in the same area.

We introduce our design as shown in Figure 8, which represents the main components in the HyCLS system. For example, according to the historical information, Initial Offloading strategy first assigns broadcaster A and B to the second dedicated datacenter and the second public cloud area, respectively. After several time slots, Ingesting Redirection and Transcoding schedule modules migrate them to a proper service area based on our strategies. Next, we propose the design of offloading decision and optimize the Ingest Redirection and Transcoding Schedule together in Section 5. In our hybrid framework, the first challenge is how to select proper ingest server to broadcasters at the beginning of live-broadcast. Therefore, we have to estimate the stability of each broadcaster based on her/his historical activities. For one broadcaster $b$ who has activities in recent $n$ days ($n \geq 2$), we first divide the $i$th day to $m$ equal time slots, each time slot $j$ has a value $d_{i,j}$ that indicates whether $b$ have a live streaming in current time slot. In fact, $SI^{(b)}$ reflects the similarity of $b$'s resource consumption in recent $n$ days.

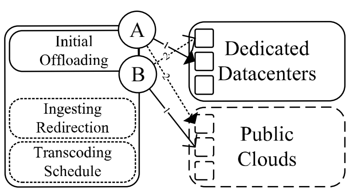



Figure 8: The design of HyCLS.

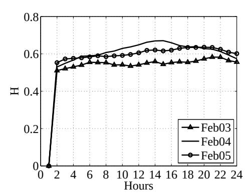



Figure 9: Impacts of $H$.

$$SI^{(b)} = \begin{cases} \frac{1}{n}\sum_{i=2}^{n} \frac{\sum_{j=1}^{m} d_{i,j}^{(b)} \cdot d_{i-1,j}^{(b)}}{\sum_{j=1}^{m} d_{i-1,j}^{(b)}} & \text{if } \sum_{j=1}^{m} d_{i-1,j}^{(b)} \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Given the stable index $SI^{(b)}$ of a broadcaster $b$, a straightforward way to give the offloading decision is to set a threshold $H$: if $SI^{(b)} \geq H$, $b$ will be assigned to the ingest servers in dedicated datacenters, otherwise, public cloud ingests the live streaming of $b$. Using a firm threshold, however, suffers from the following drawback: during lower workload stage, leasing public clouds to specifically ingest unpopular workloads is not a cost-effective strategy, because the existing resources in dedicated datacenters can completely process every broadcaster's live stream. We solve this problem by detecting the existing broadcasters' $SI$ in one dedicated datacenter and update the value of $H$ to the average $SI$ of all broadcasters per unit time. Followed by the growth of broadcasters, more and more regular broadcasters will be ingested into dedicated datacenters, and other dynamic broadcasters are offloaded to public clouds at the beginning of live-broadcast. We further evaluate the effectiveness of the stable index.

# 5. PROBLEM FORMULATION AND SOLUTION

Due to the dynamic and unpredictable features of broadcasters in CLS system, we design the ingesting and transcoding strategies based on the current workloads status. The workloads can be migrated among different service areas in real-time. Consider the critical features including crowdsourced live events and latency synchronization, we take broadcast latency as our objective and propose a formal description of our optimization problem in the current crowdsourced scenario.

## 5.1 Problem Formulation

To make the problem easy to discuss, we quantize time into discrete time slots, which may be a few minutes to several hours (e.g., five minutes in our experiment). We use $B^{(t)}$ to denote the set of broadcasters and $E^{(t)}$ to denote the set of crowdsourced live events in time slot $t$. ($\forall i = 1, 2, \cdots, m, \forall j = 1, 2, \cdots, m,\ e_i \in E^{(t)},\ |e_i| \geq 1$, $e_i \cap_{i \neq j} e_j = \varnothing$, and $\cup e_i = B^{(t)}$). We define $R$ as the set of ingest areas where a broadcaster can be connected to upload live content and define set $W_r^{(t)}$ as the bandwidth demand of ingesting area $r$. We assume that the instance in public cloud are homogeneous and let $\mathcal{W}$ denote the bandwidth capacity of each instance. Therefore, we do not consider the optimization inside each service area, a large number of works have focused on this area and acquire a better optimization [10].

We consider the impacts of ingesting stage and transcoding workload with various versions on different ingest areas. As such, $b$'s broadcast latency $L_{(b,r,v)}^{(t)}$ is calculated as:

$$L_{(b,r,v)}^{(t)} = l_{(b,r)}^{(t)} + l_{(q_b,q_v)}^{(t)} + l_{(r,v)} \quad (2)$$

where $v$ is the transcoding version, $v \in V$, $l_{(b,r)}^{(t)}$ is the link latency between $b$ and $r$, $l_r^{(t)}$ is the ingest latency that is determined by the instance type in $r$, $q_b$ and $q_v$ are bitrates of source (i.e., broadcaster $b$) and transcoding version $v$ ($v \in V, V = 0, \mathbb{Z}^+$), respectively. $l_{(q_b,q_v)}^{(t)}$ is the transcoding latency, which can be measured in advance. $l_{(r,v)}$ is the latency between ingest area $r$ to a class of viewers $v$, which will be defined in next section.

We now define utility function $U^{(t)}(b, r, v)$ as:

$$U^{(t)}(b, r) = \sum_{v \in V} G^{(t)}(b, r, v) \cdot N_{(b,v)}^{(t)} \quad (3)$$

where $N_{(b,v)}^{(t)}$ is the number of viewers who watch $b$'s $v$ version streaming in this time slot. This value is initially determined by $b$'s historical distribution of different versions. $G^{(t)}(b, r, v)$ means the gain when $b$ select $r$ as the ingest and transcode area and is calculated as follows:

$$\begin{aligned} G^{(t)}(b, r, v) &= \alpha + ln(1 - \beta L_{(b,r,v)}^{(t)}) \\ &= \alpha + ln(1 - \beta(l_{(b,r)}^{(t)} + l_{(q_b,q_v)}^{(t)} + l_{(r,v)}^{(t)})) \end{aligned} \quad (4)$$

where $l_{(q_b,q_v)}^{(t)}$ denotes the transcoding latency. If $q_b \leq q_v$, $l_{(q_b,q_v)}^{(t)} = l_{(q_b,q_b)}^{(t)}$, which depends on the current computing capacity of area $r$ and is monotonously increasing on both $q_b$ and $q_v$ [13].

Based on previous definitions, our objective is :

$$\underset{e \in E^{(t)}}{\text{Maximize}}\ F(A^{(t)}) = \min_{\substack{b \in e \\ r \in R}} \{U^{(t)}(b, r)\} \quad (5)$$

*subject to:* Resource Availability Constraints:

$$\forall r \in R, W_r^{(t)} \le \mathbb{W}_r \tag{6}$$

$$\forall r \in R, C_r^{(t)} \le \mathbb{C}_r \tag{7}$$

Budget Constraints:

$$\textstyle\sum_{r\in R} \frac{W_r^{(t)}}{\mathcal{W}} \cdot Cost_w(r) \cdot I(r) \le K_w \tag{8}$$

$$\textstyle\sum_{r\in R} \frac{C_r^{(t)}}{\mathcal{C}} \cdot Cost_c(r) \cdot I(r) \le K_c \tag{9}$$

where $\mathbb{W}_r$ is the bandwidth capacity of ingest area $r$. $Cost_w(r)$ is the bandwidth price in area $r_i$. The bandwidth constraint (6) asks that at any given time, the bandwidth demands have to be satisfied. $C_r^{(t)}$ is the computing demand of area $r$, $\mathcal{C}$ denotes the amount of per unit computing resource. $Cost_c(r)$ is the price of instance in $r$ in terms of computing capacity. The computing constraint (7) guarantees that at any given time $t$, the computation resource consumption of each transcoding task can be satisfied. The budget constraints (8) and (9) guarantees that the bandwidth/computation cost is lower than the budget $K_c/K_w$, which we assume can at least serve all offloading workloads.

## 5.2 Solution

Current formulated objective (5) has four constraints (6) (7) (8), and (9). It is hard to solve this optimization problem efficiently in a short time. Fortunately, the bandwidth cost and computation cost are not independent due to the pricing criteria of the instance on a public cloud. Previous studies on EC2 instances already reveal that the bandwidth capacity is more than 700Mbps on m3.large instance [5]. As such, based on our measurement results in [14], generating low-latency live streams will consume a vast of computation resources. If we relax constraints (6) and (8), another constraints still work for optimizing objective (5). Assuming that the capacities of the different public cloud service area are given, our assignment problem can therefore be transformed into a 0-1 Multiple Knapsack problem, which is known to be NP-hard [3]. Although the optimal solution can be reached through meticulously searching all possible assignments, this is unpractical in real CLS system. Inspired by our previous work [11], we thus propose a heuristic algorithm, which consists of scaling decrease and resource assignment, as shown in Algorithm 1. In the scaling decrease step (line 1 to 10), we eliminate the redundant assignment solutions based on the optimization target. The line 2 search the maximum value from the set of a minimum utility of each assignment $(b, r, v)$ in crowdsourced live events. The line 4 to 10 then remove all useless assignment and guarantee the rest of assignment can be effectively used in the next algorithm. We next use resource assignment (line 11 to 22) to implement an effective solution. The main idea is to utilize the utility in live streaming broadcast latency by a unit of computation resources.

**Algorithm 1** WorkloadAssignment()

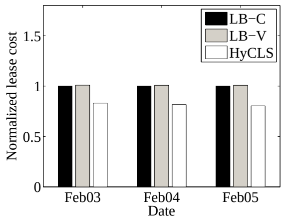
1: **for** each crowdsourced live event $e \in E$ **do**
2: $\quad U_e^{(t)} \leftarrow \max_{r\in R}\{\min_{b\in e}\{U^{(t)}(b,r)\}\}$ ;
3: **end for**
4: **for** each crowdsourced live event $e \in E$ **do**
5: $\quad$ **for** each assignment $(b,r) \in A^{(t)}(b,r)$ **do**
6: $\quad\quad$ **if** $U^{(t)}(b,r) < U_e^{(t)}$ and $IsPath(b,r) == true$ **then**
7: $\quad\quad\quad A_*^{(t)} \leftarrow A^{(t)} - (b,r)$; //Remove this assignment path
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**
11: Sort $(b,r)$ by descendant order of $U^{(t)}(b,r)/Cost_c(r)$;
12: **for** each assignment $(b,r) \in A_*^{(t)}$ **do**
13: $\quad r_{sorted} \leftarrow$ Sorted available area $r$ of $b$ by descendant order of $U^{(t)}(b,r)$;
14: $\quad$ **for** each $r \in r_{sorted}$ **do**
15: $\quad\quad$ **if** $C_r^{(t)} - c(b) \ge 0$ **then**
16: $\quad\quad\quad A_*^{(t)} \leftarrow A_*^{(t)} - (b,\cdot)$; //Remove all assignment of $b$
17: $\quad\quad\quad C_r^{(t)} \leftarrow C_r^{(t)} + c(b)$; //$c(b)$ is the computation consumption of transcoding workloads $b$
18: $\quad\quad\quad A_*^{(t)} \leftarrow A_*^{(t)} + (b,r)$;
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: **return** $A_*^{(t)}$

# 6. PERFORMANCE EVALUATION

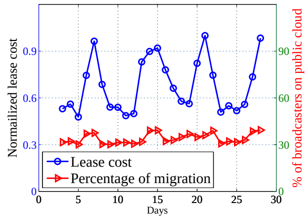
We now evaluate the performance of our solution via trace-based simulation, which captures the broadcasters' streaming pattern, including resolution, partner status, and concurrent viewers, etc., in Twitch dataset. We consider two broadcaster's types: partner, whose live streaming can be adaptively transcoded, and common broadcasters, whose viewers only can watch the source quality HTTP Live Streaming. At the meantime, we make a few simplifications in the simulation based on realistic settings: first, to simplify the complexity of algorithm, we consider that the EC2 instances are homogeneous (m3.large) and latency $l_{(r,v)}$ is fixed for a certain quality level of HTTP Live Streaming; second, due to the confidential nature of official implementation, we cannot acquire the details of dedicated datacenter, we show the comparisons of extra outlay when workloads are offloaded into public cloud. The price data of instances come from Amazon. The following settings are the default parameters in the simulation: to normalized the impacts of broadcast latency, we set $\alpha = 1$ and $\beta = 0.011$, which makes the gain $G^{(t)}(\cdot) \in [0, 1]$, if the broadcast latency $L_{(\cdot)}^{(t)} \in [0, 57]$, which embraces a general broadcast latency interval $[10, 40]$ in Twitch [15]. The algorithms are launched per five minutes, which also is the time slot of crawling data.

We first conduct simulations to study the impacts of stable index $SI$ and threshold $H$. We set $n = 2$ to calculate the broadcaster's stable index in advance and set the initial threshold $H = 0$. To illustrate the efficiency of this threshold, we use it to classify the new broadcasters without any other strategies. We assume that the offloading starts when the bandwidth consumption is up to 60% of the dedicated datacenter. Figure 9 illustrates the evolution of $H$ and its impacts for the public cloud during three days (Feb 3rd-5th, 2015). From this figure, we observe that the value of $H$ increase dramatically at the beginning of that day, and then it stables between 0.5 and 0.7. At the peak traffic time (from 9:00AM to 13:00PM), a vast of broadcasters arrive streaming system; therefore, the value of $H$ occurs a small decrease. However, the limitation of $H$ induces that public cloud only hosts a few number (maximum 6.5%) of broadcasters. Thus, threshold $H$ plays a beneficial role in the offloading process, but it still cannot reduce the impacts of dynamic broadcasters sufficiently.

With the previous parameter setting of $H$, we then conduct simulations to investigate how HyCLS performs



(a) Lease cost of three approaches



(b) Migration performance (Feb.03-28)

Figure 10: The performance of proposed solutions.

with the real data traces. Figure 10a compares the lease cost of three workload provisioning approaches: views-based (LB-V), computation-based (LB-C), and HyCLS-based approaches in three days. LB-V only considers the current number of views in different live streams, while LB-C migrates workload based on the consumption of computation resources.. For ease of comparison, the lease cost in each day is normalized by the corresponding cost of the LB-C approach. Our HyCLS-based approach has the lowest cost, decreasing 16.9%-19.5% of LB-C approach and 17.8%-20.4% of LB-V approach. Another observation is that the lease cost of Feb03 is higher than those of another two days in all approaches. This is because there has the highest number of broadcasters on Feb03. We also plot the normalized lease expenses and the average percentage of migration during our whole datasets in Figure 10b, we can observe that the daily decreasing cost performs the weekly pattern and provide elastic workload provisioning cost-effectively. Moreover, more than 30% of broadcasters are migrated to the public cloud in every day. We further explore the highest percentage of migration and find that up to 59.9% of broadcasters will be assigned to public cloud on several time slots. Our simulation results show that compared with spending massive outlays to manage and upgrade dedicated datacenters, leasing flexible public cloud is a cost-effective solution in terms of decreasing the influences of dynamic unpopular broadcasters and providing highly available live streaming services.

# 7. CONCLUSION AND FUTURE WORK

This paper presented HyCLS, a generic framework that facilitates migrating crowdsourced live streaming between dedicated datacenters and public clouds. We strived to offer the comprehensive understandings on the practical crowdsourced live streaming system and explore the potential enhancement. We first measured Twitch-based datasets to investigate the challenges therein. We observed that unpopular broadcasters consume the massive valuable dedicated resources continually. We then proposed a hybrid design for the initial offloading, as well as dynamic ingesting redirection and transcoding assignment that accommodates unpredictable workloads and realizes the adaptive offloading in demand. Extensive simulations driven by traces from Twitch and settings from Amazon EC2 demonstrated the cost-effectiveness and superior migration of HyCLS. We are currently examining the performance of our hybrid cloud design on the PlanetLab. We are also interesting in developing a better initial offloading strategy, as well as incorporating other factors into stable index and threshold evolution, e.g., broadcaster's social characteristics.

# Acknowledgment

This research is supported by an NSERC Discovery Grant, NSERC RTI Grant and NSERC Strategic Grant. C. Zhang's work is supported in part by a China Scholarship Council (CSC) Scholarship Program. H. Wang's work was supported by Chancellor's Small Grant and Grant-in-aid programs from the University of Minnesota.

# 8. REFERENCES

[1] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *ACM MMSys, 2015*.

[2] A. Carlier, G. Ravindra, V. Charvillat, and W. T. Ooi. Combining content-based analysis and crowdsourcing to improve user interaction with zoomable video. In *ACM MM, 2011*.

[3] A. Drexl. A simulated annealing approach to the multiconstraint zero-one knapsack problem. *Computing*, 40(1):1–8, 1988.

[4] A. Gaudiosi. esports: Espn's analysis on esport trends. http://goo.gl/p2aSXZ, July 2015.

[5] M. Hajjat, R. Liu, Y. Chang, T. E. Ng, and S. Rao. Application-specific configuration selection in the cloud: impact of provider policy and potential of systematic testing. In *IEEE INFOCOM, 2015*.

[6] 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 *ACM WWW, 2012*.

[7] Z. Liu, C. Wu, B. Li, and S. Zhao. Why are peers less stable in unpopular p2p streaming channels? In *NETWORKING 2009*, volume 5550 of *Lecture Notes in Computer Science*, pages 274–286. 2009.

[8] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas: Understanding captcha-solving services in an economic context. In *USENIX Security, 2010*.

[9] Y. S. Rawat and M. S. Kankanhalli. Context based photography learning using crowdsourced images and social media. In *ACM MM, 2014*.

[10] F. Wang, J. Liu, and M. Chen. Calms: Cloud-assisted live media streaming for globalized demands with time/region diversities. In *IEEE INFOCOM, 2012*.

[11] H. Wang, R. Shea, X. Ma, F. Wang, and J. Liu. On design and performance of cloud-based distributed interactive applications. In *IEEE ICNP, 2014*.

[12] Wikimapia. Wikimapia - let's describe the whole world! http://wikimapia.org/, July 2015.

[13] Y. Wu, C. Wu, B. Li, and F. C. Lau. vskyconf: Cloud-assisted multi-party mobile video conferencing. In *ACM SIGCOMM Workshop on MCC, 2013*.

[14] C. Zhang. Performance analysis of Amazon EC2 in CLS scenario. https://goo.gl/c7CTOa, Feb 2016.

[15] C. Zhang and J. Liu. On crowdsourced interactive live streaming: A twitch.tv-based measurement study. In *ACM NOSSDAV, 2015*.