Live Broadcast With Community Interactions: Bottlenecks and Optimizations

Xiaoqiang Ma, *Member, IEEE*, Cong Zhang, *Student Member, IEEE*, Jiangchuan Liu, *Fellow, IEEE*, Ryan Shea, *Member, IEEE*, and Di Fu, *Student Member, IEEE*

***Abstract*—Recent years have witnessed the rapid growth of new live broadcast services, represented by Twitch.tv and YouTube live events, where videos are crowdsourced from amateur users (e.g., game players), rather than from commercial and professional TV broadcaster or content providers. The viewers also actively contribute to the content through embedded open-chat channels. Such community interactions among viewers, or even between broadcasters and viewers, make content generation highly diversified and engaging, particularly for the young generation. In this context, cross-viewer synchronization is highly desirable; otherwise the viewers with shorter broadcast latency may act as spoilers, significantly affecting the user experience of other viewers. In this paper, we show that the end-to-end delay has a dramatically amplified impact on the broadcast latency for individual viewers. We suggest smart rate adaptation to achieve cross-viewer synchronization, and develop distributed algorithms based on dual decomposition. We further extend our solution to the cloud environment, and present the concept of ShadowCast, which moves broadcasters to the cloud to provide high-quality streams beyond broadcasters' network bandwidth constraint. Its practicability and effectiveness is demonstrated by our implementation and test bed experiments.**

***Index Terms*—Community interactions, live broadcast.**

I. Introduction

In RECENT years, live broadcast with community interactions has become very popular, represented by Twitch.tv[1] (referred to as Twitch in the rest of this paper), YouTube live events, Douyu TV,[2] Panda TV,[3] and Inke.[4] A typical live broadcast channel features a combination of a broadcaster's high-fidelity game play graphics, her/his real-life activities captured by a web camera, and an open chat channel shared by viewers in the same channel (*fellow viewers*). The broadcaster can also communicate with the viewers in real time. Such community interactions significantly foster user participation and largely contribute to the success of the new live broadcast services [1]. Launched in the year 2011, Twitch has already attracted over 100 million unique viewers per month and over 1.7 million unique broadcasters per month by the year 2015.[5]

Compared with traditional video streaming [2], most video sources in such new live broadcast platforms are generated by amateur users, rather than commercial and professional content providers, which remarkably stimulates content diversity. Apart from video broadcasters, viewers can also contribute to the channel content: they can either comment on the broadcaster's performance, or chat/argue with fellow viewers. All these real-time community interactions displayed in the open channel in turn attract a significant portion of viewers' attention.

Given the importance of community interaction, cross-viewer synchronization is highly desirable. Otherwise the viewers with shorter broadcast latency may act as spoilers, while the viewers with longer broadcast latency may post comments on the content already watched by others a while ago, both significantly affecting user experience.[6] Unfortunately, synchronization in this context has to deal with not only the scale of the viewer base, but also the amplified impact of the end-to-end network delay on the broadcast latency for individual viewers. Through a series of controllable experiments, we find that even a slight increase in the end-to-end delay (say a few hundred of milliseconds) can elongate the broadcast latency to over ten seconds, which is intolerable for real-time community interaction. Considering the heterogeneous network conditions of individual viewers, the end-to-end delay of individual viewers would inevitably differ to a high degree, leading to highly unsynchronized playback.

In this paper, we suggest smart rate adaptation to semi-synchronize playback among fellow viewers. The rate-adaptation schemes for tradition live streaming mainly focus on selecting the most suitable video rates that balances streaming quality and playback fluency [4]–[7]. In the context of community interaction, we consider rate adaptation as a network utility maximization (NUM) problem with constraints of the streaming

Manuscript received March 28, 2016; revised September 9, 2016; accepted December 15, 2016. Date of publication December 29, 2016; date of current version May 13, 2017. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2016YXMS293, in part by a Canada NSERC Discovery Grant, and in part by a Canada NSERC Strategic Project Grant. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Honggang Wang. *(Corresponding author: Jiangchuan Liu.)*

X. Ma is with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: maxiaoqiang@hust.edu.cn)

C. Zhang, J. Liu, R. Shea, and D. Fu are with the School of Computing Science, Simon Fraser University, BC V5A 1S6, Canada (e-mail: congz@sfu.ca; jcliu@sfu.ca; rws1@sfu.ca; dif@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.2016.2646182

[1][Online]. Available: https://www.twitch.tv/
[2][Online]. Available: https://www.douyu.com/
[3][Online]. Available: http://www.panda.tv/
[4][Online]. Available: https://www.inke.cn/
[5][Online]. Available: https://www.twitch.tv/p/about
[6]Broadcast latency refers to the time lag of a live event when viewers watch the live streaming from the source [3].

capacity and the bound of latency difference. We develop a distributed algorithm based on dual decomposition, which allows viewers to select appropriate playback bitrates without knowing others' information. Given that many of the amateur broadcasters rely on home networks with relatively low and unstable bandwidth (e.g., 512 Kbps to 2.5 Mbps uploading bandwidth), uploading high quality video streams in real-time can be difficult or even impossible.[7] We further extend our solution to the cloud environment and present the concept of ShadowCast, which moves broadcasters to the cloud to provide high quality streams beyond broadcasters' network bandwidth constraint. Its practicability and effectiveness is demonstrated by our prototype implementation and testbed experiments.

The rest of this paper is organized as follows. We investigate the impact of end-to-end delay on broadcast latency in Section II. We formulate the cross-viewer synchronization problem, and develop a distributed algorithm in Section III. We discuss the deployment and optimization for cloud environment in Section IV. We evaluate our proposed solutions in Section V. We provide further discussion in Section VI and conclude this paper in Section VII.

II. Community Interaction: Delay Can Kill

It has been reported that community interaction has played an important role in Twitch-like live broadcast services. For example, 61% Twitch users chat with community,[8] and the chat lines of all broadcast channels is found to constantly exceed 400 per second.[9]

To investigate the impact of network condition on broadcast latency, we conducted experiments with different end-to-end delays. We set up two computers: one serves as the broadcaster and uses the Open Broadcaster Software (OBS),[10] one of the most widely used broadcast applications, to encode a source video at 1500 Kbps bitrate; the other uses VirtualBox[11] to host two identical virtual machines (VMs). We created a Twitch channel, and the two VMs watched the video stream as viewers. With this setting, the two viewers have almost identical network conditions. We used the *ipfw* tool [8] to change VMs' propagation delays and bandwidth limits. First, we set the bandwidth limit to 2000 kbps for both VMs. We then added the propagation delay of one VM (VM A) from 100 to 400 ms, while keeping the other VM (VM B) unchanged. The broadcast latency difference between the two viewers under different network conditions is shown in Fig. 1. We can see that, with added propagation delay, the broadcast latency difference becomes significantly longer. Especially, when the added delay is 400 ms, the broadcast latency difference has a sheer increase; a broadcast latency difference of almost 20 seconds is almost intolerable for real-time interaction between the viewers. The reason is that, when the

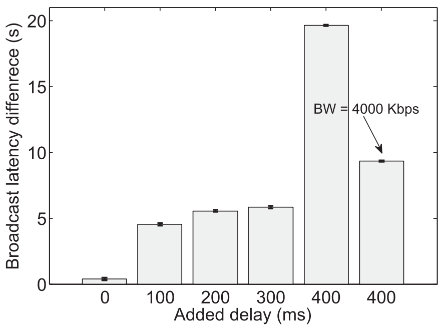


Fig. 1. Broadcast delay difference under different network conditions.

propagation delay becomes longer, it takes more time to fill the buffer. In general, most streaming players require a number of chunks, say 2 to 3, to be received before starting playback.

On the other hand, it is well-known that the end-to-end delay also depends on the bandwidth. Hence, we changed the bandwidth of VM A to 4000 Kbps, and kept the added propagation delay at 400 ms. In this case, the broadcast latency difference is dramatically reduced by half. Our experiments reveal that divergent end-to-end delays can cause intolerable broadcast latency difference for fellow viewers. Such physical constraints as propagation delay and bandwidth limit however are not easy to be lifted for viewers, and we instead should seek for solutions within the broadcast platform.

III. Cross-Viewer Synchronization: Problem Formulation and Algorithm Design

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

A. Problem Formulation

We start from a streaming session consisting of one broadcaster and a set of m viewers (denoted by $\mathcal{V}$). A streaming server with bandwidth capacity c serves the viewers in this session, and the end-to-end throughput from the server to viewer $i \in \mathcal{V}$ is d_i. The original video rate generated by the broadcaster is R^*; the exact video rate r_i allocated to viewer i however is adjustable through transcoding, as long as it is not exceeding R^*. Obviously, we have the following constraint to guarantee that the buffer of viewer i will not experience underflow: $r_i \leq \bar{r}_i = \min(d_i, R^*)$.

For viewer i, we use l_i^t to represent the transmission delay for video data, and l_i^e to represent other network-related delays (referred to as the *network delay* in the rest of this paper), includ-

[7]For example, the recommended bitrate for 1080p videos in Twitch is 3000-3500 Kbps, plus an audio bitrate of 64-128 Kbps; the bitrate of standard quality 1080p YouTube videos is even higher, around 8,000 Kbps. For the emerging 4K videos, the bitrate requirement can be easily over 20 Mbps.

[8]"Twitch 2013 retrospective," [Online]. Available: http://www.twitch.tv/year/2013

[9][Online]. Available: http://twitchstatus.com/index.html

[10][Online]. Available: https://obsproject.com/

[11][Online]. Available: https://www.virtualbox.org/

ing propagation delay, processing delay and queuing delay. The end-to-end delay of viewer i is then[12]: $l_i = l_i^e + l_i^t = l_i^e + \frac{r_i}{d_i}$.

As in previous studies [13], [16], we consider the viewing experience of viewer i is given by a utility function $U_i(r_i)$, which is strictly concave, increasing, and continuously differentiable in r_i. For the streaming session, our objective is then to optimize the viewers' experience through rate adaptation (i.e., tuning streaming rate r_i of each viewer), and meanwhile ensure the difference between any pair of viewers' network delay is bounded by an empirical threshold (denoted by δ). Furthermore, the total streaming rates of all the viewers should not exceed the server's capacity c. This leads to the following network utility maximization (NUM) problem:

$$\begin{aligned} \max \quad & \sum_{i\in\mathcal{V}} U_i(r_i) \\ \text{s.t.} \quad & \sum_{i\in\mathcal{V}} r_i \le c \\ & \left(l_i^e + \frac{r_i}{d_i}\right) - \left(l_j^e + \frac{r_j}{d_j}\right) \le \delta, \qquad \forall i,j\in\mathcal{V} \\ & 0 \le r_i \le \bar{r}_i \qquad\qquad \forall i \in \mathcal{V}. \end{aligned} \tag{1}$$

Since the objective function in problem (1) is differentiable, strictly concave, and the feasible region is compact, the optimal solution exists (though it may not be unique) [17]. This convex problem can be directly solved in a centralized way via the classic simplex and interior point based algorithms [18], [19], provided that the streaming server has the information of each viewer, namely the end-to-end throughput d_i and the end-to-end delay l_i^e. It is however worth noting that the second constraint in problem (1) actually contains $m \times m$ inequalities; a centralized solver can therefore be very time-consuming for large sessions of thousands of current viewers, not to mentioning viewers' dynamic join and leave activities.

B. Distributed Algorithm Design

We now show an efficient distributed solution through dual decomposition [12].

We first obtain the Lagrangian relaxation [17] of problem (1). It is worth noting that directly relaxing the second constraint introduces $m \times m$ Lagrange multipliers, which, for a large-scale session, would incur massive message passing as well as significant computation. Rather, we consider the following compact form of this constraint:

$$\max_{i\in\mathcal{V}}\left(\frac{r_i}{d_i} + l_i^e\right) - \min_{j\in\mathcal{V}}\left(\frac{r_j}{d_j} + l_j^e\right) \le \delta \tag{2}$$

which is equivalent to

$$\begin{cases} \max\limits_{j\in\mathcal{V}}\left(\dfrac{r_j}{d_j} + l_j^e\right) - \left(\dfrac{r_i}{d_i} + l_i^e\right) \le \delta & \forall i\in\mathcal{V} \\ \left(\dfrac{r_i}{d_i} + l_i^e\right) - \min\limits_{j\in\mathcal{V}}\left(\dfrac{r_j}{d_j} + l_j^e\right) \le \delta & \forall i\in\mathcal{V}. \end{cases} \tag{3}$$

[12]To be more accurate, l_i^t should be equal to the total traffic during one time slot divided by the throughput, namely, $l_i^t = \frac{r_i \Delta t}{d_i}$. In this paper, we set Δt to one second, and hence is omitted for ease of exposition.

We use (3) to replace the second constraint in the original formulation, and obtain the Lagrangian form of problem (1) by relaxing the constraints while keeping the last one ($0 \le r_i \le \bar{r}_i, i \in \mathcal{V}$) as follows:

$$\begin{aligned} L(\boldsymbol{r},\lambda,\boldsymbol{\mu},\boldsymbol{\nu}) = & \sum_{i\in\mathcal{V}} U_i(r_i) - \lambda\left(\sum_{i\in\mathcal{V}} r_i - c\right) \\ & - \sum_{i\in\mathcal{V}} \mu_i \left[\max_{j\in\mathcal{V}}\left(\frac{r_j}{d_j} + l_j^e\right) - \left(\frac{r_i}{d_i} + l_i^e\right) - \delta\right] \\ & - \sum_{i\in\mathcal{V}} \nu_i \left[\left(\frac{r_i}{d_i} + l_i^e\right) - \min_{j\in\mathcal{V}}\left(\frac{r_j}{d_j} + l_j^e\right) - \delta\right] \end{aligned} \tag{4}$$

where $\lambda \ge 0$ and $\boldsymbol{\mu},\boldsymbol{\nu} \succeq 0$ are the Lagrange multipliers, or dual variables, which can be interpreted as the shadow prices associated with the corresponding inequality constraints. The dual function is then

$$g(\boldsymbol{r},\lambda,\boldsymbol{\mu},\boldsymbol{\nu}) = \max_{0\le r_i\le\bar{r}_i} L(\boldsymbol{r},\lambda,\boldsymbol{\mu},\boldsymbol{\nu}). \tag{5}$$

And the dual of problem (1) is defined as follows:

$$\begin{aligned} \min \quad & g(\boldsymbol{r},\lambda,\boldsymbol{\mu},\boldsymbol{\nu}) \\ \text{s.t.} \quad & \lambda \ge 0, \boldsymbol{\mu},\boldsymbol{\nu} \succeq 0. \end{aligned} \tag{6}$$

We solve it at two levels. At the lower level, each viewer solves the following subproblem:

$$\begin{aligned} & \max_{0\le r_i\le\bar{r}_i} \left\{U_i(r_i) - \lambda r_i + \mu_i\frac{r_i}{d_i} - \nu_i\frac{r_i}{d_i}\right\} \\ & = \max_{0\le r_i\le\bar{r}_i} \left\{U_i(r_i) - \left(\lambda - \frac{\mu_i - \nu_i}{d_i}\right) r_i\right\} \end{aligned} \tag{7}$$

which corresponds to maximizing the surplus (i.e., utility minus payment) of viewer i based on the aggregate price $\lambda - \frac{\mu_i - \nu_i}{d_i}$ of bandwidth. Given that the utility function is strictly concave, increasing, and continuous, the optimal solution to (7) is unique, denoted by

$$r_i^*(\lambda,\mu_i,\nu_i) = \arg\max_{0\le r_i\le\bar{r}_i}\left\{U_i(r_i) - \left(\lambda - \frac{\mu_i-\nu_i}{d_i}\right) r_i\right\}. \tag{8}$$

Each viewer then feedbacks the value of $r_i^*(\lambda,\mu_i,\nu_i)$ to the streaming server.

At the higher level, the streaming server solves the following problem by adjusting the dual variables λ, $\boldsymbol{\mu}$, and $\boldsymbol{\nu}$:

$$\begin{aligned} \min_{\substack{\lambda\ge 0 \\ \boldsymbol{\mu},\boldsymbol{\nu}\succeq 0}} g(\lambda,\boldsymbol{\mu},\boldsymbol{\nu}) = & \sum_{i\in\mathcal{V}} g_i(\lambda,\mu_i,\nu_i) + \lambda c + \sum_{i\in\mathcal{V}}(\mu_i - \nu_i) l_i^e \\ & - l_{\max}\sum_{i\in\mathcal{V}}\mu_i + l_{\min}\sum_{i\in\mathcal{V}}\nu_i \\ & + \left(\sum_{i\in\mathcal{V}}\mu_i + \sum_{i\in\mathcal{V}}\nu_i\right)\delta \end{aligned} \tag{9}$$

where $g_i(\lambda,\mu_i,\nu_i)$ is the maximum value of (7) for given values of λ, μ_i, and ν_i; the parameters $l_{\max}$ and $l_{\min}$ are defined as $l_{\max} = \max_{j\in\mathcal{V}}(\frac{r_j}{d_j} + l_j^e)$ and $l_{\min} = \min_{j\in\mathcal{V}}(\frac{r_j}{d_j} + l_j^e)$, respectively, which can be easily computed according to the feedback information of individual viewers, namely $r_i^*(\lambda, \mu_i, \nu_i)$. Problem (9) can then be solved through a subgradient method as follows:

$$\lambda^{(k+1)} = \left[\lambda^{(k)} + \alpha^{(k)}\left(\sum_{i\in\mathcal{V}} r_i^{(k)} - c\right)\right]^+ \tag{10}$$

$$\mu_i^{(k+1)} = \left[\mu_i^{(k)} + \alpha^{(k)}\left(l_{\max}^{(k)} - \frac{r_i^{(k)}}{d_i} - l_i^e - \delta\right)\right]^+ \tag{11}$$

$$\nu_i^{(k+1)} = \left[\nu_i^{(k)} + \alpha^{(k)}\left(\frac{r_i^{(k)}}{d_i} + l_i^e - l_{\min}^{(k)} - \delta\right)\right]^+ \tag{12}$$

where k is the iteration index; $\alpha^{(k)} > 0$ is the step size at the k-th iteration which is sufficiently small; $[\cdot]^+$ denotes the projection onto the nonnegative orthant; $r_i^{(k)}$ is short for $r_i^*(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$, namely the optimal solution to (7) for viewer i at the k-th iteration; the values of $l_{\max}^{(k)}$ and $l_{\min}^{(k)}$ are updated according to $\boldsymbol{r_i}^{(k)}$ obtained at the k-th iteration.[13]

The dual variables $\lambda^{(k)}$, $\boldsymbol{\mu}^{(k)}$, and $\boldsymbol{\nu}^{(k)}$ will converge to the corresponding dual optimal λ^*, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$ as $k \to \infty$, if the step sizes satisfy $\lim_{k\to\infty} \alpha^{(k)} = 0$, and $\sum_{k=1}^{\infty} \alpha^{(k)} = \infty$ [13]. For example, we can select $\alpha^{(k)} = \frac{t+1}{t+k}$, where t is a nonnegative constant [20]. Let us denote the maximum value of the primal problem (1) by z_p^*, and denote the minimum value of the dual problem (8) by z_d^*. According to the weak duality property [19], we have $z_p^* \le z_d^*$, and the difference between the optimal solutions of the primal and dual problems, namely $z_d^* - z_p^*$, is referred to as the *optimal duality gap*. Since we have assumed that the utility functions are continuous, increasing, and strictly concave, we have the strong duality $z_p^* = z_d^*$ (Recall that all the constraints in problem (1) are affine, and thus Slater's condition holds under the assumption that problem (1) is feasible [19].), which means that given the optimal dual variables λ^*, $\boldsymbol{\mu}^*$, and $\boldsymbol{\nu}^*$, the corresponding primal variables $\boldsymbol{r}(\lambda^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\nu}^{(k)})$ are also the optimal solution to the primal problem (1).

Algorithm 1 illustrates the distributed algorithm for problem (1) via dual decomposition. The stopping criterion is that the difference between the current value and the updated value for each of the variables λ, μ_i, and ν_i is below a certain threshold, namely $0 < \epsilon \ll 1$. It is worth noting that in the initialization phase, the streaming server can set identical values for $\mu_i^{(0)}$ and $\nu_i^{(0)}$, $\forall i \in \mathcal{V}$; while in line 5, the values of $\mu_i^{(k)}$ and $\nu_i^{(k)}$ would be different for different viewers in general.

Remark: Compared to a centralized solver that directly obtains the optimal solution to the primal problem (1) at the streaming server, our proposed Algorithm 1 is much easier to implement in practical systems. As mentioned before, the centralized solver requires the information of of all viewers' utility functions. Even such information is available, the centralized solver is not scalable to large sessions due to the complexity of the objective function, as well as the massive constraints. In Algorithm 1, on the other hand, only the dual variables are to be updated by the server through simple arithmetic computation, and each viewer only needs to solve a simple convex problem with a single constraint $0 \le r_i \le \bar{r}_i$. At the k-th iteration, the message passing overhead consists of the following two parts: all the viewers send back the updated value of $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$ to the streaming server, and the streaming server feedbacks the updated tuple $(\lambda^{(k+1)}, \mu_i^{(k+1)}, \nu_i^{(k+1)})$ to each viewer $i \in \mathcal{V}$. All the information can be easily piggybacked in normal packets, e.g., video content packets and ACK packets, with minimum overhead.

Algorithm 1: Distributed Algorithm for Single Server

```
Initialization: set iteration index $k = 0$; the streaming
    server broadcasts arbitrary positive initial values for
    $\lambda^{(0)}$, $\boldsymbol{\mu}^{(0)}$, and $\boldsymbol{\nu}^{(0)}$ to all viewers.
1: for each viewer $i \in \mathcal{V}$ do
2:     Locally determine the rate $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)}) =$
       $\arg\max_{0\le r_i\le \bar{r}_i}\{U_i(r_i) - (\lambda^{(k)} - \frac{\mu_i^{(k)}-\nu_i^{(k)}}{d_i})r_i\}$;
3:     Send the value of $r_i^{(k+1)}(\lambda^{(k)}, \mu_i^{(k)}, \nu_i^{(k)})$ to the
       streaming server;
4: end for
5: The streaming server updates the dual variables
     $\lambda^{(k+1)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\nu}^{(k+1)}$ according to (10)–(12),
     respectively, and updates the values of $l_{\max}^{(k+1)}$ and $l_{\min}^{(k+1)}$
     based on $\boldsymbol{r}^{(k)}$. The streaming server then sends the
     updated values of $\lambda^{(k+1)}$, $\boldsymbol{\mu}^{(k+1)}$, and $\boldsymbol{\nu}^{(k+1)}$ individual
     viewers;
6: Set $k = k + 1$ and go to 1 until the stopping criterion
     is satisfied.
```

To better reflect viewers personal preference, we add an option (interaction or not) for viewers. If a viewer selects interaction, s/he is allowed to submit comments to participate community interaction and is subject to our developed video bitrate adaption mechanism; if s/he selects no interaction, s/he is not to allowed to submit comments and her/his video bitrate is only adapted based on the network bandwidth and the server's capacity.

For dynamic viewer behavior and network conditions, we will first check if the viewer selects the video bitrate only based on the network condition and the servers capacity, her/his latency will lie within the threshold of the community, namely the second constraint problem (1), or not. If not, a lightweight solution is to only adapt this viewers bitrate based on the network condition and the servers capacity subject to the latency constraint. The proposed distributed bitrate adaptation algorithm may be conducted for all viewers periodically, say every ten minutes, to optimize the global watching experience.

IV. Prototype Implementation and Cloud Deployment

So far we have focused on the solutions for a single streaming server with abundant bandwidth capacity to serve all the viewers. For larger sessions with thousands of or even tens of thousands of viewers, which are common in the real world now, it is necessary to leverage a cluster of servers so as to guarantee reasonable user experience. In this context, migrating to a public cloud that offers elastic resource provisioning becomes a natural choice, particularly considering that such new media services, though

[13]"AWS case study: Netflix," 2015. [Online]. Available: http://aws.amazon.com/solutions/case-studies/netflix/

potentially grow very fast, have highly fluctuating user demands and limited upfront investment.

A. Cross-Viewer Synchronization in the Cloud

In our implementation, a collection of virtual machine (VM) instances from Amazon are leased for delivering video content to end viewers. VM instances are the basic units of resource provisioning in state-of-the-art public cloud service providers. They can be opened and configured in a relatively short time, and are charged by the actual running time, offering flexible *pay-as-you-go* pricing for cloud users. Given the cost of opening and maintaining a VM instance (i.e., *leasing cost*), we need an optimal strategy for VM resource provisioning that balances the operation costs and the QoS level for viewers.

For simplicity, in our prototype implementation, all VM instances have identical service capacity c as well as identical leasing cost f. Let n be the maximum number of VM instances that can be opened concurrently, which is bounded by the budget, and $\mathcal{S}$ be the set of this VM pool. We use a binary variable y_i to indicate the *provisioning decision* such that $y_i = 1$ if VM i is opened, and $y_i = 0$ otherwise; similarly, for viewer i, another binary variable x_{ij} indicates the *assignment decision* such that $x_{ij} = 1$ if viewer i is served by VM j, and $x_{ij} = 0$ otherwise.

We consider a single source scenario that each viewer can be served by only one VM, which leads to a constraint $\sum_{j\in\mathcal{S}} x_{ij} = 1\ (\forall i \in \mathcal{V})$. Furthermore, it is obvious that $x_{ij} \le y_j$ ($\forall i \in \mathcal{V}$ and $\forall j \in \mathcal{S}$), since x_{ij} can be positive only when VM j is opened.

Apart from the leasing cost of VMs, the cost of bandwidth usage should also be considered. Existing public cloud providers, e.g., Amazon EC2, typically charges on outbound traffic from the cloud only, while the traffic into the cloud as well as the traffic within the cloud is free. We consider a flat rate charging policy that the cost of outbound traffic (per second) is $p\sum_{i\in\mathcal{V}} r_i$, where p is the charge per unit traffic.

Now the extension problem that incorporates the leasing costs of VM instances and the traffic charge, is as follows:

$$\begin{array}{lll}
\max & \omega\sum_{i\in\mathcal{V}} U_i(r_i) - f\sum_{j\in\mathcal{S}} y_j - p\sum_{i\in\mathcal{V}} r_i & \\
\text{s.t.} & \sum_{i\in\mathcal{V}} r_i x_{ij} \le c & \forall j \in \mathcal{S} \\
& \sum_{j\in\mathcal{S}} x_{ij} = 1 & \forall i \in \mathcal{V} \\
& x_{ij} \le y_j & \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& \left(l_i^e + \frac{r_i}{d_i}\right) - \left(l_j^e + \frac{r_j}{d_j}\right) \le \delta & \forall i, j \in \mathcal{V} \\
& 0 \le r_i \le \bar{r}_i & \forall i \in \mathcal{V} \\
& x_{ij}, y_j \in \{0,1\} & \forall i \in \mathcal{V}, \forall j \in \mathcal{S}
\end{array} \tag{13}$$

where ω is a scalar weight parameter that reflects the trade off between utility and cost.

The optimal solution to the problem formulated above is difficult to obtain even in a centralized way, due to the integral constraints. In fact, a special case of this problem can be regarded as the *bin packing* problem [21]. In this case, the viewers' rates, namely r_i, are given, which satisfy the cross-viewer synchronization constraint, namely the fourth constraint in the extension problem. Then the first term and the last term in the object function are also fixed, and thus can be eliminated. Now the problem becomes

$$\begin{array}{lll}
\min & \sum_{j\in\mathcal{S}} y_j & \\
\text{s.t.} & \sum_{i\in\mathcal{V}} r_i x_{ij} \le c & \forall j \in \mathcal{S} \\
& \sum_{j\in\mathcal{S}} x_{ij} = 1 & \forall i \in \mathcal{V} \\
& x_{ij} \le y_j & \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& x_{ij}, y_j \in \{0,1\} & \forall i \in \mathcal{V}, \forall j \in \mathcal{S} \\
& \text{given } r_i & \forall i \in \mathcal{V}.
\end{array} \tag{14}$$

This is a bin packing problem, which is known to be NP-hard, and various approximation algorithms have been developed [21]. Consider the widely used LP-relaxation, which can be obtained by relaxing the integral constraints, namely $x_{ij}, y_j \in \{0,1\}$ to $x_{ij}, y_j \ge 0$ $(\forall i \in \mathcal{V}, \forall j \in \mathcal{S})$. A fractional y_j can be interpreted a partially opened VM, and a fractional x_{ij} can be interpreted a partial assignment of viewer i's traffic demand to VM j. The solution to the relaxed problem can be obtained through a centralized method, which is an upper bound of the optimal solution to (13). Rounding can then be used to obtain a feasible solution to (13), which is a lower bound of the optimal solution to (13).

Rather than developing the best approximate centralized algorithm for (13), we propose a distributed algorithm. The main idea is shown in Algorithm 2. In the first phase, the rate of each viewer $i \in \mathcal{V}$ is set to $\bar{r}_i$, and problem (13) now becomes the form of (14), namely a bin packing problem. The service provider then uses heuristic algorithms, for example, *Next-Fit* (online, running time $O(n)$, 2-approximation ratio) or *First-Fit-Decreasing* (offline, running time $O \log n$, 3/2-approximation ratio) [21], to determine the set of opened VMs (denoted by $\hat{\mathcal{S}}$) and the assignment of viewers to opened VMs. In the second phase, for each opened VM and the connected viewers, a NUM subproblem similar to (1) is solved in a distributed way using Algorithm 1, which updates the viewers' rates $\boldsymbol{r}$. In the third phase, the service provider uses the same heuristic algorithm to solve the bin packing problem, based on the newly obtained rates $\boldsymbol{r}$. The last two phases are repeated until the stopping criterion is satisfied.

There are some details to be further explained. First, in line 3, the leasing cost of an VM is amortized over the connected viewers' rates. As such, for an opened VM $j \in \hat{\mathcal{S}}$, each connected viewer $i \in \mathcal{V}_j$ solves the following subproblem:

$$r_i^* = \underset{0\le r_i\le\bar{r}_i}{\arg\max}\left\{\omega U_i(r_i) - \left(\frac{f}{\sum_{v\in\mathcal{V}_j} r_v} + p + \lambda_j - \frac{\mu_i - \nu_i}{d_i}\right) r_i\right\} \tag{15}$$

where $\sum_{v\in\mathcal{V}_j} r_v$ is the sum of all connected viewers' rates (namely the actual consumed bandwidth of the considered VM)

```
Algorithm 2: Distributed Algorithm for Cloud Deployment
 1: Set $r_i = \bar{r}_i, \forall i \in \mathcal{V}$; the service provider computes the
    values of $y_j$ and $x_{ij}$ using the Next-Fit algorithm, and
    assigns viewers to VMs accordingly;
 2: Each opened VM initializes dual variables and sends
    them to its connected viewers;
 3: for each opened VM $j \in \hat{\mathcal{S}}$ do
 4:     Each connected viewer $i \in \hat{\mathcal{V}}_j$ computes the rate of
        each viewer according to (15);
 5:     Compute local $l_{\max}$ and $l_{\min}$;
 6:     Send local $l_{\max}$ and $l_{\min}$ to the service provider;
 7: end for
 8: The service provider computes global $l_{\max}$ and $l_{\min}$, and
    broadcasts them to all opened VMs.
 9: Each opened VM updates dual variables;
10: Repeat lines 3-9 until convergence;
11: The service provider updates the values of $y_j$ and $x_{ij}$
    using the Next-Fit algorithm, and re-assigns viewers
    to VMs accordingly;
12: Repeat lines 3-11 until the stopping criterion is satisfied.
```

in the previous iteration, $\frac{f}{\sum_{v \in \mathcal{V}_j} r_v}$ is the leasing cost per unit consumed bandwidth, p is the charge per unit traffic, and λ_j, μ_i, as well as ν_i are the dual variables as the same in (7). It is worth noting that λ_j is computed at each opened VM j locally, while μ_i and ν_i are computed based on global $l_{\max}$ and $l_{\min}$. Second, each opened VM, which acts as the central streaming server in Algorithm 1, also needs to compute the value of $\sum_{v \in \mathcal{V}_j} r_v$, and sends it to the connected viewers. Third, for the heuristic algorithm solving the bin packing problem, we use the online *Next-Fit* algorithm to minimize the overhead such as redirecting viewers to different VMs, as well as to make timely decision.

The stopping criterion of Algorithm 2 is different from that of Algorithm 1. Algorithm 1 is guaranteed to converge, which is not the case in Algorithm 2. The reason is that in Algorithm 2, the heuristic *Next-Fit* algorithm does not ensure to find the optimal solution, so the computed assignment may require additional VMs for the newly obtained rates, which reduces the overall surplus and may lead to oscillation when running lines 3-11 iteratively. Recall that we start from the maximum number of VMs by setting the initial rate of each viewer to the maximum, and wish to find the optimal number of VMs which balances the operation costs and the QoS level for viewers. In practice, it is not feasible to frequently shut down and resume VMs, which incurs considerable operational delays caused by opening and configuring a VM, and it is also not cost-efficient to let idle VMs standby. Hence, one stopping criterion is that if additional VMs are needed by the heuristic algorithm, Algorithm 2 stops. If the number of VMs does not increase, lines 3-11 are repeated for a number of iterations.

Remark: Our proposed Algorithm 2 solves the extension problem (13) by dividing it into two subproblems that are solved alternatively. In the first subproblem, we assume that the viewers' rates are given, and minimize the number of opened VMs by solving a bin packing problem. This problem is solved at the service provider through a centralized approach that is scalable to very large systems. In the second subproblem, given the opened VMs and the assignment of viewers, we maximize the surplus of each opened VM through adapting the connected viewers' rates through the previously developed distributed Algorithm 1 (with slight modifications).

B. Moving Broadcasters to the Cloud

Now we discuss further optimizations for cloud-based live broadcast, so as to provide high quality video streams beyond amateur broadcasters' network bandwidth. In the new generation of Twitch-like social media, the crowdsourced video sources from amateur users remarkably stimulate the content diversity, which however also introduce new challenges to both content generation and content distribution. The source video quality of a channel, one of the most important factors determining the streaming quality received by viewers, is strictly limited by the broadcaster's upload speed. Considering that many of the amateur broadcasters rely on home networks, which have relatively low and unstable bandwidth (e.g., 512 Kbps to 2.5 Mbps upload bandwidth for typical home use Internet plans provided by ISP providers in Canada), uploading a high quality video stream in real time can be difficult or impossible to achieve. For example, the recommended bitrate for 1080p videos of OBS is 3000-3500 Kbps, plus an audio bitrate of 64-128 Kbps; the bitrate of standard quality 1080p YouTube videos is even higher, around 8,000 Kbps. For the emerging 4K videos, the bitrate requirement can be easily over 20 Mbps. Further, We have observed that for a broadcaster who simultaneously plays game and generates video streams, the high-fidelity video recording/encoding activities can consume a large portion of computation and network resources, and thus cause noticeable interference to the game experience, e.g., significantly lower frames per second (FPS).

To take a closer look at how the upload bandwidth quantitatively affects the received video quality, we have conducted a series of experiments with bandwidth control. We set up a testbed with one computer serving as the broadcaster. The broadcaster is connected to the Internet through a gigabit switch (Netgear GS108PEv2) that can limit the maximum bandwidth of specic links. We apply a Twitch account and use the broadcaster to stream a 10-minute Dota 2 game replay video to our Twitch channel. We use OBS as the video streaming and recording software, and adopt two recommended encoding settings: 720p at 2000 Kbps, and 1080p at 2800 Kbps. We enable the recording function of OBS, so the successfully uploaded frames would be also store locally. We repeat the experiments under different uploading bandwidth settings, namely 512 Kbps, 1 Mbps, 2 Mbps, and no limit. We obtain the frame loss ratio at the broadcaster under different bandwidth settings, which is shown in Fig. 2. We can see that the upload bandwidth is critical to the frame loss ratio of the source video. When the bandwidth is high, the result is perfect, no frame loss ratio. When the bandwidth is not enough for timely streaming, a significant portion of frames will be discarded; the frame loss ratio becomes higher when the gap

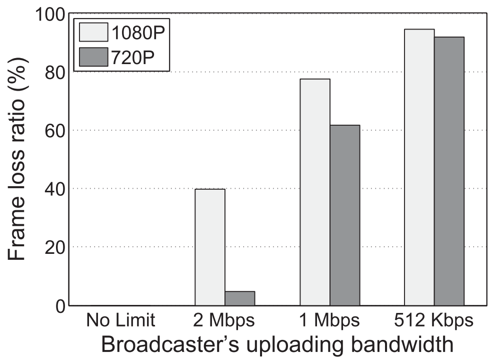


Fig. 2. Frame loss ratio under different bandwidth.

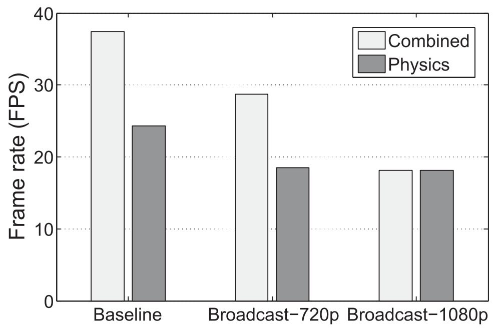


Fig. 3. Penalty on game experience (FPS stands for "frames per second").

between the upload bandwidth and the source video bitrate gets larger.

This observation poses challenges to both broadcasters and viewers in Twitch-like live broadcast systems. Broadcasters need to carefully set the video encoding rate, say not exceeding 80% of the nominal upload bandwidth. However, the nominal upload bandwidth is not always guaranteed. In fact, the achievable throughput between a certain broadcaster and the streaming server can be highly dynamic due to a lot of factors, e.g., the traffic congestion at the edge network, or the streaming server becomes overloaded. Buffering is useful to deal with slight variation of bandwidth; on the other hand, if the achievable bandwidth becomes constantly lower than the encoding rate for a relatively long period of time, the buffer would be quickly overflow, which results in intolerably high frame loss ratio. The broadcaster needs to lower the encoding rates for smooth playback, which however, degrades the received video quality at viewers. When some frames get lost, the viewers will experience a period of graphics freeze until a certain amount of new frames are received and played.It has been shown that such graphics freeze, together with the bitrate, have strong correlation with the user engagement in live video streaming [9], [10].

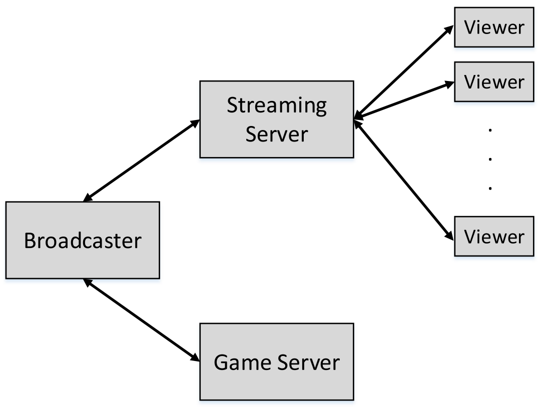
To examine the possible impact of live broadcast on gaming experience, and the frame rate drop in particular, we have run the 3DMark gaming benchmark with and without live streaming videos of 720p and 1080p, respectively. The benchmark contains a physics test that benchmarks the CPU-wise performance, and a combined test that benchmarks both the CPU-wise and GPU-wise performance, which allow us to understand how live broadcast activities affect the gaming experience. For the broadcaster, our test system is equipped with an Intel Haswell Xeon E3-1245 quad core processor, 8 GB 1600 MHz DDR3 main memory, 1 Gbps Ethernet card, and in particular a recently released NVIDIA GTX 970 Maxwell GPU with 4 GB GDDR5 memory. The measurement results are illustrated in Fig. 3, which show strong evidence that the live broadcast activities have noticeable negative influence on the game experience of broadcasters. For example, compared with the baseline without live video streaming activities, the performance penalty reaches about 25.3% on the combined test for 720p, and 52.1% for 1080p. The physics test incurs a penalty of around 26.5% and 27.3% for 720p and 1080p, respectively. Hence, the significant performance degradation triggers us to optimize the existing design of Twitch-like systems to alleviate the burden on broadcasters.

Our observations motivate us to optimize the architecture of the existing live broadcast systems, where the dual role of broadcasters as the content generator and publisher, as shown in Fig. 4(a), potentially throttles the video quality and may also incur interference. Inspired by the emerging cloud gaming techniques [24], [25],[14] we propose *ShadowCast*, which offloads the role of content publisher to the cloud, as shown in Fig. 4(b). In ShadowCast, a broadcaster exclusively acts a content generator that s/he plays games as a ordinary player and no longer needs to conduct video encoding/uploading. Instead, the broadcaster transmits the necessary *meta data* such as keyboard and mouse operations, game session information for multi-player games, as well as the web camera content to a *shadow client* deployed in a public cloud platform. The Shadow client, which also installs the game application, then reconstructs the game play graphics according to the received meta data, encodes the graphics together with the web camera content, and uploads them to the Twitch server which then uses CloudFront or other CDN platforms for global content distribution.

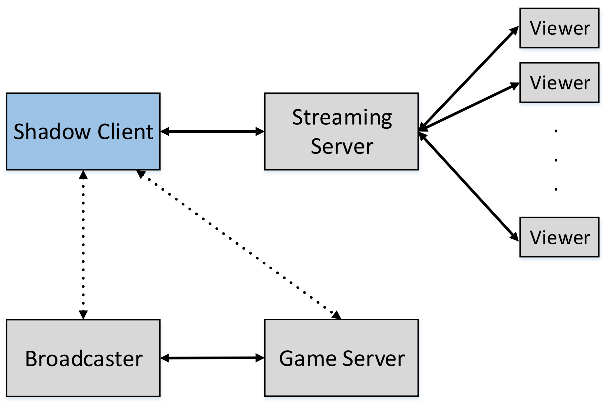
ShadowCast has three major benefits. First, the broadcasters do not need to perform complex video encoding/uploading tasks. Hence, the game experience will not be impaired. Second, the video quality is no longer limited by broadcasters' uploading bandwidth. Actually, ShadowCast can provide even better game graphics than that on the original broadcaster's computer if the shadow client is powerful enough. Considering Twitch-like service providers may directly deploy their servers on the cloud, the network bandwidth between the shadow client and streaming servers can be very high, which is more than enough to transmit 4K videos. Third, with ShadowCast, the service providers now have full control of the encoding rates of source videos, which makes it easier for global resource provisioning and allocation among different channels.

It is worth noting that here we just take gaming as an example to illustrate the design of ShadowCast, yet ShadowCast is not limited to the gaming scenario. Once the video content can be reconstructed from the meta data, the live broadcast service

[14]"Onlive," [Online]. Available: http://www.onlive.com/

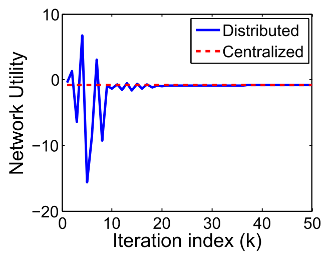


(a)

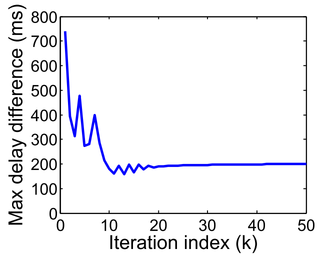


(b)

Fig. 4. Architecture of (a) existing and (b) ShadowCast live broadcast systems.

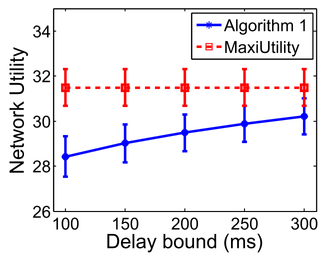


(a)

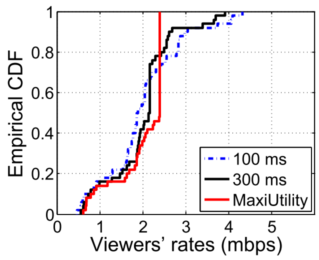


(b)

Fig. 5. Convergence of (a) network utility and (b) maximum delay difference.



(a)



(b)

Fig. 6. (a) Comparison of network utility. (b) CDF of viewers' rates.

provider can leverage ShadowCast for delivering high quality video content, minimizing the requirements on broadcasters.

V. Performance Evaluation

In this section, we first evaluate our proposed rate adaptation algorithms through extensive simulations. We then demonstrate the effectiveness of ShadowCast through testbed experiments on our proof-of-concept prototype.

A. *Single Server Scenario*

As for the proposed distributed Algorithm 1, we first examine its convergence, and then compare its performance against the centralized method. We use MATLAB 2013a to build up our customized simulator, and CVX, a package for specifying and solving convex programs [22], [23].

Our experimental settings are as follows. The bandwidth capacity of the streaming server is 10 Mbps, and the number of viewers is 10. Each viewer's bandwidth is uniformly distributed between 0.5 and 5 Mbps, and the network delay is uniformly distributed between 50 and 500 ms. The source encoding rate is 5 Mbps. The value of the end-to-end delay bound δ is 200 ms. We consider a widely used log utility function $U(r) = \log(r)$ for all viewers.

We first set the maximum number of iterations to 50, and illustrate the convergence of the network utility, namely the sum of all viewers' utility, and the convergence of the maximum delay difference in Fig. 5. We can see that our proposed distributed Algorithm 1 converges very fast; in about 25 iterations, the network utility is already close to optimum computed by a centralized solver, and the maximum delay difference already satisfies the cross-viewer synchronization constraint. Hence, we set the maximum number of iterations to 25 for Algorithm 1 in the remaining simulations.

We now compare Algorithm 1 with a baseline *MaxUtility* in a larger system. In MaxUtility, the network utility is maximized only subject to the bandwidth capacity of the streaming server, while the cross-viewer synchronization constraint is not considered. The bandwidth capacity of the streaming server is 100 Mbps, and the number of viewers is 50. We vary the end-to-end delay bound δ from 50 to 300 ms with the step size of 50 ms, and other settings are the same as before. We run each value of δ 10 times with random generated viewers' bandwidth/network delay.

We report the average and the standard deviation of network utility of Algorithm 1 and *MaxUtility* under different delay bounds in Fig. 6(a). Since the delay bound is not considered in *MaxUtility*, the obtained network utility does not change. The network utility obtained by Algorithm 1 becomes higher as the delay bound increases, since the feasible region of the optimization problem (1) also expands with larger delay bound. Although *MaxUtility* outperforms Algorithm 1 in terms of network utility, the real-time community interaction becomes intolerable. In our simulation, the maximum end-to-end delay difference of *MaxUtility* ranges from 826.0 to 914.4 ms, with an average of 861.4 ms. As evidenced by the observation in Section II, this

TABLE I
COMPARISON OF TOTAL SURPLUS ($)

		$\omega = 0.01$	$\omega = 0.05$
Algorithm 2	δ=100 ms	-2.576 (0.006)	10.964 (0.219)
	δ=300 ms	-2.363 (0.014)	13.421 (0.212)
	MaxSurplus	-2.137 (0.007)	14.264 (0.532)

TABLE II
COMPARISON OF VIEWERS' TOTAL RATES (MBPS)

		$\omega = 0.01$	$\omega = 0.05$
Algorithm 2	δ=100 ms	932.001 (2.899)	1168.140 (11.783)
	δ=300 ms	891.159 (2.211)	1319.338 (16.321)
	MaxSurplus	829.825 (5.446)	1367.254 (37.847)

level of delay difference would easily lead to tens of seconds broadcast delay among viewers.

We also plot the empirical cumulative distribution function (CDF) of the computed individual viewers' rates in Fig. 6(b). Recall that the utility function is $\log(r)$ for each viewer. Due to the concavity of the log function, the marginal utility of unit rate diminishes as the original rate increases. Hence, the maximum network utility will be obtained when the deviation of all viewers' rates is minimum. Hence, we can see that with MaxUtility, almost half viewers have the same rates, while the remaining viewers' rates already achieve the bandwidth limit. In fact, all viewers' rates will be equal if their bandwidth are all over 2 Mbps. As for the proposed Algorithm 1, due to the end-to-end delay bound, the viewers with shorter network delays will have larger rates, while the viewers with longer network delays will have lower rates. When the delay bound becomes larger, the viewers' rates tend to concentrate, in order to obtain higher network utility. We can see from Fig. 6(b) that the distribution of the viewers' rates gets closer to that of MaxUtility, namely the deviation of the viewer's rates becomes smaller, when the delay bound increases from 100 to 300 ms.

B. Cloud Scenario

We next evaluate Algorithm 2 that addresses the cross-viewer synchronization problem in the cloud. For such system parameters as VMs' bandwidth and leasing cost, as well as the outbound traffic charge, we take Amazon EC2 as our main reference: 200 Mbps stable network capacity of each VM, \$0.126 hourly leasing cost for each *m4.large* type On-Demand VM, and \$0.09 per GB outbound traffic from EC2 to Internet. We normalize the cost to one minute, which is line with the user engagement of video streaming services [9]. The number of viewers is set to 500, with the same bandwidth/network delay settings as in previous simulations. We compare our proposed Algorithm 2 with a baseline *MaxSurplus*, which is similar to Algorithm 2 yet does not subject to the cross-viewer synchronization constraint.

We investigate the resource provisioning with different values of the scalar weight parameter ω (0.01, and 0.05) under different delay bound (100 and 300 ms). The value of ω denotes the monetary reward (\$) per unit utility per minute. We run at most 25 iterations for per-VM surplus maximization, and 5 iterations for overall surplus maximization for both Algorithm 2 and MaxSurplus. For each setting, we run the simulation for 10 times. We report the total surplus and viewers' total rates in Tables I and II, respectively, in the format average (standard deviation).

We can see that MaxSurplus has the highest surplus in all settings, since it has the largest feasible region for the optimization problem, and the viewers' rates are more concentrated, as analyzed before. Similarly, the broadcast latency difference of MaxSurplus is not desirable for real-time community interaction. The average broadcast latency difference of MaxSurplus is 1058.2 and 448.7 ms on average, when $\omega = 0.01$ and 0.05, respectively.

As for the proposed Algorithm 2, an appropriate value of the end-to-end delay needs carefully examination, which balances the surplus and user experience. The reward factor ω plays an important role in Algorithm 2. A higher reward factor, say $\omega = 0.5$, would favor larger end-to-end delay which leads to higher network utility.

C. ShadowCast: Testbed Experiments

To conduct a synthetic evaluation on the ShadowCast framework, we choose the multi-player online game Dota 2 as the gaming application, which is one of the most popular games played and live-broadcasted nowadays. With the high-end system setup, we are allowed to play Dota 2 at 4K resolution (3840×2160) with the highest graphics and texture settings while achieving satisfying FPS.

To measure the bandwidth consumption, we set up a software router with Linux installed, and used the `nload` tool[15] to record the network behaviors. For the broadcaster, our test computer is equipped with an Intel Haswell Xeon E3-1245 quad core processor, 8 GB DDR3 1600 MHz RAM, 1 Gbps Ethernet card, and in particular a recently released NVIDIA GTX 970 Maxwell GPU with 4 GB GDDR5 memory. To collect game-specific data such as FPS, we configure and use the statistic tools provided by the Dota 2 game engine. Meanwhile, Dota 2 supplies a feature called the *Spectator Mode* where other gamer players, namely spectators, can watch live gameplay in a player' first person views, so every operation executed by the player will be forwarded to the Dota 2 server and then delivered to the spectators' own Dota 2 game engine to be replayed. We can therefore conveniently build up a proof-of-concept prototype of ShadowCast.

To establish a baseline, we set up the Dota 2 game engine to supply 4K resolution, the highest graphics effects, and a maximum of 120 FPS frame rate. We launched the game and carefully selected a normal game scene of Dota where there is only minor frame rate fluctuation (± 1 FPS). We then started the network bandwidth and frame rate measurement which lasts for 5 minutes. It is also worth noting that we fixed these game settings and selected scene across the following local-broadcast and ShadowCast experiments to make a consistent and fair comparison.

In the local-broadcast experiment, we also choose OBS as the broadcasting software. We configure it to capture and encode

[15][Online]. Available: http://www.roland-riegel.de/nload/

TABLE III
COMPARISON OF AVERAGE FRAME RATE (FPS) AND BANDWIDTH (KBPS)

Player Mode	Frame-rate	In-bandwidth	Out-bandwidth
Baseline	85	88.25	33.11
Local broadcast	68	168.12	3732.07
ShadowCast	85	85.60	33.91

frames in the CBR mode with 3500 Kbps and 2 key frames per second. The capture resolution will be down-scaled and down-sampled from 4K to 1080p at 60 FPS, as in reality viewers have lower frame rate and resolution demands than players. Noticeably these settings are also recommended by Twitch for broadcasting at 1080p.

On the other hand, for our ShadowCast prototype, locally we use the same setup as the local-broadcast one, except that we do not run broadcasting software on the broadcaster anymore. Instead we cast the game commands and controls to the remote Shadow Client, which is hosted remotely on the Amazon GPU virtual Instance (G2) in Oregon. We also install Dota 2 on the virtual instance and leverage the Spectator Mode to replay each and every operation of our local player does. We then start the OBS on the instance with identical settings to broadcast the game.

The experiment results are shown in Table III. As we can see, compared with the baseline without live broadcast, the local-broadcast, the default setup of the existing Twitch broadcaster, have a significantly degraded game experience for players/broadcasters; the FPS is nearly 20% lower. On the other hand, ShadowCast allows the broadcaster to play game with nearly zero performance penalty. In terms of network usage, we can see that ShadowCast consumes no more than the baseline, while the local-broadcast setting requires more than 3600 Kbps for uploading video streams. Here we did not involve the broadcaster's web camera content; yet it will not affect the advantage of ShadowCast. Considering that a 360p video only needs 600-800 Kbps after encoding, which is affordable by most broadcasters' network conditions, ShadowCast can still significantly outperform the local-broadcaster in both game experience and network usage.

VI. DISCUSSION

In our proposed cross-viewer synchronization approach, a viewer's video rate is assumed to be adjusted continuously between zero and the source rate that is configured by the broadcaster. Yet in such existing video streaming and broadcast platforms as YouTube and Twitch, the source video is normally encoded into several versions with different bitrates to accommodate users with heterogeneous network conditions. Our proposed solution can be adapted to this discrete video rate scenario in the following ways.

First, when computing the optimal solution of (8) for each viewer, we can replace the original domain of viewer's rate, which is continuous between 0 and $\bar{r}_i$, with the set of available video rates. When the number of video versions is small, the optimal video rate for each viewer in each iteration can be obtained efficiently by enumerating.

The second approach is to increase the number of video versions through advanced transcoding techniques. For example, through scalable video coding (SVC) [26], [27], viewers can adapt the requested video bitrate in a much broader range. Combined SVC with cloud-based online transcoding [28], [29], a viewer can be provided with a video stream with customized bitrate.

Another issue in our cross-viewer synchronization framework is the feasibility of the optimization problem. In fact, in our simulation, the range of network delay, which is from 50 to 500 ms, already covers a wide spectrum of network conditions, where a feasible solution can always be obtained. Theoretically, rate adaptation can tune the end-to-end delay to a maximum of one second. Considering a end-to-end delay bound of 200 ms, our problem is infeasible only when the network delay between a pair of viewers has a 1200 ms difference, which is a very rare case in practice.

VII. CONCLUSION

In this paper, we identified that the end-to-end delay has a remarkably amplified impact on viewers' broadcast latency. In order to achieve cross-viewer synchronization, which is necessary for real-time community interaction, an important feature in today's live broadcast services, we suggested smart rate adaptation, and develop distributed algorithms based on dual decomposition. We further extended our solution to the cloud environment, and presented the concept of ShadowCast, which moves broadcasters to the cloud to provide high quality streams beyond broadcasters' network bandwidth constraint. We evaluated the proposed interaction-aware rate adaptation algorithms through extensive simulations. A proof-of-concept prototype was implemented to demonstrate the practicability and effectiveness of ShadowCast, which clearly illustrates the promise of decoupling the dual role of broadcasters as the content generator and publisher.

For the future work, we first plan to investigate the HTTP streaming protocol and collect more detailed data, so as to identify the root cause of the amplified impact of end-to-end delay on the broadcast latency and develop possible optimizations. Second, we plan to implement our proposed rate adaptation algorithm in the existing HTTP streaming protocol, and deploy it on the cloud, to evaluate its performance in real systems. Third, we will extend our problem to the multi-source scenario, where multiple broadcasters at different locations collaboratively generate video sources, and the geo-distributed cloud is to be incorporated.

REFERENCES

[1] W. A. Hamilton, O. Garretson, and A. Kerne, "Streaming on twitch: Fostering participatory communities of play within live mixed media," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, Apr. 26–May 1, 2014, pp. 1315–1324.

[2] 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.

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

[4] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, 2011, pp. 157–168.

[5] Z. Wang *et al.*, "A joint online transcoding and delivery approach for dynamic adaptive streaming," *IEEE Trans. Multimedia*, vol. 17, no. 6, pp. 867–879, Jun. 2015.

[6] G. Gao, W. Zhang, Y. Wen, Z. Wang, W. Zhu, "Towards cost-efficient video transcoding in media cloud: Insights learned from user viewing patterns," *IEEE Trans. Multimedia*, vol. 17, no. 8, pp. 1286–1296, Aug. 2015.

[7] C. Zhou, C. W. Lin, and Z. Guo, "mdash: A markov decision-based rate adaptation approach for dynamic http streaming," *IEEE Trans. Multimedia*, vol. 18, no. 4, pp. 738–751, Apr. 2016.

[8] M. Carbone and L. Rizzo, "Dummynet revisited", *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 12–20, Mar. 2010. [Online]. Available: http://info.iet.unipi.it/ luigi/dummynet/

[9] F. Dobrian *et al.*, "Understanding the impact of video quality on user engagement," in *Proc. ACM SIGCOMM*, 2011, pp. 362–373.

[10] A. Balachandran *et al.*, "Developing a predictive model of quality of experience for internet video," in *Proc. ACM SIGCOMM*, 2013, pp. 339–350.

[11] M. Claypool and K. Claypool, "Latency can kill: precision and deadline in online games," in *Proc. ACM 1st Annu. ACM SIGMM Conf. Multimedia Syst.*, 2010, pp. 215–222.

[12] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006.

[13] J. Huang, Z. Li, M. Chiang, and A. K. Katsaggelos, "Joint source adaptation and resource allocation for multi-user wireless video streaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 8, pp. 582–595, May 2008.

[14] C. Liang, M. Zhao, and Y. Liu, "Optimal bandwidth sharing in multiswarm multiparty p2p video-conferencing systems," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1704–1716, Dec. 2011.

[15] H. Xu and B. Li, "A general and practical datacenter selection framework for cloud services," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 9–16.

[16] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *Proc. ACM SIGMETRICS*, 2008, pp. 169–180.

[17] F. Kelly, "Charging and rate control for elastic traffic," *Eur. Trans. Telecommun.*, vol. 8, no. 1, pp. 33–37, Jan./Feb. 1997.

[18] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Belmont, MA, USA: Athena Scientific, 2003.

[19] S. Boyd and L.Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[20] D. P. Palomar and M. Chiang, "Alternative distributed algorithms for network utility maximization: Framework and applications," *IEEE Trans. Autom. Control*, vol. 52, no. 12, pp. 2254–2269, Dec. 2007.

[21] E. G. Coffman, Jr., M. G. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems*. Boston, MA, USA: PWS-Kent, 1996, pp. 46–93.

[22] M. Grant and S. Boyd, *CVX: MATLAB Software for Disciplined Convex Programming, Version 2.0 Beta*. (Sep. 2013). [Online]. Available: http://cvxr.com/cvx

[23] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control (a tribute to M. Vidyasagar)*, V. Blondel, S. Boyd, and H. Kimura, Eds. New York, NY, USA: Springer-Verlag, 2008, pp. 95–110. [Online]. Available: http://stanford.edu/ boyd/graph_dcp.html.

[24] R. Shea, J. Liu, E.C.-H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

[25] R. Shea, D. Fu, and J. Liu, "Rhizome: Utilizing the public cloud to provide 3D gaming infrastructure," in *Proc. 6th ACM Multimedia Syst. Conf.*, 2015, pp. 97–100.

[26] Y. Sánchez *et al.*, "iDASH: Improved dynamic adaptive streaming over http using scalable video coding," in *Proc. 6th ACM Multimedia Syst. Conf.*, 2011, pp. 257–264.

[27] Z. Huang *et al.*, "Cloudstream: Delivering high-quality streaming videos through a cloud-based SVC proxy," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 201–205.

[28] Z. Li *et al.*, "Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices," in *Proc. 22nd Int. Workshop Netw. Oper. Syst. Support Digit. AudioVideo*, 2012, pp. 33–38.

[29] 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*, Apr.–May 2014, pp. 91–99.

Xiaoqiang Ma (S'12–M'16) received the B.Eng. degree from Huazhong University of Science and Technology, Wuhan, China, in 2010, and the M.Sc. and Ph.D. degrees from Simon Fraser University, Burnaby, BC, Canada, in 2012 and 2015, respectively.

He is currently an Assistant Professor with the School of Electronic Information and Communication, Huazhong University of Science and Technology. His research interests include wireless networking, multimedia, cloud, and big data.

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 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. degree (cum laude) 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. He is an EMC-Endowed Visiting Chair Professor with Tsinghua University, Beijing, China, and an Adjunct Professor in Tsinghua-Berkeley Shenzhen Institute, Shenzhen, China. From 2003 to 2004, he was an Assistant Professor with the Chinese University of Hong Kong, Hong Kong, China. His research interests include multimedia systems and networks, cloud computing, social networking, online gaming, big data computing, wireless sensor networks, and peer-to-peer networks.

Prof. Liu is an NSERC E.W.R. Steacie Memorial Fellow. He has served on the editorial boards of the IEEE TRANSACTIONS ON BIG DATA, the IEEE TRANSACTIONS ON MULTIMEDIA, the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, IEEE ACCESS, the IEEE INTERNET OF THINGS JOURNAL, *Computer Communications*, and *Wiley Wireless Communications and Mobile Computing*. He is currently a Steering Committee Chair of the IEEE/ACM IWQoS (2015–2017) and a TPC Co-Chair of the IEEE IC2E'2017 and the IEEE/ACM IWQoS'2014. He serves as an Area Chair of the IEEE INFOCOM, ACM Multimedia, and IEEE ICME. He was the recipient the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), the ACM SIGMM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), the ACM Multimedia Best Paper Award (2012), the IEEE Globecom Best Paper Award (2011), and the IEEE Communications Society Best Paper Award on Multimedia Communications (2009). His students were the recipients of the Best Student Paper Award of IEEE/ACM IWQoS twice (2008 and 2012).

Ryan Shea (S'08–M'16) received the B.S. and Ph.D. degrees in computer science from Simon Fraser University, Burnaby, BC, Canada, in 2010 and 2016, respectively.

He is currently a University Research Associate with Simon Fraser University, where he also completed the Certificate in University Teaching and Learning. His research interests include computer and network virtualization and performance issues in cloud computing.

Dr. Shea was the recipient of the Natural Sciences and Engineering Research Council of Canada Alexander Graham Bell Canada Graduate Scholarship in 2013. In 2012, he was the recipient of the Best Student Paper Award at the IEEE/ACM 21st International Workshop on Quality of Service for his paper titled "Understanding the impact of denial of service attacks on virtual machines."

Di Fu (S'14) is currently working towards the B.S. and M.S. degrees at Simon Fraser University, Burnaby, BC, Canada, in the dual degree program from Simon Fraser University and Zhejiang University.

His research interests include cloud computing topics surrounding virtualization, networking, and online gaming.