CloudFog: Leveraging Fog to Extend Cloud Gaming for Thin-Client MMOG with High Quality of Service

Yuhua Lin, *Member, IEEE* and Haiying Shen, *Senior Member, IEEE*

Abstract—With the increasing popularity of Massively Multiplayer Online Game (MMOG) and fast growth of mobile gaming, cloud gaming exhibits great promises over the conventional MMOG gaming model as it frees players from the requirement of hardware and game installation on their local computers. However, as the graphics rendering is offloaded to the cloud, the data transmission between the end-users and the cloud significantly increases the response latency and limits the user coverage, thus preventing cloud gaming to achieve high user Quality of Service (QoS). To solve this problem, previous research suggested deploying more datacenters, but it comes at a prohibitive cost. We propose a lightweight system called *CloudFog*, which incorporates "fog" consisting of supernodes that are responsible for rendering game videos and streaming them to their nearby players. Fog enables the cloud to be only responsible for the intensive game state computation and sending update information to supernodes, which significantly reduce the traffic hence the latency and bandwidth consumption. To further enhance QoS, we propose the reputation based supernode selection strategy to assign each player with a suitable supernode that can provide satisfactory game video streaming service, the receiver-driven encoding rate adaptation strategy to increase the playback continuity, the social network based server assignment strategy to avoid the communication interaction between servers in a datacenter to reduce latency, and the dynamic supernode provisioning strategy to deal with user churns. Experimental results from PeerSim and PlanetLab show the effectiveness and efficiency of CloudFog and our individual strategies in increasing user coverage, reducing response latency and bandwidth consumption.

Index Terms—Cloud gaming; P2P network; Online gaming; Quality of Service

1 Introduction

Massively Multiplayer Online Game (MMOG) (e.g., World of Warcraft, Second Life) allows users to inhabit in the same virtual world and interact with each other. It is characterized by a huge number of simultaneous players. At the height of its popularity, World of Warcraft had over 12 million users. Generally, MMOG uses the centralized client/server infrastructure, in which players need to install games, receive the game information from the servers, update game status and render new game videos [1]. MMOG requires players to have sufficiently powerful computers, which excludes users with thin clients such as tablets and smartphones. Buying and maintaining servers to support the tremendous number of players is cost-prohibitive to the game service provider. Nowadays, the number of smartphone users has been increasing rapidly and mobile gaming is also seeing the fastest growth among the gaming models [2]. The number of smartphones in use worldwide reached one billion during the third quarter of 2012, and will increase by another billion by 2015 [3]. Thus, thin-client MMOG is an inevitable trend of current MMOG and a cost-efficient system to support thin-client MMOG is desirable for game service providers.

Cloud gaming, as a flourishing gaming model, is a solution for thin-client MMOG, which frees players (we use players, clients, nodes and users interchangeably in this paper) from the requirement of hardware and game installation on their local computers. Nowadays, the cloud gaming is becoming a flourishing gaming model, with OnLive [4] and Gaikai [5] as two pioneers in cloud gaming. In cloud gaming, games are stored and run on remote servers, and game videos are streamed to end-users through broadband Internet connections. Cloud gaming also saves the cost of game service providers. They can buy cloud resource based on the actual demands in the large-scale system. Also, game service providers do not have to develop multiple versions of the same game to meet different operating systems (e.g., Linux, Windows, Mac), and spend money on software piracy protection.

Mobile gaming is seeing the fastest growth [2] in the gaming area. The advantages of cloud gaming makes it a very promising model to cater to the dramatically rapid growth of MMOG and online mobile gaming considering their very large user scale and thin clients. Though the advantages of cloud gaming makes it a very promising model to cater to thin-client MMOG, it currently faces severe challenges (i.e., latency, network connection, user coverage and bandwidth cost) that prevent it from becoming a leading gaming model. First, response latency is a critical factor in user quality of service (QoS). By offloading computation to a remote host, cloud gaming suffers from long *response latency*; the delay in sending the user action information and game video between the end-user and the cloud. Second, cloud gaming services post a strict requirements of high-speed network connection for a relatively high constant downlink bandwidth (e.g., 5Mbit/s recommended by OnLive). Third, the shortage of datacenters limits user coverage. Players begin to notice a response delay of 100ms [6]; 20ms attributed to playout delay on client side and processing delay on the cloud, 80ms attributed to the network latency. The playout delay of a client includes the time to send action information, receive and play the game video. Choy *et al.* [7] found that Amazon's EC2 (with 13 datacenters) can provide

• *Yuhua Lin and Haiying Shen are with the Department of Electrical and Computer Engineering, Clemson University, Clemson, South Carolina 29634.*
E-mail: {yuhual, shenh}@clemson.edu

Manuscript received April 19, 2005; revised September 17, 2014.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

a median latency of 80ms or less to only fewer than 70% of their 2500 tested end-users in the US. They also found that substantial increase in the total number of datacenters is required to significantly increase user coverage. Existing cloud infrastructure is not sufficient for hosting cloud gaming, as a sizeable portion of the population would experience significantly degraded QoS. Fourth, besides server time, bandwidth costs represent a major expense when renting on-demand resources. An average traffic of 27TB per 12 hours leads to about $130k monthly fee for bandwidth with Amazon EC2's price (i.e., $0.085 per GB) [8]. Considering the MMOG's huge user scale, these costs can significantly affect the feasibility of thin-client MMOG [9] on the cloud.

The great promises of cloud gaming and the obstacles it faces motivate us to explore approaches to efficiently handle the challenges. Though previous study suggested deploying more datacenters [6], building and maintaining a large number of datacenters is cost-prohibitive. In this paper, we propose a lightweight system called *CloudFog*. We introduce a concept called "fog", formed by powerful supernodes that are close to end-users and connected to the cloud. Considering that desktops, being idle for about 12 hours per day [10], [11], are underutilized in most organizations, the supernodes can be from these idle resources or from players' computers. In CloudFog, the intensive computation [1] of the new game state of the virtual world is conducted in the cloud. The cloud sends update messages to supernodes, the supernodes update the virtual world, render game videos for different players and stream videos to the players. Thus, users without high speed network connection to cloud or out of the coverage of the cloud can be supported by nearby supernodes, and the cloud does not need to transmit entire game videos to far-away users. This strategy can increase user coverage, shorten response latency, ensure relatively high-speed network connection for high QoS and reduce bandwidth cost. Specifically, CloudFog incorporates the following strategies to handle the challenges and enhance QoS.

(1) **Fog-assisted cloud gaming infrastructure.** We leverage the hardware and bandwidth capacity of some idle machines from players and organizations, and deploy them as supernodes. These supernodes constitutes the "fog", which are responsible to stream game videos for nearby players.
(2) **Reputation based supernode selection.** In order to assign each player with a suitable supernode that can provide satisfactory game video streaming service, each player calculates reputation scores for all candidate supernodes according to previous interactions, and selects a supernode that has high reputation score, available capacity and low transmission delay.
(3) **Receiver-driven encoding rate adaptation.** In order to ensure the playback continuity even in network congestion, when a supernode streams a game video to a player, it adaptively changes the encoding rate of the video based on the segment size in the player's buffer according to the game's tolerance on delay and packet loss.
(4) **Social network based server assignment.** The communication between servers in a datacenter for generating game videos leads to latency. As social friends always play together [12], we assign social friends who usually play together to the same server, so their interaction will not trigger communication between different servers, thus reducing response latency.
(6) **Dynamic supernode provisioning.** When a large number of players join a game within a short time during peak hours, the cloud servers face a heavy burden. In order to deal with user churns and reduce server loads, we dynamically predict the number of players and then determine the number of predeployed supernodes based on the predicted value.

This is the first work that uses lightweight approaches to handle the aforementioned challenges for cloud gaming to support thin-client MMOG. The remainder of the paper is organized as follows. Section 2 and presents an overview on the related work. Section 3 describes the detailed design of CloudFog. The performance evaluation is presented in Section 4. Section 5 concludes this paper with remarks on our future work.

2 Related work

MMOG on the client-server architectures has gained much attention in the research communities in recent years. Common approaches of MMOG divide the virtual environment into regions and assign each region to different servers [1]. Bezerra *et al.* [13] proposed a kd-tree mechanism to partition the game environment into regions, and perform load balancing among multiple servers based on the distribution of avatars in the virtual world. Many works proposed to leverage the bandwidth contribution of peer-to-peer (P2P) networks to reduce server load [14]. Ahmad *et al.* [14] presents a P2P live video system to help players share screen-captured video of their games. Chen *et al.* [15] proposed a content-oriented pub/sub system that exploits the network condition and end-systems to enable efficient player management and decentralized information dissemination. These P2P and information dissemination methods cannot be directly applied to the context of cloud gaming, in which each player receives its own game video that cannot be shared with other players. Also, the players with thin clients may not be able to conduct rendering, computation and storage [7], which are offloaded to the cloud.

Previous works developed different cloud gaming systems. GamingAnywhere [16] is the first open cloud gaming system with high extensibility, portability, and reconfigurability. Zhao *et al.* [17] designed a game cloud with a visualized cluster of CPU/GPU servers to reduce game latency of thin computers. Wang *et al.* [18] proposed to shift the burden of executing gaming engine from mobile devices to cloud servers. Hemmati *et al.* [19] presented a content adaptation encoding scheme, in which only the most important objects from the perspective of the player's activity are encoded in the scene and irrelevant or less important objects are omitted. To reduce the bandwidth consumption from the cloud to players, LiveRender [20] incorporates intra-frame compression, inter-frame compression and caching to achieve compressed graphics streaming in a cloud gaming system. This system only reduces the bandwidth when streaming game videos to players, while CloudFog aims to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

offload the streaming burden from the cloud to supernodes. EdgeCloud [21] augments the cloud infrastructure with a number of content delivery network (CDN) servers. These CDN servers have specialized resources and are located near end-users to increase user coverage; they are responsible for computing new game state and rendering game video for players. Compared to using stable but expensive CDN servers in EdgeCloud, CloudFog aims to deploy cheap idle resources from individual players and organizations. Given the same amount of revenue, *CloudFog* can deploy more servers than EdgeCloud by using proper incentives to motivate players or organizations to contribute their spare machines. Also, we have proposed several strategies to enhance the performance of *CloudFog*.

Early works also studied user experience in cloud gaming. Hobfeld *et al.* [22] discussed some technical challenges emerging from shifting gaming services to the cloud, and studied impacts caused by this change on user Quality of Experience (QoE). Jarschel *et al.* [6] conducted user studies to measure and model the QoS of OnLive during game play. Studies [6], [23] investigated how the response latency in cloud gaming affects QoS in various online games. There are also plenty of works on analyzing the challenges and benefits of system design in cloud gaming. Claypool *et al.* [24] studied detailed measurements of motion and scene complexity for a wide variety of video games, and measured the efficiency of streaming video games for thin clients. Choy *et al.* [7] demonstrated that the expansion of potential users for an on-demand gaming service is hindered by strict latency requirements, and indicated that the addition of a small number of servers can increase user coverage. Ojala *et al.* [25] studied a successful cloud gaming business model. They pointed out that deploying the games on cloud makes illegal copying practically impossible.

In spite of the previous research efforts on cloud gaming, except deploying more datacenters which is costly, no other approaches have been proposed to handle its critical challenges. We propose light-weight strategies to tackle the challenges to support thin-client MMOG.

3 System Design of CloudFog

3.1 Fog-assisted Cloud Gaming Infrastructure

Previous studies [6], [7] revealed that the uploading from the players to the cloud does not seriously affect the response latency, and downstream latency is an important factor for QoS [6], which is affected by the game video streaming rate. Thus, we aim to reduce the downstream latency by reducing the traffic transmitted from the cloud. In our design, game videos are streamed from nearby supernodes to players, instead of from remote game servers. As the computation of a virtual world for MMOG has a very high demand on server capacities [1], cloud is responsible for this task. Figure 1 shows our fog-assisted cloud gaming infrastructure. The fog is formed by supernodes, and normal nodes are connecting to their nearby supernodes. The normal nodes that cannot find nearby supernodes directly connect to the cloud.

We use n_i to denote a normal node, and sn_j to denote a supernode in the system. When each supernode is initially deployed, it is pre-installed with the *game client*. During the game playing, when node n_i makes an action

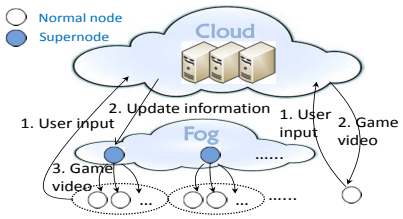


Fig. 1: Fog-assisted cloud gaming infrastructure.

(e.g., launching a strike or moving to a new place), this information is sent to the cloud server. The server collects action information from all involved players in the system and performs the computation of the new game state of the virtual world (including the new shape and position of objects and states of avatars). The cloud then sends the update information to the supernode of n_i (sn_j), which updates its virtual world accordingly. sn_j then renders game video for n_i based on $n_i's$ viewing position and angle. sn_j finally encodes the game video and stream it to n_i. As a player is close to its supernode in network distance, and the traffic from the cloud is significantly reduced, so the game video transmission delay is much shorter than that of downloading game video directly from the cloud as in the current cloud computing systems. Important notations used in this paper are listed in Table 1.

TABLE 1: Table of important notations.

n_i	a normal node
sn_j	a supernode
c_s	reward for one unit of bandwidth contributed by sn_j
$P_s(j)$	profit gained by supernode sn_j
c_j	supernode sn_j's upload capacity
u_j	sn_j's bandwidth utilization
$cost_j$	cost paid by sn_j's contributor in the same unit of c_s
$N(t)$	number of existing users at time t
n	number of normal nodes
m	number of supernodes
Λ	bandwidth usage for the cloud to send update information to one supernode
$G_s(j)$	game service provider's revenue gain by deploying sn_j
s_{ij}	overall reputation score of supernode sn_j evaluated by player n_i
r_k	k^{th} rating that n_i gives to sn_j
N_r	total number of ratings
λ	aging factor of ratings
d_k	age of rating r_k in days
q_i	game video quality for quality level i
b_{q_i}	game video bitrate for quality level i
$s(t_k)$	size of the video buffered at time t_k
r	number of video segments in the buffer
τ	game video segment size
β	game video bitrate adjust-up factor
θ	game video bitrate adjust-down threshold
ρ	latency tolerance degree
Γ	modularity of network communities

3.1.1 Requirements and Incentives for Supernodes

Rendering game video is relatively less hardware demanding than computation and communication in MMOG [21]; most modern computers with discrete graphics cards are sufficient to meet the rendering requirement. The nodes with sufficient hardware are chosen as supernodes, and the emerging technique of rendering multiple videos makes it possible for a supernode to suport multiple players simultaneously [26], [27]. As shown in [10], [11], desktop PCs in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

office are idle for about 12 hours per day, and 67% of desktop PCs remain powered on outside work hours (including nighttime). On the other hand, the number of players in online games reaches a peak during the nighttime [28], which matches the idle time of office desktop PCs. So the supernodes can be contributed by different organizations that have idle computer resources, and game players that have powerful computers can also be selected as supernode candidates. Besides, game service providers can deploy their own supernodes by placing servers in different areas or rent on-demand resources from existing cloud providers like Amazon EC2. A game client of MMOG usually takes about 5-6GB storage space, and it is pre-installed in the supernode. The supernodes are required to be: 1) reliable, as malicious supernodes may distribute spam or virus that may degrade player experience or harm players' machines; 2) stable, supernodes need to provide stable support and notify the central server of game service providers before leaving the system; and 3) superior network connection, as supernodes need to stream game videos to players within short latency. To satisfy these requirements, organizations and individual players need to provide credentials to game service providers, game service providers will verify the information of supernode contributors and have contracts with them. The purpose of this contract is to ensure that supernodes can provide high QoS for players and will not leave the system abruptly during their service time. Contributing a machine as a supernode generates costs of running the machine (e.g., electricity and maintenance costs). Therefore, to incentivize other organizations and players to contribute supernodes, an incentive mechanism is needed to reward supernodes based on the amount of upload bandwidth they contribute. The reward can be in the form of real money or virtual money for online games, and we use c_s to denote the reward for each bandwidth unit contributed by a supernode. An organization or a player considers to contribute a supernode only when it brings about certain profit, which is calculated by subtracting its running costs from its earned rewards. We use $P_s(j)$ to denote the profit gained by supernode sn_j:

$$P_s(j) = c_s \times c_j \times u_j - cost_j, \tag{1}$$

where c_j represents sn_j's upload capacity, u_j denotes sn_j's bandwidth utilization, and $cost_j$ denotes the cost paid by sn_j's contributor in the same unit of c_s. $P_s(j)$ quantifies the profit of contributing a supernode. Contributing a supernode is lucrative when $P_s(j)$ is greater than a certain threshold (different contributors set their own thresholds based on their expectations on profits). Then, the supernode's owner is motivated to contribute this supernode. Also, studies in [10], [11] found that 67% of desktop PCs remain powered on when they are idle, so there are powered idle PCs available to serve as supernodes. Companies and organizations are motivated to earn rewards by contributing these idle resources. Though some desktop PCs do not always serve players, as long as they function as supernodes in CloudFog, they can still receive a small amount of monthly sign up bonus. When they contribute bandwidth and support players, they can receive more credits. We will evaluate the effectiveness of this incentive mechanism in Section 4.

3.1.2 Economic Benefits for Game Service Providers

The game service provider needs to guarantee that the money spent on rewarding supernodes is smaller than the bandwidth costs saved by the contribution of supernodes. We use $N(t)$ to denote the number of existing users at time t. For simplicity, we omit t in the notations. Given the streaming rate of game video R, the total system demand for bandwidth equals $N \times R$. Suppose there are m supernodes, each having c_j upload capacity with utilization u_j. Then, supernode bandwidth contribution equals $B_s = \sum_{j=1}^{m} c_j \times u_j$. We use Λ to denote the bandwidth usage for the cloud to send update information to one supernode, and use n to denote the number of users that supernodes support. Then, in CloudFog, the bandwidth consumption for one player action for nodes connecting to supernodes equals $\Lambda \times m$, and that for users directly connecting to the cloud equals $(N-n)R$. The bandwidth reduction (B_r^-) of CloudFog compared to current cloud computing system equals:

$$\begin{aligned} B_r^- &= N \times R - \Lambda \times m - (N-n)R \\ &= n \times R - \Lambda \times m \end{aligned} \tag{2}$$

Suppose c_c is the revenue gained by saving each server bandwidth unit, the goal of the game service provider is to maximize the saved cost by leveraging supernode bandwidth contribution, which can be formulated as below.

$$\begin{aligned} C_g &= \max(c_c \times B_r^- - c_s \times B_s) \\ &= \max\{c_c[n \times R - \Lambda \times m] - c_s \times B_s\} \end{aligned} \tag{3}$$

$$s.t. \ \sum_{j=1}^{m} c_j \times u_j \geq n \times R \tag{4}$$

$$u_j \leq 1, \ \forall j \in \{1, 2, ...m\} \tag{5}$$

Equation (4) guarantees that the total supernode bandwidth contribution must reach the required node support bandwidth, while Equation (5) restricts the utilization of a supernode's upload bandwidth within its bandwidth capacity. In Equation (3), we see that given a specific number of n (*i.e.* the coverage of normal nodes is determined), saved cost C_g increases when m decreases; that is, a smaller number of supernodes lead to higher cost saving. For the game service provider, it should consider the pay and gain before deploying a supernode. Suppose a new supernode sn_j is deployed in an area; as a result, the coverage of players supported by supernodes is increased by ν new players. We use $G_s(j)$ to denote the game service provider's revenue gain by deploying sn_j, and $G_s(j)$ is estimated by:

$$G_s(j) = c_c[\nu \times R - \Lambda] - c_s \times c_j \times u_j. \tag{6}$$

If $G_s(j) > 0$, the cost of deploying supernode sn_j is surpassed by the benefit of bandwidth saved from the ν new players supported by sn_j.

3.2 Reputation Based Supernode Selection

3.2.1 Supernode Reputation Management

We define supernode sn_j's capacity as the maximum number of normal nodes that sn_j can support. Though supernodes are encouraged to contribute their computation and bandwidth resources to support other players' gaming activities, a supernode's quality of service to a given

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

player is affective by three factors. First, there exits capacity heterogeneity among supernodes due to different upload bandwidth and computation power. A supernode may not be able to support all game video streaming requests with satisfactory QoS due to its limited available capacity. Second, different supernodes have different physical distances and transmission delays to a given player. Third, a supernode may not be willing to support players and deliberately throttles its upload bandwidth under certain circumstances. For example, when a supernode's owner runs many applications on the supernode, the owner may not be willing to support many players. Thus, jointly considering these three factors in selecting reliable supernodes is crucial to provide players with high QoS during gaming activities. To consider the third factor, we use a reputation system, which is an effective tool to guide the selection of supernodes that are willing to be cooperative in providing game video streaming service. In the reputation system, a player evaluates its supernode's reputation based on its performance in providing fluent game video streaming (i.e., playback continuity).

To facilitate the first two factors, the cloud stores the information of supernodes in the system in a table including their IP addresses and available capacities. When a newly joined node n_i requests a supernode, the cloud returns a number of supernodes that have available capacities and are physically close to player n_i by referring to the table. To do this, it first identifies the supernodes with available capacities. It then calculates the distance between each of the supernode candidates and the player, and selects a certain number of physically close supernodes. To calculate the distance, the cloud uses a supernode's IP address [29], [30] to determine its coordinate, and then uses the coordinate to calculate its distance from a player.

A physically close supernode may not guarantee a short transmission delay. Therefore, after the newly joined node n_i receives its close supernode candidates from the cloud, it tests the transmission delay to all of them. It then removes candidates with transmission delay greater than its threshold L_{max_i}, which is determined based on the response latency requirement of the genre of its game [31]. This threshold is used to ensure that its supernode is capable of providing quick streaming support. From the remaining supernode candidates, node n_i then chooses the supernode with the highest reputation score. If there are no remaining supernode candidates, n_i directly connects to the cloud. Below, we describe the details of the reputation system.

To evaluate a supernode's willingness to be cooperative, a straightforward scheme is to evaluate a supernode's overall reputation by gathering opinions from all players interacted with this supernode [32]. However, this scheme is vulnerable to sybil attack [33], where a malicious supernode forges multiple identities and gains advantage by receiving high ratings from these identities. Also, it cannot prevent collusion in which a collective of nodes intentionally rate each other with high scores. To circumvent these problems, we let each player use its own evaluation without gathering opinions from other players.

Specifically, a player evaluates its supernode's performance in providing fluent game video streaming service after each game. It periodically calculates the overall reputation scores of its supernodes that provided it game video streaming service. As recent interactions between players and supernodes can more accurately reflect the supernodes' future performance than earlier interactions, we weight the ratings according to their ages when calculating a supernode's overall reputation score. Each rating is associated with an age measured by the number of days that have passed since the rating is given. We use s_{ij} to represent the overall reputation score of supernode sn_j evaluated by player n_i. It is computed as the weighted average of all ratings that sn_j receives from n_i:

$$s_{ij} = \sum_{k=1}^{N_r} r_k \lambda^{d_k}, \ (0 < \lambda < 1), \tag{7}$$

in which r_k is the k^{th} rating that n_i gives to sn_j, N_r is the total number of ratings, λ is the aging factor used to control the weights of ratings according to their ages, and d_k is the age of rating r_k in days. The reputation scores of the supernodes that have no previous interactions with the player equal to 0. The computation complexity of calculating reputation scores for all supernodes is $O(mnN_r)$, where m and n are numbers of supernodes and normal nodes.

3.2.2 Player and Supernode Churns Management

Recall that player n_i receives a number of supernode candidates from the cloud when it joins the system. In order to select supernodes with high reputations, it orders these candidates in descending order of their reputation scores. During the time when a player selects a supernode, a supernode candidate may be connected by more players and no longer has available capacity. To ensure that the selected supernode has available capacity, the player sequentially asks the supernodes in the ordered list whether it has available capacity. Once a supernode has available capacity, the player selects this supernode to connect to. If no supernode is selected after the player examines all supernode candidates, the player connects to the cloud for game video streaming.

Normal nodes probe their supernodes periodically for connection maintenance. When a normal node disconnects from its supernode, it first tries to find qualified supernode from its candidate supernode list by choosing the one with high preference ranking and available capacity. If it fails to find a new supernode from the candidate list, it contacts the cloud to find a new supernode using the method introduced above. When a new supernode is deployed, that is, a player or organization devotes a spare machine to earn rewards, the machine's location is identified based on the IP address. The cloud then notifies the normal nodes that are physically close to the new supernode, and these normal nodes will test the transmission delay to the newly deployed supernode. Finally, the suppernode will be added to the normal node's supernode candidate list if the transmission delay is less than L_{max_i}.

3.3 Receiver-driven Encoding Rate Adaptation

A player stores its received segments into its buffer while playing the game video. To guarantee the continuity of the video playback, the player needs to continuously fetch segments from the buffer and play. Game video bitrate affects the number of video segments received by a player

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

TABLE 2: Video parameters for different quality levels.

Quality level	Video resolution	Video bitrate	Latency requirement	Latency tolerance degree
5	1280x720	1800 kbps	110 ms	1
4	720x486	1200 kbps	90 ms	0.9
3	640x480	800 kbps	70 ms	0.8
2	384x216	500 kbps	50 ms	0.7
1	288x216	300 kbps	30 ms	0.6

during a unit time period, hence the player's playback continuity. Thus, we can adjust game video bitrate based on the size of buffered segments. The game video can be encoded to different bitrates based on the requirements on pixel size (resolution), hence the video quality level. A video segment with a higher quality level (i.e., a higher bitrate) leads to longer transmission latency. In order to illustrate the differences in video resolution, bitrate, latency requirement and tolerance degree for videos with different quality levels, we generate Table 2 as an example of parameter settings. Note that the parameter values in Table 2 are not precise, and the game service providers can set the parameters based on actual needs. As shown in Table 2, 500kbps corresponds to 384x216 resolution, and such a segment leads to 50ms latency. We use q_i ($i \in [1, ..., Q]$) to denote the video quality for quality level i and use b_{q_i} to denote the corresponding bitrate.

Different genres of games have different requirements on response latency [23]. Based on Table 2, if a game video has a latency requirement of 90ms, the supernode should use 1200kbps encoding bitrate, corresponding to a quality level of 4. To reduce the latency of the game video under unfavorable network condition, the supernode can choose encoding bitrates corresponding to quality level lower than 4; that is, sacrificing quality for lower latency. Due to unexpected network condition (e.g., network congestion), packets may be transmitted at a lower speed. Users may prefer fluent play of the game though the game video gets a bit blur when the encoding rate is reduced. To provide flexible options, users can also disable the encoding rate adaptation strategy before they start the game. In this case, the game video rate is fixed to the game's default video rate.

We aim to ensure that the playback rate is always lower than or equal to the segment downloading rate. When this condition cannot be satisfied, the video quality needs to be reduced by one level. When the size of buffered video at current quality level q_i is expected to reach the size of buffered video at quality level q_{i+1} (i.e., the downloading rate is faster than the playback rate), the current encoding bitrate b_{q_i} can be increased to $b_{q_{i+1}}$ to increase the video quality to q_{i+1}. Below, we explain the details of the adjustment operation. The estimated size of the video buffered at time t_k (denoted by $s(t_k)$) is calculated by:

$$s(t_k) = s(t_{k-1}) + (t_k - t_{k-1})(d(t_k) - b_p(t_k)), \quad (8)$$

where $d(t_k)$ and $b_p(t_k)$ denotes the downloading rate and video playback rate at time t_k. We use r to denote the number of segments in the buffer:

$$r = \frac{s(t_k)}{\tau} = \frac{s(t_{k-1}) + (t_k - t_{k-1})(d(t_k) - b_q(t_k))}{\tau}, \quad (9)$$

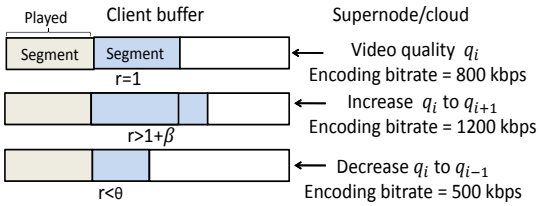


Fig. 2: Receiver-driven encoding rate adaptation.

where τ denotes video segment size. If

$$r > 1 + \beta, \quad (10)$$

the video bitrate adjusts up. β is an adjust-up factor, and

$$\beta = max\{(b_{q_{i+1}} - b_{q_i})/b_{q_i}, \forall i \in [1, 2, ..., Q]\}. \quad (11)$$

β guarantees that the size of the buffered segments reaches that of the incremented quality level. When the video bitrate adjusts up, the user will not suffer from playback delay during the game. The adjust-down operation is performed if

$$r < \theta \ (\theta \leq 1), \quad (12)$$

where θ denotes adjust-down threshold. Formula (12) enables to proactively adjust down video bitrate to ensure the playback continuity in network congestion, in which the segment transmission time is typically much longer than usual. In order to prevent the fluctuation of the video bitrate for a client, the client can conduct the calculations of r for a number of times consecutively. The video bitrate is adjusted only when all results satisfy Formula (10) or Formula (12). Figure 2 shows an example of the encoding rate adaptation. When $r > 1 + \beta$ for several consecutive estimations, the supernode increases the video encoding quality by one level; from 800kbps to 1200kbps for the player. When $r < \theta$, the supernode decreases the video quality by one level; from 800kbps to 500kbps.

Different games have different latency-tolerant degree [23]. We consider this property to further enhancing the probability of meeting the response latency requirement for different games. Specifically, we require higher latency-sensitive games to have larger buffered video size for the encoding rate adjustment. We use $\rho \in [0, 1]$ to denote the latency tolerance degree; higher ρ means higher latency tolerance degree. We then change Formula (10) to $r > (1+\beta)/\rho$ and change Formula (12) to $r < \theta/\rho$ for triggering encoding bitrate adjustment. As a result, latency-sensitive (lower latency-tolerant) games have a higher r threshold while latency-tolerant games have a lower threshold for adjusting the encoding bitrate.

3.4 Social Network Based Server Assignment

A cloud datacenter consists of many servers, which cooperate to accomplish the computation and storage function of the datacenter. Therefore, when multiple players assigned to different servers interact with each other in the game (e.g., fighting each other in a battle), their servers need to communicate with each other in order to receive game states of all players and compute the game state of the virtual world. Online games involve intensive player interactions, if two players are assigned to different servers within datacenters, the interactions among these two players will lead to communications between the two servers. For example,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

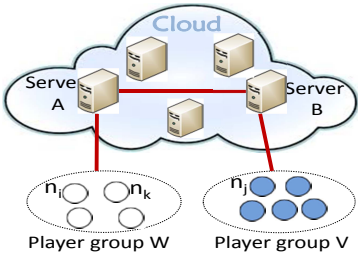


Fig. 3: Server assignment based on social networks.

in Figure 3, when player n_i is playing with player n_j, it will result in communication between server A and server B. Such server communications contribute to the response latency and degrade QoS. Thus, we propose the social network based server assignment strategy to reduce the interactions between servers by assigning players who are likely to play games together to the same server. In Figure 3, if n_i is playing with n_k, their interactions will not lead to server interactions.

When a new player signs up in CloudFog, if it builds friendships with other players, this new player is assigned to a server that most of its friends are allocated to; otherwise, it is randomly assigned to a server. To improve the accuracy of friend clustering and reduce the interactions between servers, we propose the social network based server assignment strategy that runs periodically (e.g., weekly) to reassign players to servers.

The players in the system can be represented by an undirected graph $G = (V, E)$; V is the set of players and E is the set of edges between the players. $e_{ij} = 1$ if player n_i is a friend of player n_j. We use a set $F(i)$ to store all friends of n_i. The friendship relationship can be determined by two schemes: explicit friendship and implicit friendship. 1) Studies in [12] show that players tend to build friendship with each other in online computer games and players tend to play with their friends. By "friends" we mean that players who build friendship in the game. 2) CloudFog keeps record of each user's playing activities (e.g., who they are playing with, how long do they play), when the number of times that two players play together within the recent week CP_{ij} is larger than a threshold υ, we regard it as an implicit friendship. Thus, given z servers, this problem turns to finding z network communities (also sometimes referred to as modules or clusters), and the game information of each community is allocated in a distinguished server.

Modularity Γ is commonly used to evaluate the quality of resultant network communities [34]. It first defines a $z \times z$ symmetric matrix $\mathcal{Q}$, whose element q_{ab} is the fraction of edges connecting community A and B over $|E|$, then calculates Γ by:

$$\Gamma = \sum_{b=1}^{z}(q_{ab} - p_a^2) = tr(\mathcal{Q}) - ||\mathcal{Q}^2||. \quad (13)$$

$tr(\mathcal{Q})$ is the trace of matrix $\mathcal{Q}$; $p_a = \sum_{b=1}^{z} q_{ab}$; and $||X||$ is the sum of elements of matrix X. High value of Γ indicates a good community clustering, in which the game information of friends are likely to be allocated to the same server. Existing works generally partition a user and its friends to the same server by using replication [35], that is, a user's data is copied to multiple servers. These approaches are not applicable in the gaming area where a single copy of each player's data (e.g., profile data and game status data) is kept on a server to avoid the synchronization of user data on different servers. CloudFog first greedily assigns a player and its friends to the same community; then in order to optimize the community structure (i.e., increase the value of Γ), it repeatedly selects some players and switches their communities. The performance of community clustering and computation overhead can be controlled by setting different number of repetitions. We present the detailed steps below.

At first, all players are assigned to one community g_1, then we divide it into z communities using the following steps. 1) Randomly select a player n_i and put it and all its friends into a new community g_2. 2) Select a random play n_j from g_2, and put $F(j)$ (n_j's friends) into g_2. 3) Repeat step 2 until the number of player in g_2 is larger or equal to $|V|/z$. 4) Repeat step 2 and 3 to assign all players into z communities. 5) Calculate the modularity Γ_{pre} for current communities. Randomly select player n_i and n_j from 2 random communities, swap the communities of $n_i + F(i)$ and $n_j + F(j)$, and calculate the current modularity Γ_{cur}. If $\Gamma_{cur} > \Gamma_{pre}$, swapping communities for players $n_i + F(i)$ and $n_j + F(j)$ leads to a better communities structure, so we keep the current structure; otherwise, we call it a **Miss** and rollback the swapping operation. 6) Repeat step 5 by h_1 times or until there are h_2 consecutive **Miss** ($h_2 < h_1$). Assuming $z^2 > |E|$, the computation complexity of calculating Γ_{cur} and Γ_{pre} is $O(z^2)$. Therefore, the computation complexity of this server assignment method is $O(h_1 z^2)$.

3.5 Dynamic Supernode Provisioning

In MMOGs, the number of online players generally varies with a diurnal pattern [36], [37]. When a large number of players join a game within a short time during peak hours, the surge in player arrival rate places a heavy burden on the cloud servers. MMOG designs should take into account the dynamicity of players and minimize the overhead of the cloud servers during peak hours. Provisioning supernodes to assist the cloud in game video streaming is an effective way to deal with player dynamicity.

In our dynamic supernode provisioning algorithm, the cloud pre-deploys a sufficient number of supernodes before the peak time to support players and removes these supernodes after the peak time. These supernodes serve newly-arrived players' requests and thus mitigate the peak bandwidth demand towards the cloud. A key challenge in our algorithm is to determine the number of supernodes that should be pre-deployed. If the game service provider reserves an excessive number of supernodes from players and organizations, some of them may be idle. If an insufficient number of supernodes are reserved, most requests for game video streaming will still be served by the cloud during peak hours.

We predict the number of players and then determine the number of pre-deployed supernodes based on the predicted value. Accurate prediction of online players is possible since previous works [36], [37] show that the workload of MMOGs has a regular weekly pattern and week-to-week load variations of players are less than 10%, for example, the trend of this Friday's online players mirrors that of last Friday. Thus, we can forecast the number of online

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

players based on the data from previous weeks and reserve sufficient supernodes in advance. This forecasting and reservation process can be carried out at a frequency of every m-hour time window. Then, each week is divided into $24 * 7/m$ (denoted by T) time windows. We use $\hat{N}_t$ to denote the expected number of players in time window t. $\hat{N}_t$ can be predicted from the number of players at the same time of last week, i.e., N_{t-T}. We use the seasonal ARIMA model [38], which is widely used to forecast time series with seasonal patterns, to forecast the number of players. It predicts $\hat{N}_t$ based on the data of the current time window (i.e., $t-1$) and data of the same time windows of last week (i.e., $t-T$ and $t-T-1$),

$$\hat{N}_t = N_{t-T}+N_{t-1}-N_{t-T-1}-\theta W_{t-1}-\theta W_{t-T}+\theta\Theta W_{t-T-1}, \quad (14)$$

in which θ is the moving average MA(1) coefficient and Θ is the seasonal moving average SMA(1) coefficient. $\{W_t\} \sim WN(0,\sigma^2)$ is a sequence of white noise with zero mean and variance σ^2. To support $\hat{N}_t$ number of players, the number of supernodes that need to deploy (denoted by Ns_t) is calculated by:

$$Ns_t = (1+\varepsilon)\hat{N}_t/\hat{C}, \quad (15)$$

where $\hat{C}$ is the average capacity of supernodes, and ε is the scale factor for the number of supernodes.

After determining the number of supernodes, the game service provider needs to select supernodes from available candidates. In order to maximize the number of players supported by the supernodes and utilization of supernodes' capacities, we need to select supernodes that are likely to receive a large number of service requests considering its location. Since the density of players in each area tends to be stable [39], a supernode that supports a large number of players previously has a high probability to attract a large number of players in the future. We leverage this intuition and select supernodes based on the number of players they support in the previous time slot. We use N_i to denote the number of players supported by supernode sn_i in the previous time slot. We then rank all supernode candidates by N_i in descending order. Finally, we create a supernode preference vector $V =< sn^1, sn^2, ..., sn^{N_s} >$. We select a supernode with rank j with probability P_j calculated by:

$$P_j = \frac{1/j}{\sum_{n=1}^{N_s} 1/n}, \quad (16)$$

where N_p is the number of supernodes. Finally, the predeployed supernodes have sufficient capacities to handle the request surge in their areas.

3.6 Discussion on Security Issues of CloudFog

As supernodes can be contributed by players and organizations, attackers can control supernodes to reach their malicious goals such as gaining undeserved rewards and destroying normal functionality of the gaming system. For example, some supernodes may generate a large amount of junk files and send them to players so as to earn rewards from the game service provider; some supernodes can intercept or wiretap users' personal information; some supernodes may deliberately delay the transmission of game videos in order to destroy user satisfactions. These issues are critical but beyond the scope of this paper, and we will study them in our future work.

4 Performance Evaluation

4.1 Experimental Settings

We conducted experiments on the PeerSim [40] simulator and the PlanetLab [41] real-world testbed to evaluate the performance of CloudFog in comparison with other systems. We measured the performance in response latency, playback continuity and user coverage. *Basic CloudFog (CloudFog/B)* denotes the fog-assisted cloud gaming infrastructure without applying our proposed strategies; *Advanced CloudFog (CloudFog/A)* denotes our system with all proposed strategies. We compared *CloudFog* with the current cloud gaming model [16] (denoted by *Cloud*) and *CDN* [21]. In *CDN*, a number of powerful CDN nodes are deployed to increase user coverage, which take over all the cloud's tasks (including storing and computing game status and rendering new game videos). The default number of main datacenters is 5 and 2 for all systems in simulation and PlanetLab, respectively. The number of servers within each datacenter is 5. As shown in Figure 16(b), the cost of renting a cloud server is twice as much as deploying a supernode, so the number of servers in *CDN* is set to $1/2$ of the number of supernodes in *CloudFog*. We also conducted additional experiments for CDN with 45 and 8 randomly distributed servers in simulation and PlanetLab (denoted by *CDN-45* and *CDN-8*). Other default settings are: $\theta = 0.5$, $\lambda = 1$, $h_1 = 100$ and $h_2 = 10$. We used the statistics in [42], [43] for the distribution of download bandwidth in the simulation. To simulate real-world internet connections, a node's upload bandwidth capacity was set to $1/3$ of its download bandwidth [44], [45]. In order to simulate a system with supernodes of various capacities, the capacity of supernodes (i.e., maximum number of normal nodes that can support) in the system follows a Pareto distribution [46], [47] with parameter $\alpha = 2$.

The experiment is divided into 28 cycles with each cycle representing one day's gaming activities; each cycle is further divided into 24 one-hour subcycles. According to [36], [37], we assume that 8pm-12am (i.e., subcycle 20 to 24) are peak hours when large number of online players are playing games. According to studies in [48], we randomly selected 50% nodes and 30% nodes to play for a period randomly selected from (0, 2] and (2, 5] hours a day, and let the remaining 20% nodes to play for a period randomly selected from (5, 24] hours a day. Inside one cycle, the game start time of each player is randomly selected from subcycle [1,19] with a probability of 30%, and randomly selected from subcycle [20,24] with a probability of 70%. To simulate supernodes' willingness in providing satisfactory streaming service, we randomly chose $1/5$ and $1/10$ supernodes that set their upload bandwidth at 80% and 50% of their capacities, respectively, with 50% probability in each cycle. After each experiment cycle, each player rates the supernode using the value of its game video playback continuity during this gaming activity. As defined in Figures 8(a) and 8(b), continuity is measured by the proportion of packets arrived within the required response latency over all packets in a game video. We use the first 21 cycles (i.e., 3 weeks divided into 126 time windows) as a warmup

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

period to accumulate reputation scores for all supernodes. We record the number of online players for each subcycle and used this data to predict the number of online players. We then record the experimental results of the last 7 cycles and report the average value of these cycles.

Simulation settings. In the simulation, there were 10,000 game players (including online and offline players), 10% of which have the capacity to be supernodes. We randomly selected 600 supernodes for *CloudFog*. This is reasonable as the hardware requirement of servers in *CDN* is much more demanded than that of supernodes in *CloudFog*, thus, given the same amount of revenue, *CloudFog* can deploy more supernodes than the number of servers. The number of friends for each player follows power-law distribution with skew factor of 0.5 [49]. In order to simulate the dynamics of supernodes and players, the players join the system following the Poison distribution with an average rate of 5 players per second [50]. Each node leaves the system after it finishes playing and joins the system for the next experiment cycle. As in [51]–[53], the capacities of nodes follow Pareto distribution with a mean of 5 and shape parameter $\alpha = 1$.

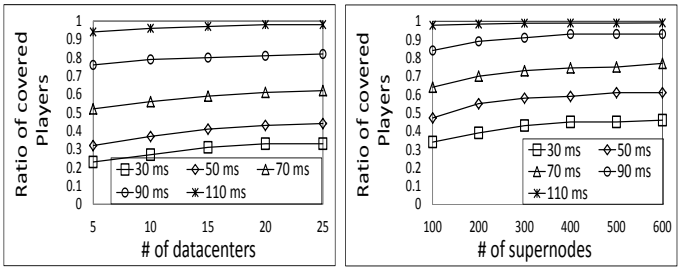
We defined 5 games, their quality levels and latency requirements are shown in Table 2. When a player joins the system, if none of its friends is playing, it randomly chooses a game to play; otherwise, it chooses the game that has the largest number of its friends playing. OnLive provides gaming service at a frame rate of 30fps [4]. Thus, the frame rate of game videos in our experiment is set to 30fps. The communication latency between each pair of nodes was randomly selected from the ping latency traces from the League of Legends [54] based on each latency's occurrence frequency.

PlanetLab experiment settings. We used 750 distributed nodes nationwide, and 300 of them have the capacity to be supernodes. The nodes with IP 128.112.139.43 in Princeton University and IP 131.179.150.72 in the University of California, Los Angeles were set as cloud datacenters, due to their stable connection during the experiment. All other settings are the same as in the simulation.

4.2 Experimental Results for Overall Performance

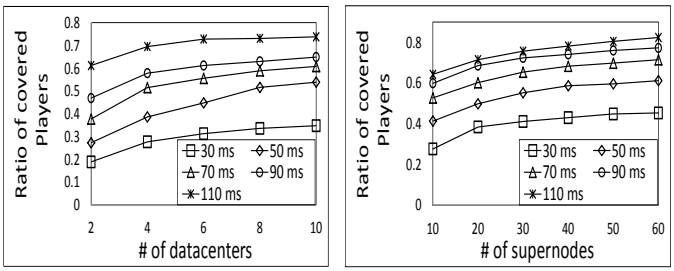
We first tested the effectiveness of building datacenters in increasing user coverage in *CloudFog/B*. Recall that the general response latency requirement is 100ms [6]; 20ms is attributed to playout and processing delay and 80ms is the *network latency*. A user is covered by a datacenter or a supernode if the response latency is no more than the latency requirement of the user's game, and we measured the ratio of covered players as the number of players covered by a datacenter or a supernode over all players in the system.

Figure 4(a) and Figure 5(a) show the ratio of covered players with different number of deployed datacenters and different network latency requirements of games on Peersim and PlanetLab, respectively. The figures illustrate that more datacenters lead to increased user coverage, as users are more likely to connect to close datacenters. Also, given a certain number of datacenters, stricter latency requirement leads to a smaller user coverage. In order to guarantee a better coverage of the user population, previous research suggested deploying more datacenters nationwide [21]. If OnLive chooses to build its own datacenters and building



(a) User coverage VS # of datacenters. (b) User coverage VS # of super nodes.

Fig. 4: Impact of # of datacenters and supernodes on PeerSim.



(a) User coverage VS # of datacenters. (b) User coverage VS # of supernodes.

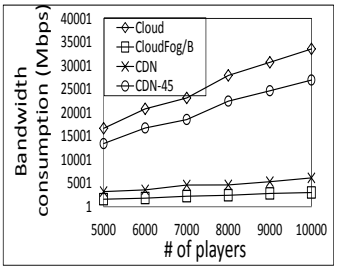
Fig. 5: Impact of # of datacenters and supernodes on PlanetLab.

a medium size datacenter of approximately 300,000 gross square feet costs around 400 million dollars [55], [56], it would cost OnLive around 8 billion dollars to build 20 more datacenters; however, 25 datacenters can only cover 60% players with the general response latency requirement. Thus, increasing user coverage by deploying more datacenters is cost-prohibitive for game service providers. bandwidth costs represent In *CloudFog*, a game service provider can offer a small amount of monetary rewards as incentives to encourage supernodes, and user coverage can be increased by deploying supernodes.

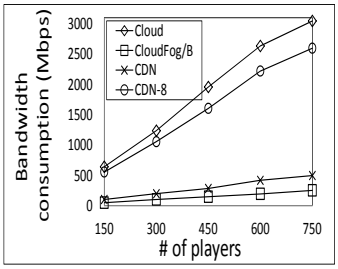
We then examined the effectiveness of supernodes in increasing user coverage in *CloudFog/B* using 5 datacenters on PeerSim and 2 datacenters on PlanetLab. We see from Figure 4(a) that when the network latency requirement is 90ms, deploying 10 datacenters can increase about 10% user coverage than deploying 5 datacenters in PeerSim. Figure 5(a) reflects a similar trend as that in Figure 4(a), the two figures show that the effectiveness of increasing user coverage by deploying more datacenters weakens when the number of datacenters reaches a specific value. Figure 4(b) and Figure 5(b) show the ratio of covered players with different number of randomly selected supernodes and network latency requirements, Figure 4(b) shows that 100 supernodes can increase user coverage from 0.25 to 0.65 when the network latency requirement ranges from 110ms to 30ms. 200 supernodes can help achieve user coverage of deploying 25 datacenters. Figure 4(b) and Figure 5(b) show that instead of building datacenters, deploying supernodes is an effective alternative in increasing user coverage.

As players do not need to pay for bandwidth usage when they subscribe for internet services, we measure bandwidth consumption of different gaming systems from the side of cloud servers. Figures 6(a) and 6(b) show the bandwidth consumption of the cloud versus the number

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems

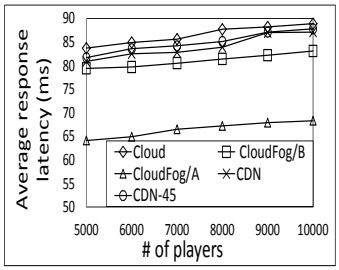


(a) The PeerSim simulator.

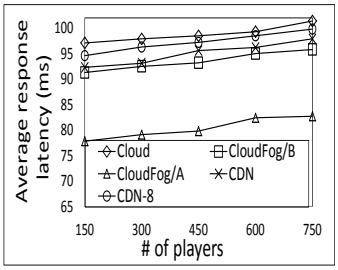


(b) The PlanetLab real-world testbed.

Fig. 6: Server bandwidth consumption.

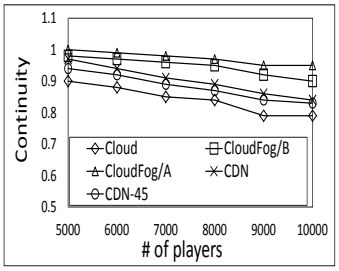


(a) The PeerSim simulator.

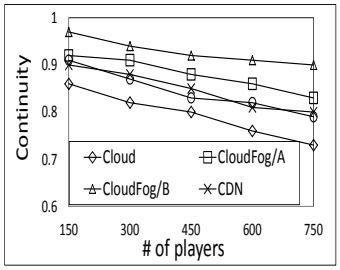


(b) The PlanetLab real-world testbed.

Fig. 7: Response latency.

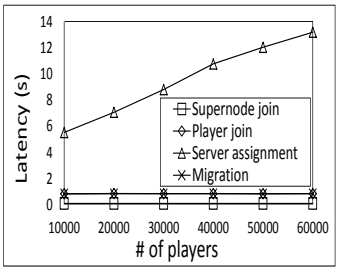


(a) The PeerSim simulator.

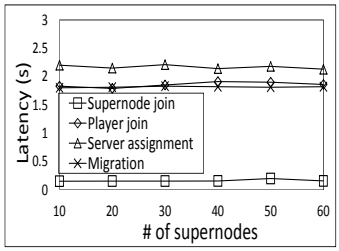


(b) The PlanetLab real-world testbed.

Fig. 8: Playback continuity.



(a) The PeerSim simulator.



(b) The PlanetLab real-world testbed.

Fig. 9: System setup latency and player join latency.

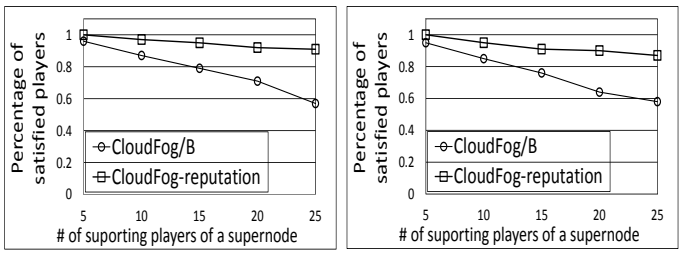
of players in the system. As *CloudFog/A* does not influent the bandwidth consumption of *CloudFog*, thus we use *CloudFog/B* to represent the bandwidth consumption of both *CloudFog/A* and *CloudFog/B*. We see that the result follows *Cloud*>*CDN*>*CDN-45*/*CDN-8*>*CloudFog/B*. The bandwidth consumption of *CDN* does not include those of additional servers. If we include them, *CDN*'s bandwidth consumption is similar to that of *Cloud*'s. *CDN* generates less bandwidth consumption than *CDN-45* and *CDN-8* as more servers are deployed to stream game videos to the players. *CloudFog/B* saves significant bandwidth consumption cost due to its employment of supernodes to stream game videos to the players. The cloud only needs to send update information rather than the entire game video to the supernodes.

Figures 7(a) and Figure 7(b) show the average response latency per player in different systems in PeerSim and PlanetLab, respectively. We see that *CDN-45* and *CDN-8* generate slight shorter response latency than *Cloud* due to the use of scattered servers, and users are more likely to connect to servers within a short distance. *CDN* further reduces the response latency as more servers are deployed. However, the improvement is not significant because the servers need to cooperate with each other to compute new game status, which lead to relatively long latency. *CloudFog/B* shows a slight reduction in response latency than that of *CDN*, which indicates the effectiveness of our fog-assisted infrastructure in reducing the latency. In *CloudFog*, users are supported by supernodes that are physically close to them. As the game video is streamed from supernodes to the users, instead of from servers that are physically far away. Thus, *CloudFog* is able to reduce the response latency for users. This result shows that our system not only reduces the response latency of the system of deploying many datacenters but also saves the prohibitive cost of building more datacenters. *CloudFog/A* further reduces the latency, which indicates the effectiveness of our proposed strategies in reducing response latency.

Video playback continuity is an important metric for QoS. We measured continuity by the proportion of packets arrived within the required response latency over all packets in a game video. Figures 8(a) and 8(b) show the average playback continuity of game videos when different number of players are playing games concurrently, which is a metric to measure weather a player can enjoy smooth video playback. We see that *Cloud* yields the lowest playback continuity because there are only a small number of cloud servers, which may locate far away from some players. So most game videos need to be transmitted from remote servers to clients, thus large portion of packets cannot be received within the required response latency. *CDN-45* and *CDN-8* produce higher continuity than *Cloud* because players are supported by their nearby servers. *CDN* increases the playback continuity of *CDN-45* and *CDN-8* as players are more likely to find nearby servers when more servers are deployed. *CDN* generates smaller continuity than *CloudFog/B* and *CloudFog/A*, because not all users in *CDN* are able to connect to a nearby server due to the shortage of servers. So game video packets need to travel longer distance than that in *CloudFog*. *CloudFog/B* increases the continuity of *CDN* due to the effectiveness of the fog-assisted infrastructure, a large portion of users are supported by supernodes that are close to them. *CloudFog/A* provides an average of more than 90% continuity, with the contribution of all other proposed strategies.

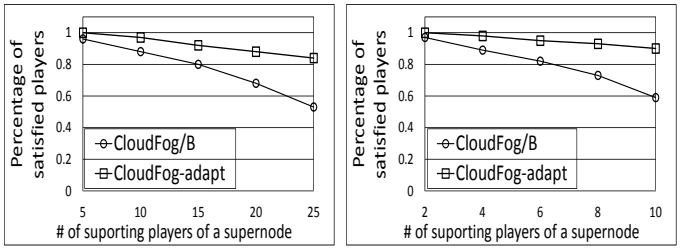
We further tested: 1) server assignment latency, which is the time needed to allocate all players to cloud servers based on the social network based server assignment strategy; 2) average supernode join latency, which is average time from the time a supernode joins *CloudFog* until the time when it is connected to the cloud; 3) average player join latency, which is the average time from the time a player joins *CloudFog* until the time that it is connected to a supported supernode; 4) average migration latency, which is the average time needed for a player to connect to a new supernode when

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

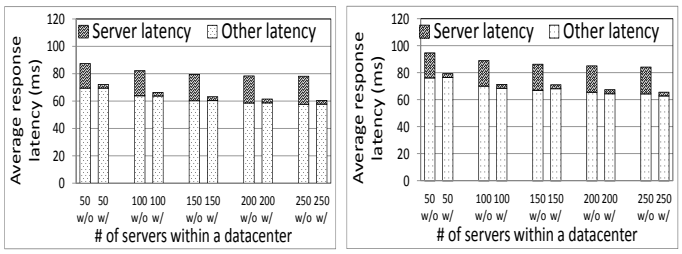
Fig. 10: Effectiveness of reputation based supernode selection.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 11: Effectiveness of encoding rate adaptation.

its supported supernode fails. When a player's supported supernode is out of service, the player needs to connect to a new supernode. We call the process of connecting to a new supernode a migration. In this experiment, we randomly chose 100 supernodes on PeerSim and 10 supernodes on PlanetLab, we then simulated supernode failures by disconnecting all players from these supernodes. In order to test the scalability of *CloudFog* on PeerSim, we varied the numbers of players from 10,000 to 60,000 and set the numbers of supernodes to 6/100 of players. Figure 9(a) shows the latency results on PeerSim. We see that when the numbers of players increase, the server assignment latency rises because the cloud needs to assign more players to servers, however, the server assignment latency does not increase rapidly. As the server assignment operation is conducted periodically (e.g., weekly), so the assignment latency does not compromise the QoS of *CloudFog*. The average supernode join latency remains low because supernodes only need to connect to the cloud; the the average player join latency remains constant since each player only needs to select a supernode from a small number of candidates. We also see that the migration latency is around 0.08 second, which is low. Because the game status is calculated on the cloud and supernodes do not need to store players' gaming information, there is no information transfer from the disconnected supernode to the new supernode. Thus, the migration overhead is small. During the migration, a player does not need to restart the game, and the game will resume after around 0.08 second. Figure 9(b) shows the latency performance when different numbers of supernodes are deployed on PlanetLab. We see that server assignment latency keeps stable as it is not affected by the number of supernodes. We also see that supernode and player join latency and migration latency stay low due to the same reason as in Figure 9(a). Figure 9(a) and Figure 9(b) indicate that the setup and dynamical reconfiguration of *CloudFog* can be completed within a short time.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 12: Effectiveness of social network based server assignment.

4.3 Experimental Results for Proposed Strategies

In the following, we show the effectiveness of each of our proposed strategies: i) reputation based supernode selection, ii) encoding rate adaptation, iii) social network based server assignment, and iv) dynamic supernode provisioning.

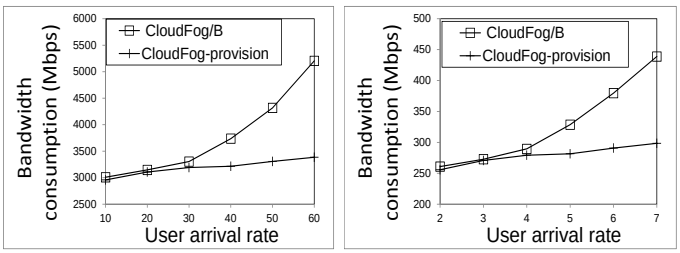
4.3.1 Performance of Reputation Based Supernode Selection Strategy

QoS is determined by packet loss rate and response delay. Thus, if a user can receive 95% of its game packets within the game's response latency, we consider this user as a satisfied player, and this definition is adopted in all figures within the paper. Figures 10(a) and 10(b) show the percentage of satisfied players with and without the reputation based supernode selection strategy, denoted by *CloudFog-reputation* and *CloudFog/B*, respectively. In *CloudFog/B*, among the final selected supernode candidates (introduced in Section 3.2), a player randomly selects a supernode from this set. We see that *CloudFog-reputation* significantly increases the percentage of satisfied players due to the reason that players are prone to select supernodes that can provide high QoS in streaming game videos. In *CloudFog-reputation*, each player evaluates supernodes' quality of service from previous interactions and selects the supernode that provides high QoS with high probability. Thus, the selected supernode is likely to support the player with high QoS in game video streaming. On the other hand, *CloudFog/B* randomly assigns supernodes to players. Though the assigned supernode is within the player's transmission delay threshold, the supernode may not be willing to provide all connected players with satisfactory streaming services.

4.3.2 Performance of Encoding Rate Adaptation Strategy

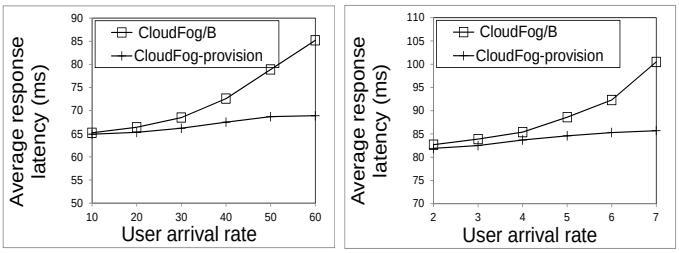
Figure 11(a) and Figure 11(b) show the percentage of satisfied players with and without (denoted by *CloudFog-adapt* and *CloudFog/B*) the encoding rate adaptation strategy, in PeerSim and PlanetLab, respectively. We see that *CloudFog-adapt* increases the percentage of satisfied users in *CloudFog/B*. The increase rate reaches 27% when the number of supported players of a supernode is 25 in the simulation. When the network condition is not good enough to support high quality streaming of game videos, this strategy decreases the video quality level to meet the response latency based on loss rate tolerance, thus increasing the number of satisfied players.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 13: Effectiveness of the dynamic supernode provisioning strategy in reducing cloud bandwidth consumption.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

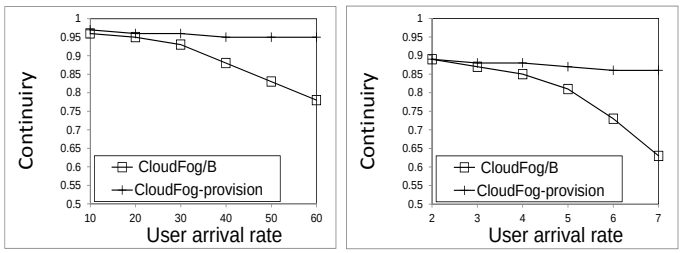
Fig. 14: Effectiveness of the dynamic supernode provisioning strategy in reducing response delay.

4.3.3 Performance of Social Network Based Server Assignment Strategy

Figures 12(a) and 12(b) show the average response latency with (*w/*) and without (*w/o*) the social network based server assignment strategy on PeerSim and PlanetLab, respectively. In *w/o*, the users are randomly assigned to servers in a datacenter. We decompose the response latency to *server latency* (the communication latency among servers) and *other latency*. We see that *w/* produces about 20ms reduction in server latency, which leads to the reduction of overall response latency. This is because with this strategy, users that interact with each other in a game are more likely to be assigned to the same server within a datacenter, thus their interaction is less likely to involve communication among servers.

4.3.4 Performance of Dynamic Supernode Provisioning Strategy

In order to test the performance of *CloudFog* under user churns, we manually set different player arrival rates for peak hours and off-peak hours. In PeerSim simulation, we set the average player arrival rates during off-peak hours (subcycles 1-19) at 5 players/minute, and varied the average user arrival rate during peak hours (subcycles 20-24) from 10 to 60 players/minute with 10 players/minute increase in each step. In PlanetLab experiment, we set the average player arrival rates during off-peak hours at 1 players/minute, and varied the average user arrival rate during peak hours from 2 to 7 players/minute with 1 player/minute increase in each step. In *CloudFog/B*, the game service provider reserves a constant amount of supernodes, i.e., 400 supernodes in PeerSim simulation and 40 in PlanetLab experiments, while in *CloudFog-provision*, we dynamically set the number of supernodes according to the method in Section 3.5. The game service provider predicts the number of online players every 4 hours (subcycles) and reserves supernodes based on the prediction. Figures 13(a) and 13(b) show the cloud bandwidth consumption with and without the proposed dynamic supernode provisioning strategy. We see that as user arrival rate increases in *CloudFog/B*, cloud bandwidth consumption drastically rises in both PeerSim and PlanetLab experiments. This is due to the reason that *CloudFog/B* reserves a fixed number of supernodes regardless of the online player population. Then, when a large number of players are crowded into the system, most players cannot find support from supernodes and need to resort to the cloud for game video streaming. *CloudFog-provision* greatly reduces the cloud bandwidth consumption because it forecasts the potential rise in player population and reserve a sufficient number of supernodes in advance.



(a) The PeerSim simulator. (b) The PlanetLab real-world testbed.

Fig. 15: Effectiveness of the dynamic supernode provisioning strategy in increasing continuity.

Figures 14(a) and 14(b) show the average response latency for *CloudFog/B* and *CloudFog-provision* in PeerSim and PlanetLab experiments, respectively. We see that *CloudFog-provision* can reduce the average response latency due to the reason that it reserves a sufficient number of supernodes in advance. When there are a large number of concurrent online players, these players can find support from supernodes that are physically close to them, so the response latency is reduced compared to downloading game videos from the cloud. While in *CloudFog/B*, a large portion of players rely on the cloud for game videos due to lack of supernodes, which generates long response latency as the cloud is physically far from the players.

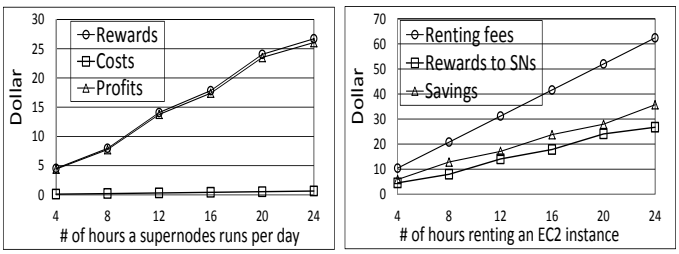
Figures 15(a) and 15(b) show the average continuity for *CloudFog/B* and *CloudFog-provision* in PeerSim and PlanetLab experiments, respectively. We see that when user arrival rate increases, *CloudFog/B* leads to deteriorated average continuity for players. This is because when there are insufficient supernodes for an excessive number of concurrent players, the cloud needs to stream game videos to most players. As the video packets need to travel a long distance to the players, game interruption occurs when the packets cannot arrive within the game's required response latency. *CloudFog-provision* manages to sustain a high average continuity due to the same reason as in Figures 14(a) and 14(b).

These results verify that *CloudFog* is resilient to user churns. That is, when user arrival rate increases, the performance of *CloudFog* in providing high QoS in gaming activities will not be degraded.

4.4 Analysis of Incentives for Supernodes and Savings for Game Service Providers

As supernodes play important roles in *CloudFog*, we also evaluate the incentives for supernodes and the costs of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2563428, IEEE Transactions on Parallel and Distributed Systems



(a) Rewards, costs and profits for supernodes.

(b) Renting fees and savings for a game service provider.

Fig. 16: Economical incentives for supernodes and game service providers.

deploying supernodes for game service providers. We select a random supernode and depict the profits earned by its owner. Assume that a supernode is a typical server that uses approximately 0.25kW electric power [57], and it is located in a region where the electricity cost is 10.8 cents/kWh, which is the US average price of electricity [58]. The hourly electricity cost of running the server is then $0.25 \times 0.108\ cents = 0.027$ dollar. We also assume that the game service provider pays 1 dollar for 1GB bandwidth a supernode contributes. Figure 16(a) shows the monetary rewards the supernode's owner earns from the game service provider, the costs of running the supernode and the owner's profits (calculate by Equation (1)) when the supernode runs for different number of hours. We see that the costs are trivial comparing to the rewards, so players and organizations are motivated to contribute their machines to earn profits.

For a game service provider, if it deploys 300 supernodes and all supernodes run 24 hours a day for a full year, it needs to spend about 2.9 million dollars on rewarding the supernodes each year. While building a medium size datacenter costs around 400 million dollars, deploying supernodes rather than building extra datacenters is a more economical strategy for game service providers, and previous experimental results already show that supernodes are effective in providing high quality of service to users. Instead of building data centers, game service providers can rent instance resources from existing cloud providers. Assuming a game service provider rents a "g2.8xlarge" GPU instance from Amazon EC2 with 2.6 dollar per hour [59], we first plot the renting fees (denoted by *Renting fees*) in Figure 16(b). Compared to deploying a supernode with rewards (denoted by *Rewards to SNs*), we then plot the savings (denoted by *Savings*) for the game service provider by subtracting the *Rewards to SNs* from *Renting fees*. From Figure 16(b), we see that *CloudFog* is able to save game service providers' expenses.

5 Conclusions

Cloud gaming is a very promising model for thin-client MMOG since it frees players from this requirement, but it faces formidable challenges that prevents it from achieving high QoS and low cost. We propose *CloudFog*, which leverages supernodes functioning as "fog" to connect the cloud to users. The cloud conducts the intensive computation for producing game state and sends update information to supernodes. The supernodes then generate game videos to stream to players. To select a suitable supernode that can provide satisfactory game video streaming service, we propose a reputation based supernode selection strategy. Considering that different games have different degrees of response latency tolerance and packet loss tolerance, we propose a receiver-driven encoding rate adaption strategy to balance these two factors in guaranteeing QoS. Since social friends in online games tend to play game together, we assign these players to the same server in a datacenter to reduce the interactions of servers to further reduce the latency. We also propose a dynamic supernode provisioning strategy to deal with user churns and relieve server loads. As a result, CloudFog reduces response latency and bandwidth consumption and increases user coverage. These advantages are verified by our experiments on the PeerSim simulator and the PlanetLab real-world testbed. In our future work, we will study the security issues such as dealing with malicious supernodes and preventing cheating behaviors in *CloudFog*; we will study how to evaluate the user Quality of Experience (QoE) when using the CloudFog system; we will also study deploying cloud servers as supernodes and determining the optimal number of cloud servers so that players can perceive the best QoE.

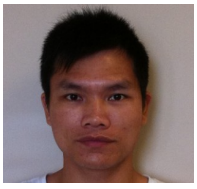
Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of ICPP 2015 [60].

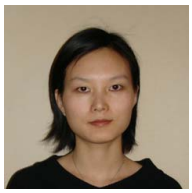
References

[1] C. Bezerra and C. Geyer. A load balancing scheme for massively multiplayer online games. *Multimedia Tools Appl*, 2009.
[2] N. Bilton. Video Game Industry Continues Major Growth, Gartner Says. *The New York Times*, 2011.
[3] CBS News. Study: Number of smartphone users tops 1 billion. *http://www.cbsnews.com/8301-205_162-57534583/study-number-of-smartphone-users-tops-1-billion/*, 2012.
[4] Onlive. Inc. http://www.onlive.com/, [Accessed in Nov 2014].
[5] Gaikai. Inc. http://www.gaikai.com/, [accessed in nov 2014].
[6] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hobfeld. An Evaluation of QoE in Cloud Gaming Based on Subjective Tests. In *Proc. of IMIS*, 2011.
[7] S. Choy, B. Wong, G. Simon, and C. Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proc. of NetGames*, 2012.
[8] K. Chen, P. Huang, and C. Lei. Game Traffic Analysis: An MMORPG Perspective. *Computer Networks*, 50(16):3002–3023, 2006.
[9] E. Carlini, M. Coppola, and L. Ricci. Integration of P2P and Clouds to support Massively Multiuser Virtual Environments. In *Proc. of NetGames*, 2010.
[10] N. Bila, E. de Lara, K. Joshi, A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: efficient idle desktop consolidation with partial vm migration. In *Proc. of EuroSys*, 2012.
[11] S. Nedevschi, J. Chandrashekar, J. Liu, B. Nordman, S. Ratnasamy, and N. Taf. Skilled in the art of being idle: Reducing energy waste in networked systems. In *Proc. of NSDI*, 2009.
[12] J. Blackburn, R. Simha, N. Kourtellis, X. Zuo, M. Ripeanu, J. Skvoretz, and A. Iamnitchi. Branded with a scarlet "C": cheaters in a gaming social network. In *Proc. of WWW*, 2012.
[13] C. Bezerra, J. Comba, and C. Geyer. Adaptive load-balancing for MMOG servers using KD-trees. *CIE*, 10(3):5, 2012.
[14] S. Ahmad, C. Bouras, E. Buyukkaya, R. Hamzaoui, A. Papazois, A. Shani, G. Simon, and F. Zhou. Peer-to-peer live streaming for Massively Multiplayer Online Games. In *Proc. of P2P*, 2012.

[15] J. Chen, M. Arumaithurai, X. Fu, and K. Ramakrishnan. Gaming over COPS: A Content Centric Communication Infrastructure for Gaming Applications. In *Proc. of ICDCS*, 2012.
[16] C. Huang, C. Hsu, Y. Chang, and K. Chen. GamingAnywhere: An Open Cloud Gaming System. In *Proc. of MMSys*, 2013.
[17] Z. Zhao, K. Hwang, and J. Villeta. Game cloud design with virtualized CPU/GPU servers and initial performance results. In *Proc. of ScienceCloud*, 2012.
[18] S. Wang and S. Dey. Cloud mobile gaming: modeling and measuring user experience in mobile wireless networks. In *Proc. of SIGMOBILE*, 2012.
[19] M. Hemmati, A. Javadtalab, A. Shirehjini, S. Shimohammadi, and T. Arici. Game as Video: Bit Rate Reduction through Adaptive Object Encoding. In *Proc. of NOSSDAV*, 2013.
[20] L. Lin, X. Liao, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li. LiveRender: A Cloud Gaming System Based on Compressed Graphics Streaming. In *Proc. of ACM Multimedia*, 2014.
[21] S. Choy, B. Wong, G. Simon, and C. Rosenberg. A hybrid edge-cloud architecture for reducing on-demand gaming latency. *Multimedia Systems*, 20(5):503–519, 2014.
[22] T. Hobfeld, R. Schatz, M. Varela, and C. Timmerer. Challenges of QoE management for cloud applications. *IEEE Communications Magazine*, 50(4):28–36, 2012.
[23] Y. Lee, K. Chen, H. Su, and C. Lei. Are all games equally cloud-gaming-friendly? An electromyographic approach. In *Proc. of NetGames*, 2012.
[24] M. Claypool. Motion and scene complexity for streaming video games. In *Proc. of FDG*, 2009.
[25] A. Ojala and P. Tyrvainen. Developing Cloud Business Models: A Case Study on Cloud Gaming. *IEEE Software*, 28(4):42–47, 2011.
[26] How to batch render in Sony Vegas. http://sony-vegas.wonderhowto.com/how-to/batch-render-sony-vegas-329667/, [Accessed in Dec, 2015].
[27] H. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna. Video flashlights: real time rendering of multiple videos for immersive model visualization. In *ACM International Conference Proceeding Series*, volume 28, pages 157–168, 2002.
[28] B. Van De Bovenkamp, S. Shen, A. Iosup, and F. Kuipers. Understanding and recommending play relationships in online social gaming. In *Proc. of COMSNETS*, 2013.
[29] P. Salvador and A. Nogueira. Study on geographical distribution and availability of bittorrent peers sharing video files. In *Proc. of ISCE*, 2008.
[30] H. Shen and G. Liu. A lightweight and cooperative multi-factor considered file replication method in structured P2P systems. *TC*, 2012.
[31] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hobfeld. Gaming in the clouds: QoE and the users' perspective. *Mathematical and Computer Modelling*, 2011.
[32] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.
[33] K. Hoffman, D. Zage, and C. Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *CSUR*, 42(1):1, 2009.
[34] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, 2004.
[35] J. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: Scaling online social networks. In *Proc. of SIGCOMM*, 2010.
[36] A. Patro, S. Rayanchu, M. Griepentrog, Y. Ma, and S. Banerjee. The anatomy of a large mobile massively multiplayer online game. *SIGCOMM Computer Communication Review*, 42(4):479–484, 2012.
[37] D. Pittman and C. GauthierDickey. Characterizing virtual populations in massively multiplayer online role-playing games. In *Advances in Multimedia Modeling*, pages 87–97. 2010.
[38] M. Kendall and J. Ord. *Time-series*, volume 296. Edward Arnold London, 1990.
[39] X. Zhuang, A. Bharambe, J. Pang, and S. Seshan. Player dynamics in massively multiplayer online games. *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-CS-07-158*, 2007.
[40] The PeerSim simulator. http://peersim.sf.net, [Accessed in Nov 2014].
[41] PlanetLab. http://www.planet-lab.org/, [Accessed in Nov 2014].
[42] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *Proc. of SIGCOMM*, 2007.
[43] Xu Cheng and Jiangchuan Liu. Nettube: Exploring social networks for peer-to-peer short video sharing. In *Proc. of INFOCOM*, 2009.
[44] The difference between upload and download speed for broadband DSL. http://fetchsoftworks.com/fetch/help/Contents/Tutorial/SlowUploads.html, [Accessed in Dec, 2015].
[45] H. Shen, Y. Lin, and J. Li. A social-network-aided efficient peer-to-peer live streaming system. *TON*, 23(3):987–1000, 2015.
[46] K. Psounis, P. M. Fernandez, B. Prabhakar, and F. Papadopoulos. Systems with multiple servers under heavy-tailed workloads. *Performance Evaluation*, 2005.
[47] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Trans. Network*, 1997.
[48] C. Hellstrom, K. Nilsson, J. Leppert, and C. Aslund. Influences of motives to play and time spent gaming on the negative consequences of adolescent online computer gaming. *Computers in Human Behavior*, 28(4):1379–1387, 2012.
[49] S. Raza A. Nazir and C. Chuah. Unveiling dacebook: a measurement study of social network based applications. In *Proc. of SIGCOMM*, 2008.
[50] D. Wu, Y. Liu, and K. W. Ross. Modeling and Analysis of Multichannel P2P Live Video Systems. *TON*, 18(4):1248–1260, 2010.
[51] H. Shen and C. Xu. Locality-aware and churn-resilient load balancing algorithms in structured peer-to-peer networks. *TPDS*, 18(6):849–862, 2007.
[52] N. Bansal and M. Harchol-Balter. Analysis of srpt scheduling: investigating unfairness. In *Proc. of SIGMETRICS/Performance*, 2001.
[53] R. Subrata and A. Y. Zomaya. Game-theoretic approach for load balancing in computational grids. *TPDS*, 19(1):66–76, 2008.
[54] Latency (lag) vs win rate in League of Legends. https://www.reddit.com/r/dataisbeautiful/comments/1t23a0/latency_lag_vs_win_rate_in_league_of_legends_oc/, [Accessed in Mar, 2016].
[55] I. Goiri, J. Guitart, and J. Torres. Economic model of a Cloud provider operating in a federated Cloud. *Information Systems Frontiers*, 14(4):827–843, 2012.
[56] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. 2009.
[57] P. Gao, A. Curtis, B. Wong, and S. Keshav. It's not easy being green. *ACM SIGCOMM Computer Communication Review*, 42(4):211–222, 2012.
[58] U.S. Energy Information Administration. http://www.eia.gov, [accessed in nov 2014].
[59] Amazon EC2 Pricing. https://aws.amazon.com/ec2/pricing/, [Accessed in Dec, 2015].
[60] Y. Lin and H. Shen. Leveraging fog to extend cloud gaming for thin-client mmog with high quality of experience. In *Proc. of ICPP*, 2005.



Yuhua Lin Yuhua Lin received both his BS degree in Software Engineering and MS degree in Computer science from Sun Yat-sen University, China in 2009 and 2012 respectively. He is currently a Ph.D student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include social networks and reputation systems.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Electrical and Computer Engineering at Clemson University. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.