Timely cloud gaming

4 authors, including:

Mehrnaz Tavan

Rutgers, The State University of New Jersey

10 PUBLICATIONS   **49** CITATIONS

Timely Cloud Gaming

Roy D. Yates
WINLAB, ECE Dept.
Rutgers University
ryates@winlab.rutgers.edu

Mehrnaz Tavan
WINLAB, ECE Dept.
Rutgers University
mt579@winlab.rutgers.edu

Yi Hu
Vencore Labs
Applied Communication Sciences
amyhuyi@gmail.com

Dipankar Raychaudhuri
WINLAB, ECE Dept.
Rutgers University
ray@winlab.rutgers.edu

Abstract—This work introduces a new model for cloud gaming systems aimed at optimizing the timeliness of video frames based on an age of information (AoI) metric. Mobile clients submit actions through an access network to a game server. The game server generates video frames at a constant frame rate. At the mobile device, the display of these frames represent game status updates. We develop a Markov model to characterize the frame delivery process in low-latency edge cloud gaming systems. Based on this model, we derive a simple formula for the average status age of a tightly synchronized low-latency mobile gaming system in which the inter-frame period is a significant contributor to the system latency. We validate the model by ns-3 simulation of a low-latency edge cloud gaming system. Our evaluation scenarios included single-player games as well as multi-player games in which the game processing was conducted by a combination of a centralized game server and edge cloud renderers.

I. INTRODUCTION

With recent improvements in the capabilities of cloud infrastructure and quality of wireless networks, mobile cloud computing systems have become an attractive choice for hosting computationally complex services. Motivated by the increasing popularity of online games [1], researchers have focused on design and implementation of cloud-assisted architectures for latency-sensitive online games [2]–[5]. In contrast to traditional platforms such as game consoles and desktop computers where most of the computations are conducted in the client device, in cloud-assisted designs, the majority of the complex tasks including game status updating, graphic rendering, and video encoding are run in cloud servers.

Cloud gaming has numerous advantages for both players and game developers [2], [3] including: a) *Cost reduction*: Migrating the game status computation and graphic rendering to a remote server enables players to use thin clients. b) *Platform independence*: A greater variety of games can be played on the same device. c) *Piracy prevention*: The game source code is stored only in the game server. d) *Resource enhancement*: Servers have greater processing power and memory as compared to mobile devices.

However, designing a cloud gaming system that provides a high quality of experience (QoE) for players involves some difficult challenges. There are two important factors affecting QoE [6]. First, since game updates are streamed back to the thin client in the form of video frames, the design must maintain user-perceived video quality; neither frozen frames nor frame rate changes are desirable. Second, the interactive and real-time nature of a cloud game, particularly its dependence

on the online generation of game updates, make it more delay sensitive.

An important QoE metric in cloud gaming is the response time which corresponds to the elapsed time between the generation of an action and the displayed result of that action. In fact, the response time reflects the accumulated delay in different components including network and processing delays. In cloud gaming, the sensitivity of users to response-time delay depends on the type of game; a low-latency game such as a first person shooter, which is the focus of this work, requires a response time on the order of 100 ms [3], [7].

Considerable work has been done to evaluate and improve the quality of bandwidth-efficient video streaming; see, for example, [8], [9]. To reduce the effect of network and server congestion, most video streaming systems employ large playback buffers and adaptive bitrate techniques. In low-latency gaming, however, a large video buffer is a non-starter. For a 30 fps system, three buffered frames contribute 100 ms delay. This latency constraint also precludes complex video compression methods [10] that code over multiple correlated frames. When image compression is restricted to individual frames, dynamic adjustment of the frame resolution based on client feedback could still be employed to facilitate smooth display. Practical techniques however use playback buffer backlog variations to signal rate changes [8] and thus are unsuitable for low-latency gaming.

Furthermore, network-layer QoS metrics such as packet throughput and delay do not precisely capture the viewer experience [2], [3]. Packet delay may only loosely correlate with the timely delivery of a frame and it generally fails to describe the likelihood of missing and frozen frames. In this work, we present a quantified representation of the user-perceived QoE of the cloud gaming system based on novel client-side measurements. The users observe the games through a fixed frame rate video. A user's QoE is degraded when there are imperfections in the sequence of displayed frames. In this work, we try to tackle QoE evaluation by direct examination of the missing frame process.

As the main performance objective of a cloud gaming system is the "timely" update of players regarding the game status, we follow the concept of age of information introduced in [11] to characterize the system performance. This approach measures system performance in terms of the average freshness of status updates. In the context of a mobile gaming system [3], a game player uses a mobile terminal to submit

actions through a network to a game server. The game server responds to these actions and the inputs of the artificial intelligence of the game (aka the game AI) to generate responses in the form of video frames for the player. The inputs/actions of the players and the input of the game AI induce changes in the game state. In principle, the game state evolves continuously at a game server. In practice, the game server simulates the game by incorporating all inputs in order to update the game at a certain tick rate. A player observes the game via a full-motion video at a fixed nominal frame rate. Each video frame displayed at the mobile client represents a sample of the game status as provided to the player.

This work examines low-latency (sub 100 ms) gaming as enabled by edge cloud game servers. In this setting, video playback delays of even a handful of frames constitutes an outage. Short-duration frame freezes are also undesirable. Status age measures these frame freezes seen by the player when frames go missing. In this setting, we propose a system model for a low-latency edge-cloud gaming system and a simple analytical approximation. Based on ns-3 simulation of single-player and multi-player scenarios, we show that this analytical model provides an accurate performance characterization. In addition, we show how careful synchronization of the game server and the mobile client display can improve performance by roughly 20 percent.

The system model appears in Section II, followed by the analytic model in Section III. Simulation-based performance comparisons are provided in Section IV. In Section V, we describe related work and we conclude in Section VI.

II. SYSTEM MODEL

As depicted in Figure 1, the gaming system has the following networked components:

- A mobile client that submits user actions to the game server.
- A game server that combines buffered user actions with game AI actions to generate game status updates.
- A frame renderer that translates game server updates to video image frames (i.e., images to be displayed on the screen of the mobile client.)
- A display buffer at the mobile client that displays buffered frames at a fixed frame rate.

The output of the game server is a sequence of instructions for the frame renderer to construct an image frame. These instructions may contain far fewer bytes than the images that they describe. We refer to these instructions as updates since they update the renderer (and ultimately the player) on the status of the game.

In the most general setting, the mobile client, the game server and the frame renderer are each entities connected by networks. In mobile gaming, the connection from the mobile client to the game server will include a wireless access link. In a conventional gaming system, the frame renderer is integrated in a relatively powerful client. In this case, the game server sends status updates across the wireless link to the mobile
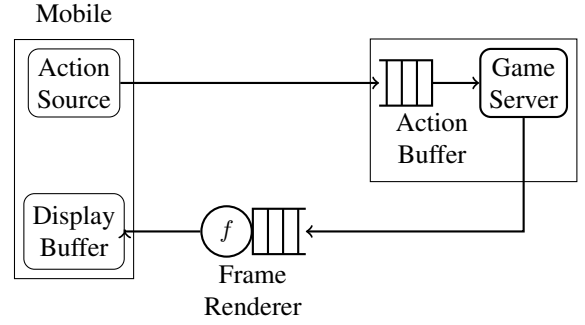

Fig. 1: Network Gaming System

client. In our thin-client mobile setting, we consider two scenarios:

- The frame renderer may be integrated with the game server so that forwarded frames from the game server to the renderer takes negligible time. In this case, image frames are transmitted over the wireless link to the mobile client.
- The frame renderer may be a separate entity in the network. For example, the renderer may be located in an edge cloud server that renders image frames for many mobiles in the same local area [5].

Each of the network entities as well as the network interfaces may degrade the game performance. If the access network is slow, a player's actions will suffer queueing delay in the network. As the game server is likely to be processing game updates for multiple mobiles, possibly in multiple games, there may be queueing of actions at the game engine. Game updates, which may process multiple queued actions in a single frame, can have varying execution times as the task complexity can fluctuate. Randomness in game updates, in combination with randomness in the network can result in queueing of updates at the output of the game server. Randomness in the execution time required to render a frame can also result in both queueing at the input to the renderer and queueing of frames at the output of the frame renderer.

To analyze this complex situation, we propose to decompose the system. The "actions" submitted by the mobile are short instructions, possibly just a few bytes each in length. Furthermore the offered rate of these actions will be low; human responses at a rate of more than 100 actions per second are unlikely [12]. Thus, the traffic generated by the user actions will be small, on the order of 10 kb/s [2], relative to the data rates sustainable on even a moderate-rate access network. Thus in a well designed system, the delays in delivering these actions to the action queue at the game server should be negligible, relative to either the delays in human response or the downlink delay of transmitted frames.

Instead, we focus on the timely rendering of game server updates at the mobile display. Specifically, we assume updates are produced at a fixed frame rate f such that the mobile client displays a frame every $T = 1/f$ seconds. Specifically, for $k = 1, 2, \ldots$, the game server instantiates game status update k

at time kT. That is, update k is the game server's authoritative update of the game state at time kT. We note that the game server may simulate the game evolution at a tick rate that is a multiple of f, but updates are generated only at the frame rate f.

In terms of the game state at the game server, user actions that arrive for processing in the interval $[(k-1)T, kT)$ are said to *occur* in that time interval. That is, latency in the user responses is assumed by the game server to be the latency of the user, rather than latency of the network in delivering those user actions. Under this model, consistency of the game state at the game server is maintained in that all actions that *occur* prior to time kT are incorporated in update k.

The mobile client employs a time lag τ such that it schedules the display of frame k at time $T_k = kT + \tau$. In the interval $(kT, kT + \tau)$,

- the game server incorporates user inputs and game AI to produce update k;
- update k is sent, possibly through a network, to the frame renderer;
- the frame renderer generates the video frame k;
- frame k is sent to the mobile for display at time T_k.

If frame k is not ready to be displayed at time T_k, then the most recently received frame is displayed instead.

The mobile client can optimize the lag τ, subject to the limitations of its access network and frame renderer. We will see that the selection of the lag parameter τ can have a substantial effect on system age. In the context of a low-latency game, every frame matters, and the precise selection of the lag τ represents a tight (though intentionally lagging) synchronization of the game server and the mobile client.

In general, the game server sends game status updates via a network to the frame renderer. It is difficult to analyze this without making strong assumptions regarding both the network and the update delivery protocol. In prior status updating work, the disadvantages of queueing of status updates has been noted in a variety of contexts [11], [13]. Specifically, status age suffers if a service facility is processing old updates when newer updates are in the system. That is, the delivery of a newer update offers a larger reduction in status age and also obviates the need for sending the older update. In short, the objective of the system is to display the most recently generated frame by avoiding the processing and queueing of old updates and frames at the expense of newer updates and frames.

To facilitate this goal, we adopt a form of preemptive stop-and-wait protocol for the forwarding of updates from the game server to the renderer. We describe this as an application-layer protocol, although its logic could be implemented at the transport layer. Specifically, the game server initiates the creation of update k at time kT. Each update may consist of multiple packets and contains the information to create one frame. After processing, the game server pushes update k to an output buffer that holds only the most recently generated update. If update $k-1$ is still in the output buffer at that time, it is preempted (i.e., discarded and replaced) by update k. The game server then attempts to send update k to the renderer. Update k remains in the game server output buffer until either a delivery acknowledgement is received from the renderer or it is preempted by update $k+1$. Note that no queueing occurs at the game server output buffer; only the most recently generated update is held in the buffer.

The logic of this preemptive service is assumed to be employed throughout the system. At the input to the frame renderer, update k is preempted if update $k+1$ arrives before update k is processed. At the network interface queue at the output of the frame renderer, frame k may be preempted by frame $k+1$. Ideally, frame k will be delivered to the mobile client in time for display at time T_k. Note, however, the frame may still be useful to display even if it is received late. Specifically, the late arriving frame k will be displayed at time T_{k+1} if frame $k+1$ fails to be delivered on time. Thus, the system's effort to generate frame k may continue until time T_{k+1}.

III. THE UPDATE AGE: AN ANALYTIC MODEL

To build a tractable analytic model, we assume that the time required for processing and sending update k plus rendering and sending frame k are described by a random variable Y_k that we refer to as the *update service time*. For the purpose of an analytic model, we further assume that the update service times $Y_1, Y_2, \ldots$ are independent and identically distributed (iid) sequence. In our system model, the role of Y_k in the display of frames is codified in these rules:

- At time T_k, the mobile displays its most recently buffered frame.
- If $Y_k \leq \tau$, then frame k will be displayed at time T_k.
- If $\tau < Y_k \leq T + \tau$, then frame k will be received by the mobile at time $kT + Y_k$. It will be displayed at time T_{k+1} unless it has been preempted by frame $k+1$.

The idea behind this model is that until time T_k, the system makes its best effort to deliver and display frame k. However, starting at time $(k+1)T$, the system components give priority to the delivery of update/frame $k+1$. In particular, the renderer may choose to terminate update k, either in rendering or in transmission, if its timely delivery to the mobile frame buffer appears to be unlikely.

This model is idealized in certain ways. First, if update/frame k is in transit in the network, protocol layering may preclude its immediate termination. If frame k is still in transit to the mobile client at time $(k+1)T$, it could ultimately be delivered and displayed at time T_{k+1}. Second, we note that backlogs in the network will generally result in dependencies among the delivery times Y_k. However, because the sending protocol has been designed to preclude the queueing of outdated frames, these dependencies will be chiefly the result of memory induced by network backlogs. In this case, even a high frame rate like 60 fps yields a new frame every 16 ms, which, in the context of modern networks, may be sufficient to decorrelate the network response to successive packets.

Even with this simplified system model, basic tradeoffs between system configurations are not well understood. Where
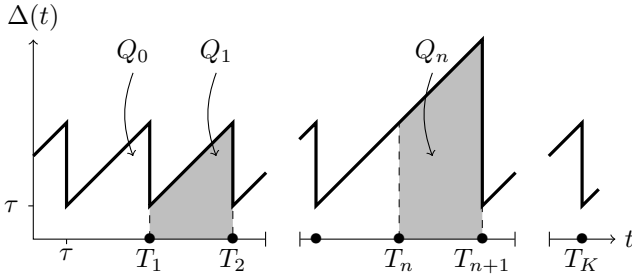
Fig. 2: Example change in status update age $\Delta(t)$ (the upper envelope in bold) at the mobile client display. In this example, frames 1 and 2 are displayed on time, but frame n is lost. At time T_{n+1}, frame $n+1$ is displayed on time, which resets the age to τ.

should the frame renderer be located? What frame rate optimizes the user experience? How should the lag τ be chosen? Here we analyze a status age metric to address these questions.

A. Notation

For random variable X, we denote the probability mass function (PMF) of X by $P_X(x) = \mathrm{P}[X = x]$, the cumulative distribution function (CDF) by $F_X(x) = \mathrm{P}[X \leq x]$ and the complementary CDF by $\overline{F}_X(X) = 1 - F_X(x)$.

B. Status Update Age Analysis

Frame k represents the game status at its generation time kT. We characterize system performance by the status age process $\Delta(t)$. That is, if at time t the current displayed frame is k with timestamp kT, then the status age is $\Delta(t) = t - kT$. In the absence of a new frame being displayed, the status age $\Delta(t)$ grows with time. If frame k with timestamp kT is displayed at time $T_k = kT + \tau$, then $\Delta(T_k) = T_k - kT = \tau$ since frame k represents the game state τ seconds ago.

Thus, the age process $\Delta(t)$ is given by the sawtooth function shown in Figure 2. Specifically, the age grows linearly at unit rate in the absence of a new frame while at time instance T_n, the age drops if a more recent frame is displayed. We now derive the *time-average* status update age $\Delta(t)$ using a graphical argument. Without loss of generality, assume that we begin observing at $t = 0$ when frame $k = 0$ is displayed, so that $\Delta(0) = \tau$.

The time-average age of the status updates is the area under the age graph in Figure 2 normalized by the time interval of observation. For simplicity of exposition, the observation interval is chosen to be $(\tau, KT + \tau)$. Over this K-frame interval, the average age is

$$\Delta^{(K)} = \frac{1}{KT} \int_{\tau}^{KT+\tau} \Delta(t)\, dt. \tag{1}$$

We decompose the area defined by the integral (1) into a sum of disjoint polygonal areas $Q_0, Q_1, \ldots, Q_{K-1}$ (with Q_1 and Q_n highlighted in the figure.) This decomposition yields the average age over K frames

$$\Delta^{(K)} = \frac{1}{KT} \sum_{k=0}^{K-1} Q_k. \tag{2}$$

The area Q_k depends on the delivery of frames to the mobile display. At time T_k, let X_k denote the number of prior frames that have gone missing. That is, if frame k is displayed at time T_k, then no frames are missing and $X_k = 0$. If frames k and $k - 1$ are missing, and frame $k - 2$ is displayed at time T_k, then $X_k = 2$. From Figure 2, we see that each Q_k includes a right triangle of base and height T atop a base rectangle of width T and height τ. In addition, each missing frame at time k contributes a square of area T^2. This implies

$$Q_k = T^2/2 + \tau T + X_k T^2. \tag{3}$$

It follows from (2) that

$$\Delta^{(K)} = T\left[\frac{1}{2} + \frac{\tau}{T} + \frac{1}{K}\sum_{k=0}^{K-1} X_k\right]. \tag{4}$$

The time-average age is $\lim_{K\to\infty} \Delta_K$ and one would expect the normalized sum in (4) to converge to an average value of X_k. However, the missing frame process X_k has memory and modeling X_k is the key to characterizing the average age.

We first observe that $X_k = 0$ if $Y_k \leq \tau$. Similarly, $X_k = 1$ if $Y_k > \tau$ but $Y_{k-1} \leq T + \tau$. In principle, this process could continue indefinitely; that is, frame $k - i$ could be displayed at time T_k as long as $Y_{k-i} \leq iT + \tau$. In practice, however, delayed frames will be preempted. In order to model preemption, we assume that frame k will be preempted (i.e. discarded by the system) if $Y_k > (B-1)T + \tau$, where $B > 1$ is a backlog parameter. This constraint ensures that no more than B updates are kept in the system. Under this assumption, frame k may be displayed at times $T_k, T_{k+1}, \ldots, T_{k+B-1}$, but if it fails to be displayed by T_{k+B-1} then it will never be displayed. This implies

$$X_k = \min\left\{i \geq 0 \,\middle|\, \begin{array}{l} Y_{k-i} \leq iT + \tau, \\ Y_{k-i} \leq (B-1)T + \tau. \end{array}\right\} \tag{5}$$

We note in (5) that the first condition ensures that frame $k - i$ is delivered by time T_k and the second condition enforces the buffering limit.

C. Missing Updates Markov Chain Analysis

Given that buffering outdated updates is generally a bad idea, we now present a Markov chain analysis of X_k process when $B = 2$. When $B = 2$, (5) simplifies to

$$X_k = \min\{i \geq 0 | Y_{k-i} \leq \min(1, i)T + \tau\}. \tag{6}$$

This implies

$$\mathrm{P}[X_k = 0] = \mathrm{P}[Y_k \leq \tau] \tag{7}$$

and for $j \geq 1$,

$$\mathrm{P}[X_k = j] = \mathrm{P}[Y_{k-j} \leq T + \tau, Y_{k-j+1} > T + \tau, \\ \ldots, Y_{k-1} > T + \tau, Y_k > \tau]. \tag{8}$$

With the assumption that the Y_k are iid, (7) and (8) imply that X_k has PMF

$$P_{X_k}(j) = \begin{cases} F_Y(\tau) & j = 0, \\ F_Y(T+\tau)\overline{F}_Y(T+\tau)^{j-1}\overline{F}_Y(\tau) & j \geq 1. \end{cases} \tag{9}$$

While (9) captures the marginal PMF of X_k, it does not fully reveal the Markov structure of the X_k process. In Appendix A, we show that X_k is described by the Markov chain shown in Figure 3 with transition probabilities

$$p_0 = \mathrm{P}[X_k = 0 | X_{k-1} = j] = F_Y(\tau), \tag{10}$$

and for $j \geq 1$,

$$p_1 = \mathrm{P}[X_k = 1 | X_{k-1} = j] = F_Y(T + \tau) - F_Y(\tau), \tag{11}$$

$$q = \mathrm{P}[X_k = j + 1 | X_{k-1} = j] = \overline{F}_Y(T + \tau). \tag{12}$$

The Markov chain is ergodic as long as $F_Y(T + \tau) > 0$. The stationary probabilities

$$\pi_j = \lim_{k \to \infty} \mathrm{P}[X_k = j] \tag{13}$$

satisfy

$$\pi_0 = \sum_{i=0}^{\infty} \pi_i F_Y(\tau) = F_Y(\tau), \tag{14}$$

$$\begin{aligned}
\pi_1 &= \overline{F}_Y(\tau)\pi_0 + \sum_{i=1}^{\infty} \pi_i [F_Y(T + \tau) - F_Y(\tau)] \\
&= \pi_0 + (1 - \pi_0)F_Y(T + \tau) - F_Y(\tau) \\
&= \overline{F}_Y(\tau)F_Y(T + \tau),
\end{aligned} \tag{15}$$

and for $j \geq 1$,

$$\begin{aligned}
\pi_j &= \overline{F}_Y(T + \tau)^{j-1}\pi_1 \\
&= \overline{F}_Y(\tau)F_Y(T + \tau)\overline{F}_Y(T + \tau)^{j-1}.
\end{aligned} \tag{16}$$

As we would expect, (16) is consistent with the PMF of X_k given in (9). Ergodicity of the Markov chain implies that

$$\lim_{K \to \infty} \frac{1}{K} \sum_{k=0}^{K-1} \mathrm{E}[X_k] = \sum_{j=1}^{\infty} j\pi_j = \frac{\overline{F}_Y(\tau)}{F_Y(T + \tau)}. \tag{17}$$

The next claim then follows from (4) as $K \to \infty$.

Theorem 1. *The average age of the system with frame period T, lag τ, $0 \leq \tau < T$, and backlog limit $B = 2$ is*

$$\Delta_2(T, \tau) = T\left[\frac{1}{2} + \frac{\tau}{T} + \frac{\overline{F}_Y(\tau)}{F_Y(T + \tau)}\right].$$

Theorem 1 provides a simple characterization of the average in terms of the distribution of update delivery times. When the system is designed cautiously, $\overline{F}_Y(\tau) \approx 0$ and virtually all updates are delivered on time. In this case, Theorem 1 says $\Delta_2(T, \tau) \approx T/2 + \tau$ where $\Delta_2(T, \tau)$ measures how out-of-date the current video frame is when frames are delivered on time. We see that both the frame lag τ and the period T contribute. Specifically $T/2$ is the average age between periodic updates; when the frame rate is low, the period T will be large. The frame lag τ is the response time of the system once a user action is incorporated in a game server update. The age metric incorporates both the lag in response time and the latency associated with periodic framing.

Despite the approximations made by the analytic model, we will see that it provides a surprisingly accurate calculation of
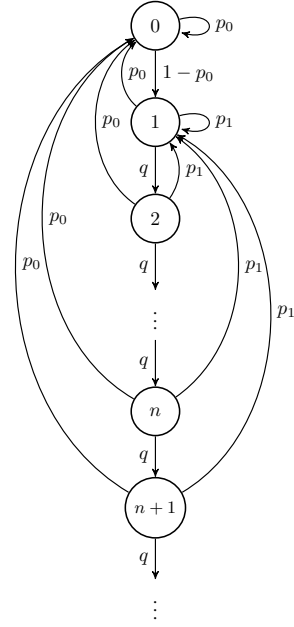

Fig. 3: The discrete-time Markov chain X_k.

the status age in a low-latency edge-cloud gaming system. For fixed T, we will see that Theorem 1 captures the complex way that the age varies with the lag τ. Nevertheless, Theorem 1 should be used carefully. Specifically, as the frame period T shrinks, the rate of updates/frames grows and queueing will cause the frame delivery time Y to increase.

D. Lag Periodicity of the Age

Theorem 1 describes the average age $\Delta_2(T, \tau)$ for $0 \leq \tau < T$. One may wonder what benefit may be obtained by a lag $\tau \geq T$. To examine this, it is sufficient to consider lag $\tau = jT + \tau_0$ with $0 \leq \tau_0 < T$. Here we refer to τ_0 as the local lag and j as the frame lag. At the mobile client, the display rule is that the most recently received frame is displayed at time $kT + \tau$. Thus, when $\tau = jT + \tau_0$, the most recently received frame at time $(k + j)T + \tau_0$ is displayed. However, as k and $k + j$ are arbitrary frame indices, we see the display rule depends only τ_0. That is, in the context of (5), the display rule at time T_k is to display the most recent frame $k - i$ such that $Y_{k-i} \leq iT + \tau_0$. What changes, however, is that the frame lag j enables additional frames to be displayed rather than discarded. Specifically, the second condition in (5) requires

$$Y_{k-i} \leq (B - 1)T + \tau = (B + j - 1)T + \tau_0. \tag{18}$$

To summarize, when $\tau = jT + \tau_0$, the X_k process is given by

$$X_k = \min\left\{i \geq 0 \,\middle|\, \begin{array}{l} Y_{k-i} \leq iT + \tau_0, \\ Y_{k-i} \leq (B + j - 1)T + \tau_0. \end{array}\right\} \tag{19}$$

Comparing (19) and (5), we see that buffering limit B and lag $\tau = jT + \tau_0$ is identical to buffering limit $B + j$ and lag $\tau = \tau_0$. It then follows that

$$\Delta_B(T, jT + \tau_0) = \Delta_{B+j}(T, \tau_0). \tag{20}$$

Parameter	Value
game-edge propagation delay	5 msec, 1 Gbps link
game-edge bandwidth	1 Gbps
edge-client channel	802.11a with 6Mbps
command packet size	60 Bytes
frame size	4 KBytes
game status packet	1 KBytes
frame rate	30
command generation	Poisson process of rate 10
Simulation time	1000 sec
Transport protocol	UDP

TABLE I: Simulation Parameters

Figure	analytical $F_Y(y)$
4a	(21) $y_0 = 5.32$, $\mu = 1/14.5$
4b	(22) $y_0 = 5.32$, $\mu = 1/7.95$
4c	(22) $y_0 = 11$, $\mu = 1/4.9$
6a	(21) $y_0 = 6.32$, $\mu = 1/16.6$
6b	(22) $y_0 = 6.9$, $\mu = 1/8.1$

TABLE II: Analytical $F_Y(y)$ parameters

When the system is well designed, $F_Y(BT + \tau) = 1 - \epsilon$ and almost all updates are delivered on time. In this case, the buffering limit B has almost no impact on system performance in that $\Delta_{B+j}(T, \tau_0) \approx \Delta_B(T, \tau_0)$. In such practical cases, the age is a periodic function of the lag τ. Henceforth we consider lags only in the interval $(0, T)$.

IV. EVALUATION

In this section we evaluate the role of the average age of the system as a metric in providing richer information on design and optimization of soft real-time interactive cloud applications. The architecture of our simulated cloud gaming system is illustrated in Figure 1 which consists of the client and server module connected through wireless network.

To evaluate the proposed analytical model, we implemented a mobile cloud gaming scenario in ns-3 where the main parameters of this simulation are summarized in Tables I and II. The simulation parameters are chosen to capture the complexity of cloud-gaming systems including bandwidth constraints, rendering, and encoding/decoding delays. In our simulations, we have considered both single-player games and multi-player games where in the multi-player scenario, different players submit their commands to the same game server, and the game server sends game status update messages to edge servers responsible for rendering the frames. Each client receives its frames from the edge server which is associated to. This model corresponds to geographically distributed players, each connected via a separate edge cloud server.

Throughout this section, the term single-server refers to a scenario where processing tasks for both game status update and frame rendering are conducted in the same server. Similarly, the term multi-server refers to the alternative case where game status update is performed in a central server and frame rendering is done in edge cloud servers.

We start with a simple example in which the single-server execution duration has an exponential distribution with expected value $1/\mu$ and the transmission time is a constant y_0 such that $y_0 < T$. In this case, random variable Y has the shifted exponential distribution

$$F_Y(y) = \begin{cases} 0 & y < y_0, \\ 1 - e^{-\mu(y-y_0)} & y \geq y_0. \end{cases} \quad (21)$$

Similarly, in the multi-server scenario, if both game and rendering servers have exponentially distributed execution times each with expected value $1/\mu$ and the transmission time is a constant y_0 such that $y_0 < T$, then Y has the shifted Erlang distribution with shape parameter $k = 2$ such that

$$F_Y(y) = \begin{cases} 0 & y < y_0, \\ 1 - e^{-\mu(y-y_0)}\left(1 + \mu(y - y_0)\right) & y \geq y_0. \end{cases} \quad (22)$$

A. Analysis of the average age of the system

As indicated in section III-B, the age of the system depends on target frame rate and the fine tuning parameter τ. Figure 4 illustrates analytical and simulated average age of the system as a function of τ for a lightly loaded system. The system is lightly loaded in the sense that network delays can be approximated by the update/frame transmission times and that the game server and renderer do not have queued jobs. In these systems, the randomness in Y_k derives solely from the randomness in processing and rendering. In this figure, we have compared different types of servers (i.e., servers with exponential and Erlang execution interval and a multiplayer scenario with two separate exponential servers). Comparing the analytical results with corresponding simulation graph, we can see that Theorem 1 precisely describes the system.

In all three presented systems, the maximum age corresponds to having τ equal to the transmission delay. In other words, when the chosen τ consider service delay as 0, all arriving frames will be buffered and age for more than one frame period (interval between two consecutive frame display instants). If τ is less than the transmission delay, all arriving frames will be buffered but they age less since the upcoming frame display instant will be earlier. If τ is more than the transmission delay, the age of the system reduces since many frames will arrive before their corresponding display instant. As indicated in the presented results, depending on the frame rate and distribution of Y_k, there exists an optimal τ that minimizes the age. For τ larger than the optimal value, the performance will be degraded. By overestimating the system delay, most of the received frames will age unnecessarily and the average age of the system will grow.

As indicated in Figure 4, the range of the values for system age is on the order of 50-63 msec. Although this might look trivial, for delay sensitive applications, especially gaming systems, it would be considered large and as shown above, the age presents an interpretation for the system such that the client device can optimize its synchronization with the server and adapt over time. This is possible for both single-player
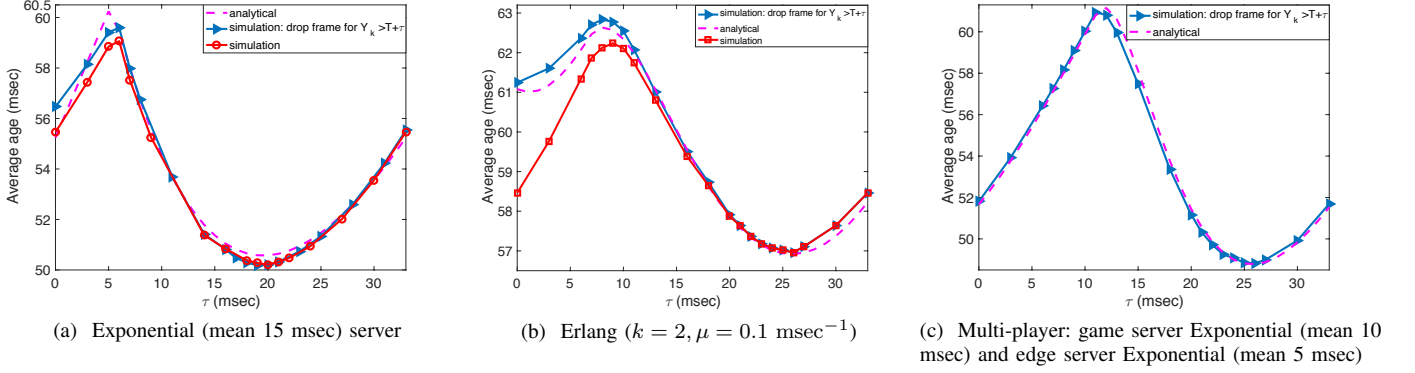
(a) Exponential (mean 15 msec) server

(b) Erlang ($k = 2, \mu = 0.1 \text{ msec}^{-1}$)

(c) Multi-player: game server Exponential (mean 10 msec) and edge server Exponential (mean 5 msec)

Fig. 4: Average age of the system as a function of τ for various servers
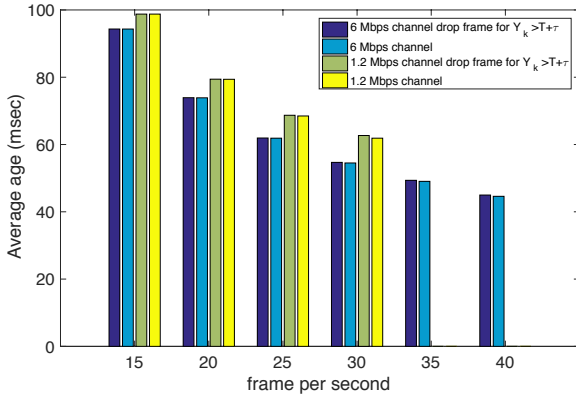

Fig. 5: Average age of the system as a function of frame rate for various servers

(Figures 4a and 4b) and multi-player games (Figure 4c).

In Figures 4a and 4b, in addition to the $B = 2$ scenario, we simulated the $B \to \infty$ scenario called "soft" policy. Comparing this soft policy with our model, we see that in lightly loaded systems for small values of τ, the soft policy will reduce age. However, for τ larger than the transmission delay, both policies correspond to the same amount of age. In fact, as indicated in section III-D, the soft policy would correspond to having the age as a periodic function of τ with periods equal to frame display period.

Figure 5 illustrates the dependency of the average age on the system frame rate. We have compared lightly loaded channel with a busy channel for both $B = 2$ and soft policy systems. In all cases, low frame rates increase the average age of the system since the frames are sampling the game infrequently. As expected, the average age is higher in busy channels since frames suffer from queuing delay more. Furthermore, the age captures the compromise between frame rate and channel congestion. As an example, a system with 30 fps in 1.2 Mbps channel updates the clients with the same age as a 25 fps system in 6Mbps channel.

B. Effect of channel load on age

To investigate the effect of suboptimal network and server resources on age of the system, we simulated two scenarios where clients are sending with rate 30 fps but the available channel is limited to 2 Mbps and 1.2 Mbps. Figures 6a and 6b illustrate the resulting age as a function of τ along with the corresponding analytical curves. Comparing Figures 4a and 6a, we can see that since the channel is not congested, the performance of the system is not significantly degraded. Furthermore, the distribution of Y_k still can be approximated as shifted exponential. For τ larger than the channel transmission delay and smaller than the optimal τ, the distribution of Y_k is more similar to Erlang.

Comparing Figure 6b with Figures 6a and 4a, we can see that in a more heavily loaded scenario, the age of the system has increased. Moreover, since the frames for different clients are queued in the network interface of the server, each game update experiences an exponential service time and an M/M/1 queue. Consequently, Y_k has Erlang ($k = 2$) distribution (see Figure 6c).

C. Multi-player scenario

In this section, we simulated a multi-player scenario with geographically distributed set of edge servers where each player is associated to a different edge server with different processing power and edge delay (i.e., transmission time from edge server to the client terminal). Figure 7 illustrates the variations of age versus τ for different players. Each client terminal is able to use the tuning parameter τ to optimize its synchronization with the game server. Furthermore, this figure exhibits the potential of age of a system as an important tool for assigning game sessions to different edge servers since it incorporates various parameters that affect the subjective quality of experience for players.

V. RELATED WORK

Most of the previous works in the area of mobile cloud gaming can be divided into two categories: i) Optimization of the user-perceived frame rendering of the game state using
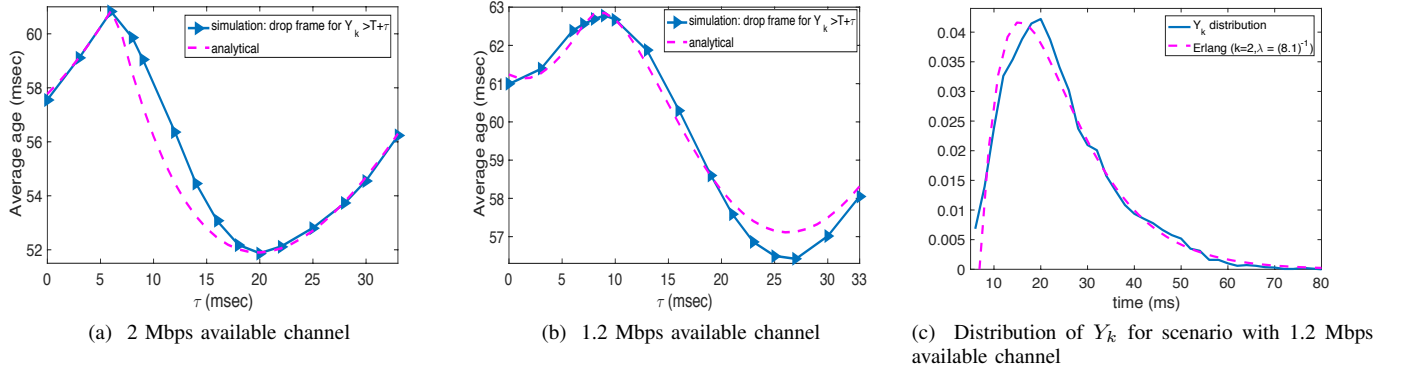
(a) 2 Mbps available channel

(b) 1.2 Mbps available channel

(c) Distribution of Y_k for scenario with 1.2 Mbps available channel

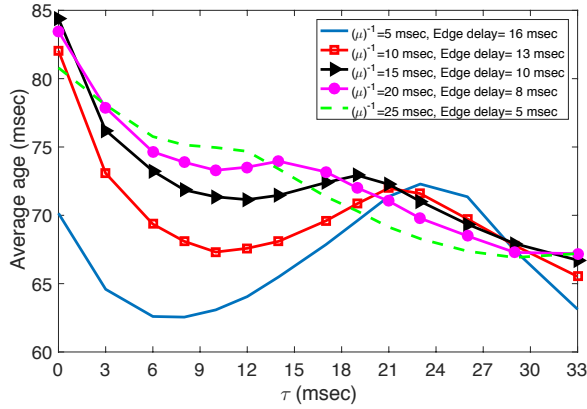Fig. 6: Average age of the system as a function of τ for various channel availability

Fig. 7: Average age of the system as a function of τ for different players of a geographically distributed multi-player game

a variety of predictive techniques [14], [15]. ii) Design, optimization, and evaluation of commercial cloud gaming systems [7] including NVIDIA GRID [16] and PS Now [17]. Outatime [18] is a predictive-based solution implemented on top of commercial games to reduce latency and substitute the lack of buffer with predicted future game frames based on user-specific behavior patterns.

Motivated by the popularity of commercial cloud gaming systems, we focus on design and optimization of edge-cloud gaming systems as described in [5]. Perhaps the most closely related system is the StreamMyGame (SMG) system evaluated in [2]. Just as in the experimental model evaluated here, the SMG system deploys a game server attached to the same local network as the client. Unlike this work, however, server processing times on the order of 300-400 ms dominated the response time of the SMG system [2].

It has been shown that user-perceived timeliness can be enhanced at the expense of consistency, a measure of how accurately a video frame describes the true game state. By contrast, we attempt to separate issues related to the depiction of the game state from the timeliness of the game state

information at the client. Our approach is based on timeliness of the state information at the mobile client, rather than how that state is depicted for the client. While a game status update generated by the game server at time t is a snapshot of the game state at time t, networking and rendering delays imply that updates will be old by the time it is displayed at the mobile client. In particular we will use *status age* to measure how the game status, as displayed on the player's screen, lags the true game state.

In the area of age of information, the analysis of the properties of queue-based models have dominated the emerging literature; see, for example [11], [13], [19]–[21]. However, these analytic models are approximations of real systems. A first effort to evaluate the accuracy of analytic models by simulation was provided in [22]. The ns-3 perfomance evaluation in Section IV of the cloud gaming system is similar in spirit in that our analytic model is only an approximation of the gaming system with both networking and processing delays.

VI. CONCLUSION

In this work, we presented a quantified representation of the user-perceived QoE of latency-sensitive real-time cloud-assisted applications such as gaming based on missing frames. As the objective of a cloud gaming system is the "timely" update of players regarding the status of the game, we employed an age of information metric to characterize the system performance. The age metric incorporates both the lag in response time and the latency associated with periodic framing. Our proposed metric, is in terms of both interaction delay and stream quality. Based on the proposed timeliness metric, we provide an analytical framework supported by extensive simulation for the problem of optimizing frame rate and lag synchronization of server and player. Based on the obtained results, age can be applied as a tool to synchronize game sessions based on the current status of the server. Furthermore, it can be used as a parameter in assigning edge servers to clients and designing resource allocation algorithms for cloud gaming systems. We have suggested that user QoE in a low latency gaming system can be captured by the delayed or

missed video frames, and proposed the average age of video frames as a QoE metric. However, we observe that further study is needed to determine whether the average frame age is an effective measure of *user-perceived* QoE for various types of games with different sensitivity to latency. It may be that other metrics derived from the missing frame process X_k (e.g., the variance of X_k) better capture the user QoE. For example, if we say the system is in outage when $X_k > 0$, the average duration of outages may be a suitable metric. We note that our Markov chain analysis of the X_k process permits analytic consideration of these alternate metrics.

APPENDIX A
MARKOV CHAIN ANALYSIS

The X_k process evolves according to the following rules:
- If $X_k = 0$, then
  - If $Y_{k+1} \leq \tau$, then $X_{k+1} = 0$; otherwise, $X_{k+1} = 1$.
- If $X_k = j > 0$, then
  - If $Y_{k+1} \leq \tau$, then $X_{k+1} = 0$.
  - If $\tau < Y_{k+1} \leq T + \tau$, then $X_{k+1} = 1$.
  - If $Y_{k+1} > \tau$ and $Y_k > T + \tau$, then $X_{k+1} = j + 1$.

For $\tau \leq T$, we now build a Markov chain from these rules. First we observe that for all $j \geq 0$,

$$\mathrm{P}[X_{k+1} = 0 | X_k = j] = \mathrm{P}[Y_{k+1} \leq \tau] = F_Y(\tau). \quad (23)$$

Of course, this implies $\mathrm{P}[X_{k+1} = 1 | X_k = 0] = \overline{F}_Y(\tau)$. Furthermore, for $j \geq 1$,

$$
\begin{aligned}
& \mathrm{P}[X_{k+1} = j + 1] \\
& = \mathrm{P}[X_k = j, Y_k > T + \tau, Y_{k+1} > \tau]. \quad (24)
\end{aligned}
$$

Note that $X_k = j \geq 1$ implies $Y_k > \tau$, and thus

$$
\begin{aligned}
& \mathrm{P}[X_{k+1} = j + 1 | X_k = j] \\
& = \mathrm{P}[Y_k > T + \tau, Y_{k+1} > \tau | X_k = j] \\
& = \mathrm{P}[Y_k > T + \tau | X_k = j] \\
& \quad \times \mathrm{P}[Y_{k+1} > \tau | Y_k > T + \tau, X_k = j] \\
& = \mathrm{P}[Y_k > T + \tau | X_k = j, Y_k > \tau] \mathrm{P}[Y_{k+1} > \tau] \\
& = \mathrm{P}[Y_k > T + \tau | Y_k > \tau] \mathrm{P}[Y_{k+1} > \tau] \\
& = \frac{\mathrm{P}[Y_k > T + \tau]}{\mathrm{P}[Y_k > \tau]} \mathrm{P}[Y_{k+1} > \tau] \quad (25) \\
& = \overline{F}_Y(T + \tau), \quad (26)
\end{aligned}
$$

where (26) follows from (25) because the Y_k are iid. With similar logic, we observe for $j \geq 1$ that

$$
\begin{aligned}
& \mathrm{P}[X_{k+1} = 1 | X_k = j] \\
& = \mathrm{P}[Y_k \leq T + \tau, Y_{k+1} > \tau | X_k = j] \\
& = \mathrm{P}[Y_k \leq T + \tau | X_k = j] \\
& \quad \times \mathrm{P}[Y_{k+1} > \tau | Y_k > T + \tau, X_k = j] \\
& = \mathrm{P}[Y_k \leq T + \tau | Y_k > \tau] \mathrm{P}[Y_{k+1} > \tau] \\
& = \mathrm{P}[\tau < Y_k \leq T] = F_Y(T + \tau) - F_Y(\tau). \quad (27)
\end{aligned}
$$

We observe that (23), (26) and (27) verify that X_k is described by the discrete-time Markov chain shown in Figure 3.

REFERENCES

[1] (2016) Cisco Visual Networking Index: Forecast and Methodology, 2015–2020. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] K.-T. Chen, Y.-C. Chang, H.-J. Hsu, D.-Y. Chen, C.-Y. Huang, and C.-H. Hsu, "On the quality of service of cloud gaming systems," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 480–495, 2014.

[3] 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 *Proceedings of the 11th annual workshop on network and systems support for games*. IEEE Press, 2012, p. 2.

[4] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, ""gaminganywhere: an open cloud gaming system"," in *Proceedings of the 4th ACM multimedia systems conference*. ACM, 2013, pp. 36–47.

[5] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "A hybrid edge-cloud architecture for reducing on-demand gaming latency," *Multimedia Systems*, vol. 20, no. 5, pp. 503–519, 2014.

[6] Y.-C. Chang, P.-H. Tseng, K.-T. Chen, and C.-L. Lei, "Understanding the performance of thin-client gaming," in *IEEE International Workshop on Communications Quality and Reliability (CQR)*. IEEE, 2011, pp. 1–6.

[7] R. Shea, J. Liu, E. C. H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, July 2013.

[8] H. Nam, K.-H. Kim, and H. Schulzrinne, "QoE Matters More Than QoS: Why People Stop Watching Cat Videos," *IEEE International Conference on Computer Communications*, 2016.

[9] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 325–338, 2015.

[10] J. Wu, C. Yuen, N. M. Cheung, J. Chen, and C. W. Chen, "Modeling and optimization of high frame rate video transmission over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2713–2726, April 2016.

[11] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. IEEE INFOCOM*, 2012.

[12] M. Claypool, "The effect of latency on user performance in real-time strategy games," *Elsevier Computer Networks*, vol. 49, no. 1, Sep. 2005.

[13] M. Costa, M. Codreanu, and A. Ephremides, "On the age of information in status update systems with packet management," *IEEE Transactions on Information Theory*, vol. 62, no. 4, pp. 1897–1910, April 2016.

[14] C. Savery, T. Graham, and C. Gutwin, "The human factors of consistency maintenance in multiplayer computer games," in *Proceedings of the 16th ACM international conference on Supporting group work*. ACM, 2010, pp. 187–196.

[15] C. Savery, N. Graham, C. Gutwin, and M. Brown, "The effects of consistency maintenance methods on player experience and performance in networked games," in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 2014, pp. 1344–1355.

[16] "Nvidia," http://www.nvidia.com/object/cloud-gaming.html.

[17] "Psnow," https://www.playstation.com/en-us/explore/playstationnow/.

[18] K. Lee *et al.*, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 151–165.

[19] S. Kaul, R. Yates, and M. Gruteser, "Status updates through queues," in *Conf. on Inf. Sciences and Systems*, Mar. 2012.

[20] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, 2013, pp. 66–70.

[21] C. Kam, S. Kompella, G. D. Nguyen, and A. Ephremides, "Effect of message transmission path diversity on status age," *IEEE Trans. Info Theory*, vol. 62, no. 3, pp. 1360–1374, March 2016.

[22] C. Kam, S. Kompella, and A. Ephremides, "Experimental evaluation of the age of information via emulation," in *Military Communications Conference, MILCOM 2015 - 2015 IEEE*, Oct 2015, pp. 1070–1075.