Measuring The Latency of Cloud Gaming Systems*

Kuan-Ta Chen[1], Yu-Chun Chang[12], Po-Han Tseng[1],
Chun-Ying Huang[3], and Chin-Laung Lei[2]

[1]Institute of Information Science, Academia Sinica
[2]Department of Electrical Engineering, National Taiwan University
[3]Department of Computer Science, National Taiwan Ocean University

ABSTRACT

Cloud gaming, i.e., *real-time game playing via thin clients*, relieves players from the need to constantly upgrade their computers and deal with compatibility issues when playing games. As a result, cloud gaming is generating a great deal of interest among entrepreneurs and the public. However, given the large design space, it is not yet known which platforms deliver the best quality of service and which design elements constitute a good cloud gaming system.

This study is motivated by the question: *How good is the real-timeliness of current cloud gaming systems?* To address the question, we analyze the response latency of two cloud gaming platforms, namely, OnLive and StreamMyGame. Our results show that the streaming latency of OnLive is reasonable for real-time cloud gaming, while that of StreamMyGame is almost twice the former when the StreamMyGame server is provisioned using an Intel Core i7-920 PC. We believe that our measurement approach can be generally applied to PC-based cloud gaming platforms, and that it will further the understanding of such systems and lead to improvements.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; J.7 [**Computers in Other Systems**]: Command and control, Consumer products, Real time; K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Measurement, Performance

1. INTRODUCTION

Thin clients have become increasingly popular in recent years, primarily because of the high penetration rate of broadband Internet access and the use of cloud computing technology to build large-scale data centers. The massive computation and storage resources of data centers enable users to shift their workload to remote servers. As a result, thin clients are more convenient and also more powerful (with the supply from remote servers) than traditional fat clients. Thus, today, it is not uncommon for people work and play by accessing remote computers via thin clients.

Although the advantages of thin clients have been demonstrated in many applications, *computer games* in particular have benefited from the advances in the thin client technology. One of the reasons is that the *overhead* of setting up a game can be significant because game software is becoming increasingly complex. As a result, players are often restricted to one computer and cannot play games anywhere, anytime. In addition, trying a new game can be difficult because there may have software and hardware *compatibility issues* between the game and players' computers. Hence, players may be faced with a dilemma of whether to upgrade their computers or forgo the opportunity to try a new game.

Cloud gaming, i.e., *real-time game playing via thin clients*, offers solutions for all the above-mentioned issues. Cloud gaming frees players from the need to constantly upgrade their computers as they can now play games that host on remote servers with a broadband Internet connection and a thin client. Here a thin client can be a lightweight PC, a TV with a set-top box, or even a mobile device. Consequently, there are no set up overheads or compatibility issues if players wish to try a game because all the hardware and software is provided in the data centers by the game operators. Given these potential advantages for both game developers and consumers, cloud gaming has been seen a possible paradigm that would change the way computer games are delivered and played.

Cloud gaming has already generated a great deal of interest among entrepreneurs, venture capitalists, and the general public. Several startup companies offer or claim to offer cloud gaming services, such as OnLive[1], StreamMyGame[2], Gaikai[3], G-Cluster[4], OTOY[5], and T5-Labs[6], though their realizations may be very different. Some of the services are only accessible via thin clients on a PC (either native or browser-based applications), while others can be accessed via a TV with a set-top box. Quite a few design alternatives

*Area chair: Wei Tsang Ooi

†This work was supported in part by the National Science Council under the grant NSC100-2628-E-001-002-MY3.

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.

[1]http://www.onlive.com/
[2]http://www.streammygame.com/
[3]http://www.gaikai.com/
[4]http://www.gcluster.com/
[5]http://www.otoy.com/
[6]http://www.t5labs.com/

can be adopted when implementing a cloud gaming service, such as 1) the way the existing game software is modified and run on the server; 2) the way the game screen is encoded (on the server) and decoded (on the client); 3) the way game streaming data is delivered to the client; and 4) the way short-term network instability is handled to maintain the game's responsiveness and graphical quality. Because of the large design space of such systems, *it is not yet known which platforms deliver the best quality of service and which design elements constitute a good cloud gaming system.*

In this paper, we perform an anatomic analysis of the latency of two cloud gaming platforms, namely, OnLive and StreamMyGame. We chose them because they were the only PC-based cloud gaming platforms on the market at the time of writing (April 2011). Measuring the latency of cloud gaming systems is challenging because most of the systems are *proprietary and closed*, i.e., none of the source codes and internals of the cloud servers, game software, and thin clients are available. In addition, cloud servers and the game software running on them cannot be modified because they are managed centrally by the service operators. Despite of these restrictions, we propose a measurement methodology that can assess the delay components of a cloud gaming system even if the system is proprietary and closed. To the best of our knowledge, this work is the first to anatomically analyze the delay components of cloud gaming systems.

Our contribution of this work is two-fold: 1) We propose a methodology for measuring the latency of cloud gaming systems. It can be applied even if the system is closed and not modifiable. 2) We profile two commercial cloud gaming platforms, OnLive and StreamMyGame, and show that the streaming latency of OnLive is reasonable for real-time cloud gaming, while that of StreamMyGame is almost twice the former when the StreamMyGame server is provisioned using an Intel Core i7-920 (2.66 GHz) PC.

The remainder of this paper is organized as follows. The next section provides a review of related works. We present our measurement methodology and the measured results in Section 3. Section 4 contains our concluding remarks.

2. RELATED WORK

Nieh and Laih [4] proposed using slow-motion benchmarking to evaluate the performance of several thin client platforms on various tasks. Unfortunately, the technique cannot be applied to cloud gaming systems because games would have to be modified so that they could run in slow motion. Besides, the performance metrics used, i.e., the amount of data transferred, are not sufficient to accurately assess the temporal and spatial quality of cloud gaming services.

3. LATENCY MEASUREMENT

3.1 Evaluated Platforms and Games

When OnLive[1] was introduced at the Game Developer's Conference in 2009, it attracted a significant amount of attention from the mass media and the public. The service is well-known partly because of its high-profile investors and partners, including Warner Bros, AT&T, Ubisoft, Atari, and HTC. It was released in June 2010 and now offers more than 120 games as of September 2011. OnLive's client is available on Microsoft Windows, Mac OS X, and as a TV set-top box. The minimum bandwidth requirement is 3 Mbps, but an Internet connection of 5 Mbps or faster is recommended. All the games are delivered in HDTV 720p format.

Unlike OnLive, which hosts game servers in its own data centers, StreamMyGame[2] (SMG) offers software solutions for remote game playing. The service was launched in October 2007 and currently supports more than 120 games from Windows to Windows- and Linux-based clients as of September 2011. It supports game streaming in a variety of resolutions, from 320x240 to 1920x1080 (1080p), which require an Internet connection between 256 Kbps (320x240) and 30 Mbps (1080p). Although StreamMyGame offers a software-based platform rather than a centralized service, we consider it a perfect fit for this present study because we focus on the performance of game streaming mechanisms rather than the capacity of service providers.

Because of the architectural differences of OnLive and SMG, their system configurations in our experiments are not identical. The main difference is that the OnLive server is operated by OnLive Inc., while the SMG server is operated by ourselves and installed with the SMG server software developed by Tenomichi/SSP Ltd. For a fair comparison, the OnLive client and SMG server/client are running on desktop computers equipped with two Intel Core i7-920 processors running at 2.66 GHz; in addition, both platforms are configured to provide remote gaming in 1280x720 resolution with same 3D effect settings.

We used three games, namely, Lego Batman: The Videogame (Batman), Warhammer 40,000: Dawn of War (DOW), and F.E.A.R. 2: Project Origin (FEAR) in this work. We chosen the games because they are supported by both platforms and they represent three game genres. Lego Batman is an action-adventure game, FEAR is a typical first-person shooter (FPS) game, and Warhammer is a real-time strategy (RTS) game. FPS games normally require a high rate of game screen updates, whereas the pace of adventure and RTS games is relatively slower with an omnipresent viewpoint [3].

3.2 Anatomy of Delay Components

We segment a cloud gaming system's response delays (RD) to players' commands into three components:

- **Network delay** (ND): the time required to deliver a players' command to the server and return a game screen to the client. It is usually referred to as the network round-trip time (RTT).
- **Processing delay** (PD): the difference between the time the server receives a player's command (from the client) and the time it responds with a corresponding frame after processing the command.
- **Playout delay** (OD): the difference between the time the client receives the encoded form of a frame and the time the frame is decoded and presented on the screen.

Unlike the network delay, which can be measured using tools like `ICMP` ping, the processing delay (at the server) and playout delay (at the client) occur *internally* in the cloud gaming system and are not accessible from outside. Our goal is to measure both delays accurately by using only external probes.

3.3 Response Delay Measurement

To measure the response delay (RD), which equals to ND+PD+OD, of a cloud gaming system, we exploit the fact

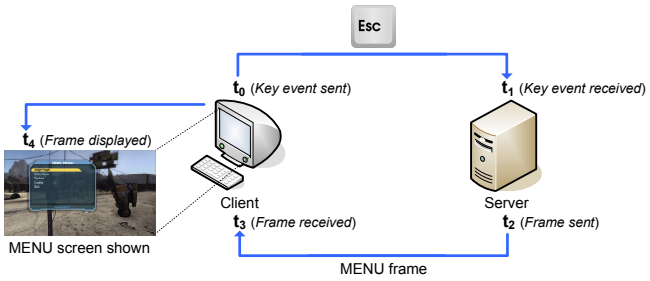

Figure 1: The key events in the measurement of the response delay of a cloud gaming platform by invoking the menu screen.

that most games support a hot key, which is used to access a menu screen anytime during game play. The key is usually the `ESC` key for computer games and the `START` button for console games. Without loss of generality, we assume the ESC key is the hot key for invoking the menu screen. As illustrated in Figure 1, assuming the ESC key is pressed at time t_0 and the menu screen is shown to the user at time t_4, the time difference $(t_4 - t_0)$ corresponds to the response delay of the ESC key. However, the processing delay $(t_2 - t_1)$ and the playout delay $(t_4 - t_3)$ are not visible and cannot be measured directly.

To determine the response delay, we utilize the hooking mechanism[7] in Windows to inject our instrumentation code into the OnLive and SMG clients. We use the `detours` library to intercept the `IDirect3dDevice9::EndScene()` function, which is called when a Direct3D application finishes drawing graphics on a hidden surface and is about to present the surface on the screen. We then use the following procedure to measure the response delay:

1. Simulate an `ESC` key press event by calling the `SendInput()` function at time t_0.
2. Each time the game screen is updated, we examine the colors of a specific set of pixels to determine if the menu screen is displayed.
3. Wait until the menu screen appears (and note the time as t_4).

We can therefore calculate the response delay by subtracting t_0 from t_4[8]. Each run of the procedure yields a sample of the response delay; thus, we can repeatedly execute the procedure to obtain a set of response delay samples.

3.4 Response Delay Decomposition

During the experiments, we periodically measure the network round-trip time using `ICMP` ping as ND samples. Since we have ND and the sum of ND, PD, and OD constitutes the response delay, we only need to determine either PD (processing delay) or OD (playout delay) in order to decompose all the components. More specifically, we need to further determine the occurrence of t_3 (shown in Figure 2) in order to obtain PD and OD.

[7]The Windows hooking mechanism is invoked by calling the `SetWindowsHookEx` function. It is frequently used to inject code into other processes.

[8]Certain games, such as FEAR, intentionally postpone the appearance of the menu screen for a short period of time. Thus, we have calibrated the response delays by subtracting the intentional menu appearance delay from the measured delays obtained using the procedure.

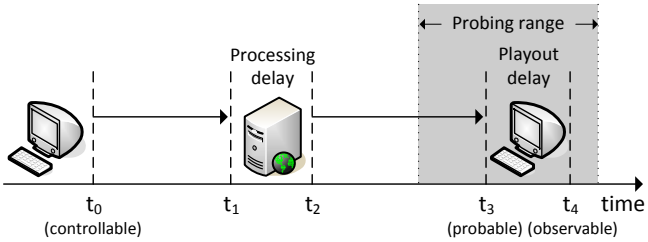

Figure 2: The decomposition of the response delay. The gray area represents the range in which the `recv()` call may be started to be blocked in order to probe the exact location of t_3.

The rationale behind probing t_3 is that it is the time the menu screen delivered to the client from the server. Thus, if incoming data is (intentionally) blocked (by us) on the client earlier than t_3, the menu screen will not be shown until the blocking is cancelled. On the other hand, if incoming data is blocked later than t_3, the menu screen will be displayed despite that no further screen updates are received and shown as long as the incoming data blocking is sustained.

To facilitate the data blocking mechanism, we hook the `recvfrom()` function, which is called when the thin clients attempt to retrieve a UDP datagram from the UDP/IP stack. The measurement procedure is as follows:

1. Simulate an `ESC` key press event by calling the `SendInput()` function at time t_0. Also, compute t_{block} as a random time between $RD - 100$ ms and $RD + 50$ ms[9], assuming the playout delay shorter than 100 ms.
2. If the menu screen appears before t_{block}, record the time as t_{menu} and terminate the procedure. Otherwise, at t_{block}, temporarily block all the subsequent `recvfrom()` calls for one second[10].
3. Wait until the menu screen appears (and note the time as t_{menu}).

If t_{menu} is later than $t_{block} + 1$ sec, we consider that the blocking is successful and t_3 should be some time later than t_{block}. In this case, t_{block} is added to the set $t_{block\_succeeded}$. On the other hand, if t_{menu} is earlier than t_{block}, we consider that the blocking is failed and t_3 must be some time earlier than t_{menu}. In this case, t_{menu} is added to the set t_{block_failed}. By repeating the procedure a number of times, we can obtain a set $t_{block\_succeeded}$ that are earlier than t_3, and another set t_{block_failed} that are later than t_3, where t_3 must lie approximately at the boundary of the two sets.

We then estimate t_3 as the point that yields the minimum sum of the two density functions formed by $t_{block\_succeeded}$ and t_{block_failed} respectively, where each density function is computed as the mixture of the Gaussian density functions centered at each element with a standard deviation of any reasonable magnitude[11]. After estimating t_3, we can com-

[9]The 50-ms interval is chosen arbitrarily in order to leave a "safe zone" that ensures the menu screen will be blocked with a non-zero probability.

[10]The one-second interval is chosen arbitrarily in order to determine whether or not the menu screen is blocked. Other values can also apply without affecting the measurement results.

[11]In our experiment, we use a standard deviation of 20 ms; however, other values of the same order of magnitude would yield nearly identical t_3 estimates.

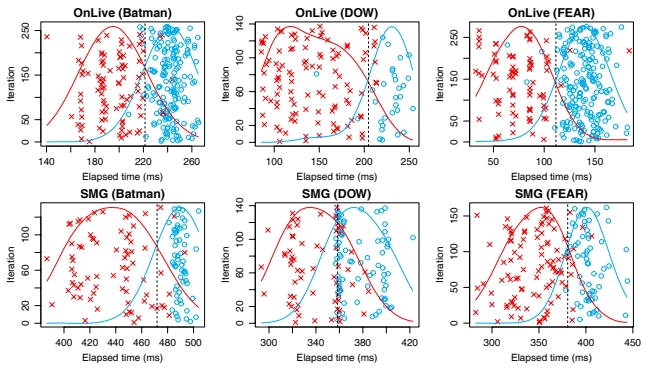

Figure 3: The scatter plots of $t_{block\_succeeded}$ and t_{block_failed} samples, denoted by red crosses and blue circles respectively. The vertical dashed lines denote the estimates of t_3 minus the network delay.

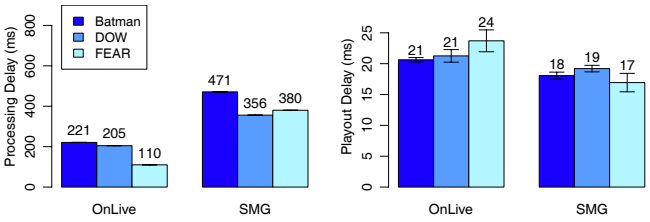

Figure 4: The estimated processing delays and playout delays of both cloud gaming platforms.

pute PD (server processing delay) as $t_3 - t_0 -$ ND and OD (playout delay) as $t_4 - t_3$.

3.5 Measurement Results

Figure 3 shows the scatter plots of $t_{block\_succeeded}$ and t_{block_failed} samples, denoted by red crosses and blue circles respectively. Note that we have subtracted all the samples by ND in order to remove the effect of network delay between our PC and the OnLive data center. We can observe that the ranges of $t_{block\_succeeded}$ and t_{block_failed} sets are not disjoint. This is reasonable because there may be fluctuations in the server's and client's workloads, and the network delay may vary due to network congestion. Because t_3 (and consequently PD and OD) are inferred based on a set of iterations, we validate the robustness of t_3 by cross-validation. That is, instead of using the data from all iterations, we select 50 iterations at random and use them to estimate t_3. The procedure is repeated at least 30 rounds. If the estimated values for t_3 in different rounds are close, we can confirm that the estimated PD and OD are reliable and not susceptible to measurement noises.

Figure 4 shows the average processing delays and playout delays as well as their 95% confidence bands of OnLive and SMG when Batman, DOW, and FEAR are played. As can be seen, the confidence bands of both processing and playout delays are fairly small, which indicates that our proposed measurement technique produces robust estimates of the delay components. From the graph, OnLive's processing delays are approximately half of those of SMG. This should be because OnLive utilizes a more efficient, possibly hardware-based, H.264 encoder to encode game screens in real time. It could also simply because the processing power of our SMG server is much lower than that of the OnLive server. Taking a closer look at OnLive, its processing delay is approximately 200 ms for Batman and DOW and 100 ms for FEAR. We consider FEAR's shorter processing delay is an intentional arrangement by OnLive because FPS games are known especially susceptible to lags [1,3], and the shorter delay may be the consequence of using higher-performance servers[12] for the game. Considering action and RTS games are slower-paced, OnLive's differential resource provisioning looks a reasonable strategy to provide an overall satisfactory gaming experience to players [2].

As to the average playout delays of both platforms, OnLive spends 20–30 ms and SMG spends ≈15–20 ms in frame decoding and display. Both systems perform similarly well in this aspect, as such short playout delays would not have a serious impact on the gaming experience.

To summarize, OnLive's overall streaming delay (i.e., the processing delay at the server plus the playout delay at the client) for the three games is between 135 and 240 ms, which is acceptable if the network delay is not significant. On the other hand, real-time encoding of 720p game frames seem to be a burden to SMG on an Intel i7-920 server because the streaming delay can be as long as 400–500 ms. Investigating whether the extended delay is due to design/implementation issues of SMG or it is an intrinsic limit of software-based cloud gaming platforms will be part of our future work.

4. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a general methodology to measure the latency components of cloud gaming systems, even those that are proprietary and closed. We applied the methodology on two platforms, OnLive and StreamMyGame, and identified that OnLive implements a game-genre-based differential resource provisioning strategy to provide sufficiently short latency for real-time gaming. On the other hand, StreamMyGame takes almost twice latency to provide 720p real-time game graphics with an Intel Core i7-920 PC, which leaves us an issue to investigate whether the extended delay is due to an intrinsic limit of software-based cloud gaming systems.

In our future work, we will continue to improve the applicability and accuracy of the proposed methodology. In addition, we plan to apply the methodology to more platforms for understanding their strengths and weaknesses, and derive general guidelines for designing quality cloud gaming systems.

5. REFERENCES

[1] Y.-C. Chang, K.-T. Chen, C.-C. Wu, C.-J. Ho, and C.-L. Lei. Online game QoE evaluation using paired comparisons. In *Proceedings of IEEE CQR 2010*, June 2010.

[2] K.-T. Chen, P. Huang, and C.-L. Lei. How sensitive are online gamers to network quality? *Communications of the ACM*, 49(11):34–38, Nov 2006.

[3] M. Claypool and K. Claypool. Latency and player actions in online games. *Commun. ACM*, 49:40–45, November 2006.

[4] A. M. Lai and J. Nieh. On the performance of wide-area thin-client computing. *ACM Trans. Comput. Syst.*, 24:175–209, May 2006.

[12] http://blog.onlive.com/2009/05/12/hopping-through-cloud-onlive/