Evaluating Streaming and Latency Compensation in a Cloud-based Game

Jiawei Sun, Mark Claypool
Worcester Polytechnic Institute
Worcester, MA, 01609 USA
{jsun|claypool}@wpi.edu

***Abstract*—The growth in cloud computing and network connectivity brings the opportunity for cloud-based game systems where players interact through a lightweight client that only displays rendered frames and captures input, while the heavyweight game processing happens on the cloud server. Compared to traditional games, cloud-based games present challenges in handling network latency and bitrate requirements. This work uses *Drizzle*, a custom cloud-based game system, to evaluate: 1) time warp to compensate for latency, and 2) graphics streaming to reduce network bitrates. A user study shows time warp mitigates the effects of latency on player performance, and laboratory experiments show graphics streaming provides bitrate reduction compared to the video streaming typically used by commercial cloud-based game systems.**

I. Introduction

Although still an emerging commercial market, cloud-based games are growing rapidly with the increase in the number of gamers and the global penetration of the Internet and smart phones. Established companies like Sony[1] and NVidia[2] are already invested in cloud-based game services, but other big players such as Google[3] and smaller companies such as Vortex[4] are looking to capture market shares.

Cloud-based games differ from traditional games in that game clients are relatively lightweight, only sending user input (e.g., key presses and mouse actions) and receiving game output (i.e., images and sounds). The heavyweight game logic – applying physics to game objects, resolving collisions, processing AI, etc. – and rendering are done at the server, with the game frames streamed to the client to display. A cloud-based game system offers advantages over traditional games including: modest client hardware requirements, no client game installation requirement, easier software piracy prevention, and fewer target platforms for developers.

While promising, cloud-based games face two major challenges when compared to traditional games: 1) *bitrates* – cloud-based games require significantly higher network bitrates from the server to the client [1] than do traditional network games; and 2) *latency* – cloud-based game clients cannot immediately act on player input but must instead send the input to the server, have it processed, the result rendered and sent back to the client for display [2].

Approaches to reduce bitrates can leverage innovations in image and video compression. However, the graphics-based nature of games present an opportunity for additional bitrate savings with only modest increases in client complexity by not necessarily streaming rendered game images but instead sending drawing information so the client can do the rendering [3].

Approaches to compensate for latency [4] have been widely used in commercial games. However, there has been limited scientific evaluation of their overall effectiveness and no specific evaluations covering the breadth of games and network conditions. Moreover, latency compensation techniques have not been applied to cloud-based game systems which are more restrictive in the techniques they can use given the client's limited knowledge of the game state and reduced hardware capabilities.

This work makes three contributions to this area: 1) the evaluation of a latency compensation technique, *time warp*, in a cloud-gaming system; 2) exploration of approaches to cloud-based game streaming to reduce network bitrates; and 3) evaluation using *Drizzle*, a cloud-gaming system designed and developed from scratch using the Dragonfly [5] game engine.

Results of a 30-person user study show that time warp for projectile weapons can ameliorate the effects of latency on player performance, but with a cost in player perception of the consistency of the game world. Results of system experiments show graphics streaming can significantly reduce network bitrates over video streaming, but still has higher bitrates compared to traditional network games. Both time warp and game streaming have considerable CPU cost on the server, particularly as the number of game objects increases.

The rest of this paper is organized as follows: Section II describes related work; Section III presents our methodology; Section IV analyzes the results; and Section V summarizes our conclusions and possible future work.

[1]https://www.playstation.com/en-us/explore/playstation-now/
[2]https://www.nvidia.com/object/cloud-gaming.html
[3]https://store.google.com/us/magazine/stadia
[4]https://vortex.gg/

II. Related Work

This section describes research related to this work: architectures for cloud-based game systems (Section II-A), studies of latency and games (Section II-B), and work on latency compensation algorithms (Section II-C).

A. Cloud-based Game Systems

While there is no single agreed-upon cloud system architecture, a four-layer architecture defined by Foster et al. [6] has often been used by researchers. Foster et al.'s model, from bottom to top, has a fabric layer, unified resource layer, platform layer and application layer. For our work, the Drizzle server runs at the application layer. Foster et al. also list cloud services at three different levels: infrastructure as a service, platform as a service, and software as a service. Cloud-based gaming in general, and Drizzle specifically, is an example of software as a service.

Cloud-based game systems can broadly be classified into graphics streaming and video streaming [7]. In graphics streaming, as done by de Winter et al. [3], instead of sending video, the server sends graphics commands to the client and the client renders the game images. In video streaming, as described by Shea et al. [8], the server is responsible for rendering the game scene, compressing the images as video, and then transmitting to the client. Both approaches reduce computation on the client versus a traditional network game architecture because only the server manages the entire game world.

The video streaming approach is discussed the most in cloud gaming research [8], [9] and is currently used by most existing commercial cloud-based systems since it reduces the workload on the client the most compared to the other two approaches.

Figure 1 depicts a cloud-based game system with video streaming. The lightweight ("thin") client on the left captures user interaction (e.g., a game controller button press) and sends the user command to the cloud gaming platform on the right. The user interactions are translated into game actions (e.g., a "shoot" command) which is processed by the game logic. When the game engine updates the game world, changes are rendered by the server's GPU and the rendered scene passed to the video encoder. The encoded video is then streamed to the thin client where it is decoded.

Drizzle, our custom cloud-based game system, supports both graphics streaming and video streaming, each of which is evaluated in this paper.

B. Latency and Games

The effects of latency has been studied for many traditional games [11]–[16]. While such work has helped better understand the impact of latency on traditional network games, the results may not hold for cloud-based games that have a fundamentally different underlying system architecture.

For cloud game systems, Chen et al. [2] discuss the effects of network latency (and other parameters) on the cloud-based game systems OnLive[5] and StreamMyGame.[6] The authors did not explicitly measure player performance with latency.

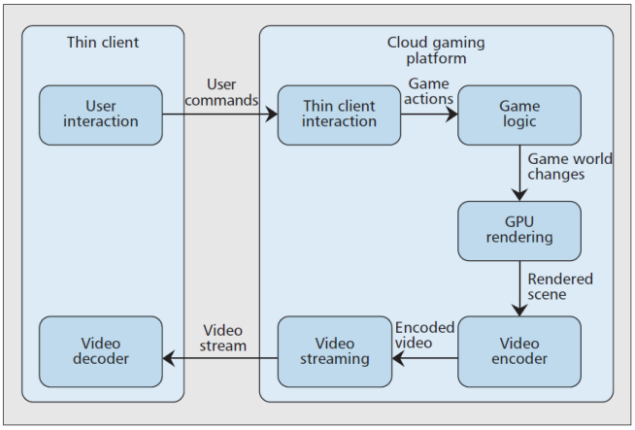

Fig. 1. Cloud-based game system with video streaming [10].

Jarschel et al. [17] conducted a user study in an emulated cloud game system, measuring the quality of experience for games users selected to play. Claypool and Finkel [18] present the results of two user studies that measured the objective and subjective effects of latency on cloud-based games. Sackl et al. [19] analyze the relationships between latency and player experience for cloud gaming. While more closely related to our work, these papers do not compare cloud-based games with and without latency compensation.

C. Latency Compensation

Bernier [20] describes methods game systems can use to compensate for network latencies, such as system-level treatments (e.g., real-time packet priorities), latency compensation algorithms (e.g., dead reckoning, sticky targets, aim dragging) and even game designs (e.g., deferred avatar response, geometry scaling). While providing an important foundation for latency compensation, the work does not provide scientific evaluation of the techniques.

Ivkovic et al. [21] carried out a controlled study of aiming in a first person shooter game with latency both with and without an aim assistance latency compensation technique. Lee and Chang [22] evaluated the effects of the latency compensation techniques time warp and interpolation on players in a commercial first person shooter game. Lee and Chang [23] continued evaluation of time warp with a custom first person shooter game, providing a guideline of 250 milliseconds as a limit for latency compensation. While helpful in better understanding latency compensation, and even used in traditional network games [24], such techniques have not been applied to cloud-based games.

Our work applies a popular latency compensation technique, time warp, to a cloud-based game and evaluates it with a user study and system CPU load measurements.

[5]https://en.wikipedia.org/wiki/OnLive
[6]https://en.wikipedia.org/wiki/StreamMyGame

III. Methodology

This section presents our methodology to evaluate graphics streaming and latency compensation in cloud-based games.

A. Cloud-based Game Streaming

There are generally three approaches to cloud-based game streaming, depicted in Figure 2. At the top left is image streaming, where the game server renders the game frames to be displayed as individual images, compresses them (e.g., as a JPEG image) and sends them to the client for decoding and playing. Next down, is video streaming, where the server renders the game frames as images, then encodes them into a video stream and sends the stream to the client for decoding and playing. In video streaming, the server applies intra-encoding for each image as in image streaming, but also takes advantage of the temporal redundancy in adjacent images, applying inter-encoding for a higher compression rate. At the bottom is graphics streaming, where the server does not render individual images but instead sends graphics information for each frame to the client whereupon the client renders the images. Unlike in image streaming or video streaming, graphics streaming requires both the server and client to have *a priori* knowledge of how to render the image from the underlying image data.

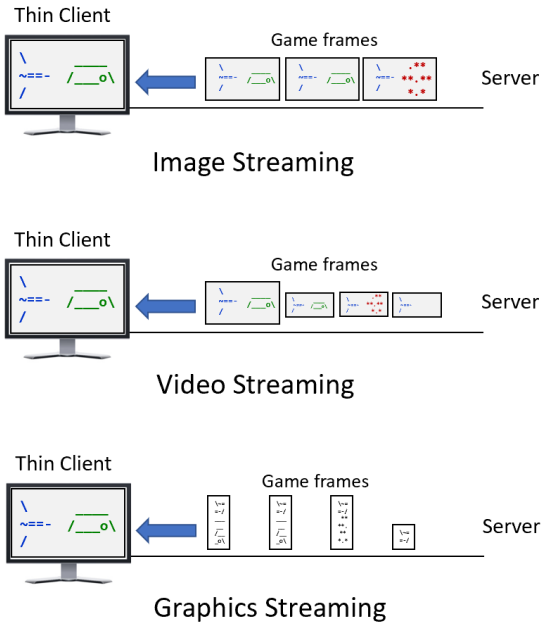

Fig. 2. Cloud streaming approaches. Top: image streaming, Middle: video streaming, Bottom: graphics streaming.

The three approaches depicted in Figure 3 – image streaming, video streaming and graphics streaming – have tradeoffs in the bitrates required by the network and the decoding and rendering complexity needed by the client. At the top left is image streaming, the simplest for the client, but with the highest network bitrate owing to the only modest compression afforded to the individual images. Video streaming requires more client complexity in that both intra- and inter-image decoding is needed, but with a significant bitrate reduction attained by the inter-encoding. Graphics streaming requires somewhat more complexity than video streaming in that the client itself must render the images from graphics data, but there is significant potential for lower bitrates than in video streaming. For comparison, traditional games are at the bottom right, having fairly low bitrates (5 kb/s to 124 kb/s) [1] but requiring complex clients, capable of running a game engine and doing a full render of the game world from game data.

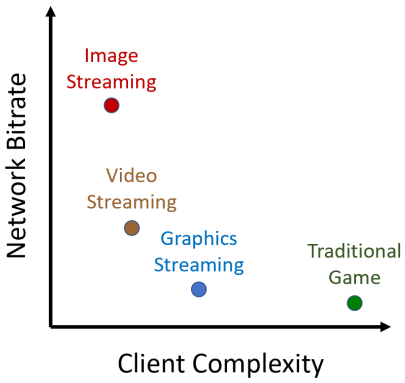

Fig. 3. Cloud-based game streaming tradeoffs.

B. Time Warp

Time warp is a latency compensation technique deployed at a game server, as depicted in Figure 4. With time warp, the player acts (e.g., shooting) based on the opponent's apparent position, but when the server gets the input Δt later, the opponent's actual position has moved. To compensate for this latency, the server warps time (and the game world) back by Δt, determines the outcome, and rolls the game world forward to the present time.

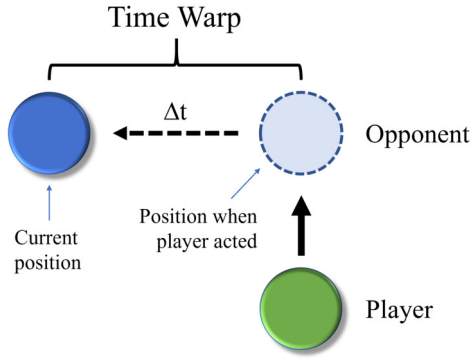

Fig. 4. Time warp – server rolls back game world Δt.

C. Drizzle

Drizzle is based on *Dragonfly* – a 2d game engine, full-featured enough to make a wide variety of arcade-style games.[7] The core engine is written in C++ and includes ASCII graphics rendered with SFML,[8] basic kinematics (velocity and acceleration), keyboard and mouse input, collision detection and resolution, and audio (sound effects and music).

We extended Dragonfly to support networking – specifically a network manager that uses TCP sockets. The network manager handles the network connection – a TCP socket and supports `accept()` and `connect()` calls for the server and client, respectively:

- The `NetworkManager::accept()` method sets up and then waits for a TCP/IP socket connection from the client, listening on a "well-known" port. This includes system calls to `socket()`, `bind()`, `listen()` and `accept()`.
- The `NetworkManager::connect()` method is invoked by the client to connect to the server, indicated by a hostname. This includes system calls to `socket()`, `getaddrinfo()`, and `connect()`.

The network manager has methods to `send()` data through the connected socket and `receive()` data from the connected socket. Receiving is non-blocking, so if there is no data pending, the method does not block but returns. Data arriving at the server triggers a network event to all interested[9] game objects.

Drizzle has a heavyweight server that does the game computations and a "thin" client that displays frames and transmits user input.

Pseudo-code for the Drizzle server is depicted in Listing 1. The server starts up the game engine and gets ready for a client to connect by waiting on a well-known port. Once the client connects, the server does all the game computations – updating positions of game objects, detecting and resolving collisions, and composing frames. However, unlike in a traditional game, there is no player sitting at the console (viewing the game and providing input). Instead, the server composes the game stream (Line 1) depending on if it's using image streaming, video streaming or graphics streaming. The game frame data is streamed down the network socket to the client. Player input from the keyboard and mouse at the Drizzle client is received at the server via the same network socket. The server applies all input received over the socket to the game as if the player were providing input to the keyboard and mouse of the server. When the game is over, the Drizzle server closes the network connection to the client.

Pseudo-code for the Drizzle client is depicted in Listing 2. The client starts up and immediately connects to the Drizzle server using the hostname provided by the player and the well-known port. Once connected, the client receives the game frame data sent by the server over the network socket, rendering the frame depending on the type of streaming (Line 2) and displaying it on the screen. The client also captures keyboard and mouse input from the player, sending all to the server. When the network connection to the server is closed, the client exits.

Drizzle can be configured to do image streaming, video streaming or graphics streaming. Image streaming is provided by using the SFML `capture()` method, sending the resulting image as a JPEG. Video streaming is provided via

Listing 1. Drizzle Server

```
0  start Dragonfly game engine
1  load game resources
2  wait until Client connects
3
4  // Run until end of game
5  while game is not over do
6
7    // Do one step in game loop
8    update positions for all game objects
9    detect and resolve collisions
10   compose game frame
11
12   // Prepare and send game frame
13   marshall game frame // image, video or graphics
14   send game frame to client
15
16   // Receive and process any player input
17   if network input then
18     receive data
19     apply input to game
20   end if
21
22 end while
23
24 // Shut it down
25 close network connection
26 exit
```

Listing 2. Drizzle Client

```
0  connect to Server
1
2  // Loop as long as connected
3  while network is connected do
4
5    // Receive and display frames
6    if network input then
7      receive data
8      draw frame on screen
9    end if
10
11   // Get user input and send to server
12   if keyboard/mouse input then
13     marshall input data
14     send input data to server
15   end if
16
17 end while
18
19 // Shut it down
20 exit
```

[7] http://dragonfly.wpi.edu/games/index.html\#trailers
[8] The Simple and Fast Multimedia Library, https://www.sfml-dev.org/
[9] Using the *Observer* software design pattern.

`ffmpeg`.[10] Graphics streaming is provided by sending send the bare minimum needed by the client to draw a packet – the character (1 byte), color (1 byte), and (x,y) location (4 bytes each).

D. Cloud Saucer Shoot

We created a Drizzle-compatible game called *Cloud Saucer Shoot* – an arcade-style shooting game set in space, where the player pilots a space ship against an endless, and ever increasing, horde of alien saucers. Figure 5 depicts the game. The player controls the blue ship on the left using arrow keys to move up and down. The green saucers move right to left, spawning in greater numbers as time progresses. If the ship is struck by a saucer, both are destroyed. The player fires bullets from the ship by pressing the spacebar. When a bullet hits a saucer, both are destroyed and the player is awarded 10 points. The player also receives 1 point each second the ship is alive. The goal is to shoot as many saucers as possible before being destroyed.

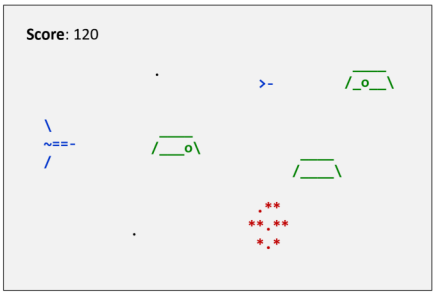

Fig. 5. Cloud Saucer Shoot. The player controls the ship on the left, firing blue bullets horizontally at the saucers. Bullets destroy one saucer upon contact. Saucers move right to left, destroying the ship upon contact.

E. Experiments

Our user study was conducted in a windowless computer lab with bright, fluorescent lighting.

Both the server and client ran on a laptop equipped with a 14" display, Intel i7 CPU 4 GHz processor, and 8 GB of memory running Windows 10. The system experiments were conducted on the same laptop. Given the lightweight nature of both the server, game and game client, the hardware was more than sufficient to provide a playout rate of 30 f/s.

Participants were volunteers solicited among graduate students in the department. First, the users heard a scripted brief about the study and signed an Institute Review Board (IRB) consent form. Next, they were asked to make themselves comfortable at the laptop by adjusting chair height and laptop screen tilt. Then users filled out a demographics and gaming experience survey coded using the Qualtrics survey tool.[11]

Users were told the objective of the Saucer Shoot game and how to control the ship and fire bullets. Users then played through a 15 second version of the game for practice. Results were not recorded for the practice session.

Immediately after the practice, users played 10 game sessions, each with an added latency selected from the range [0, 100, 200, 400, 800 milliseconds] using the network utility Clumsy.[12]. Five of the sessions had time warp on and the other 5 had time warp off. The game sessions were shuffled and users were blind to the amount of added latency and time warp.

After each of the 10 game sessions, users were asked to rate the responsiveness and graphics consistency from 0 (low) to 5 (high).

Playing through all game sessions typically took less than 15 minutes.

IV. Analysis

This section summarizes participant demographics (Section IV-A), presents analysis of the user experience with time warp (Section IV-B), and analyzes system impact for the game streaming options (Section IV-C).

A. Demographics

Thirty users participated in the study. All users were 20 to 30 years old. Twenty-five identified as male and 5 as female. Sixty percent of the users played online games every day, 25 percent once per week, and 15 percent once per month or less.

B. User Experience

Figure 6 depicts user game performance, measured by game score, versus added latency, both with and without time warp. The x-axis is the added latency (in milliseconds) and the y-axis is the user score (a combination of Saucers destroyed and seconds alive). There are two trendlines, one for sessions with time warp on and the other for sessions with time warp off. Each point is the mean score for all users at that latency, shown with standard error bars. From the graph, user performance decreases with added latency, both with and without time warp. Without time warp, the trend is a clear exponential decay. With time warp, there is an initial decline in performance from 0 to 100 milliseconds of added delay, but then performance does not decline appreciably from 100 to 400 milliseconds, before decreasing again at 800 milliseconds. 800 milliseconds is about the time it takes a Saucer to travel completely across the screen in the game.

Figures 7 and 8 depict user opinions of the game sessions in the presence of latency – specifically, responsiveness and consistency, respectively. User opinions are on a 6 point scale, from 0 (low) to 5 (high). In both graphs, the x-axis, data points, error bars and trend lines are as for Figure 6. From the graphs, the responsiveness of the game is about the same with and without time warp, evidenced by the overlapping red and blue trend lines in Figure 7. The inconsistency in the game

[10]https://www.ffmpeg.org/
[11]https://www.qualtrics.com/
[12]https://jagt.github.io/clumsy/

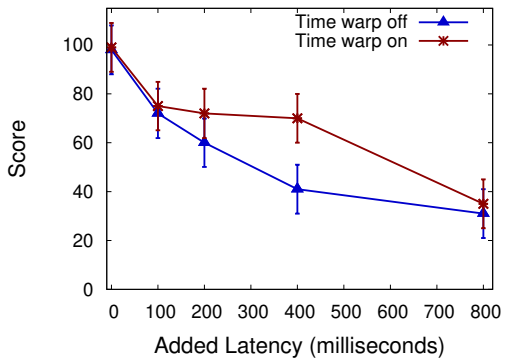

Fig. 6. User game score versus added latency.

with time warp is noticeable, however, seen by the clearly higher blue trend line in Figure 8. The absolute difference in consistency between time warp on and time warp off stays about the same (1 point) for all latencies.

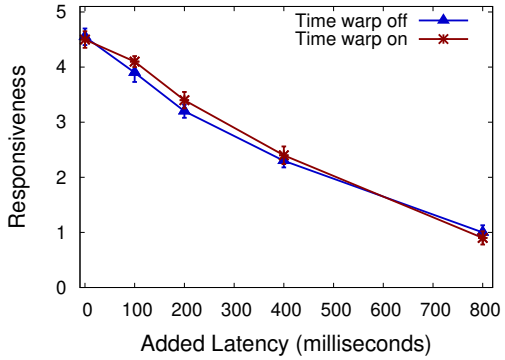

Fig. 7. Responsiveness versus added latency.

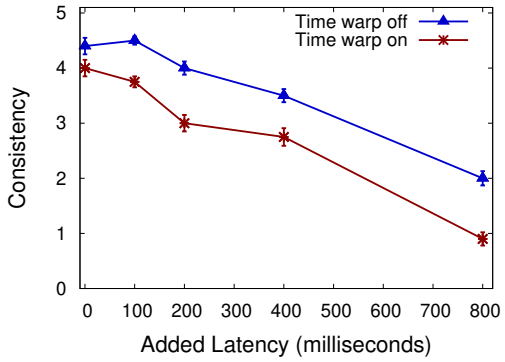

Fig. 8. Consistency versus added latency.

C. System Impact

Figure 9 depicts the average downstream (server to client) network bitrate (the y-axis) for different Drizzle streaming approaches. Each bar shown is the average bitrate measured over a complete Saucer Shoot game session. From the graph, image streaming is network intensive, needing almost 8 Mb/s. Video streaming has substantial bitrate savings, about 20% of that of image streaming. Graphics streaming has significantly reduced bitrates, about 20% that of video streaming and less than 5% that of image streaming.

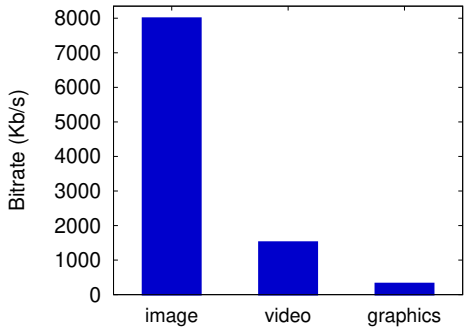

Fig. 9. Network bitrates for Drizzle streaming approaches for Cloud Saucer Shoot.

In order to provide a perspective on Drizzle bitrates, Table I compares Drizzle bitrates to a commercial cloud-based game service, traditional network games and video conferencing. From the table, traditional network games have the lowest network bitrates since the heavyweight client runs a full copy of the game as only game object updates need to be sent over the network. Drizzle image streaming has the highest bitrate, but not substantially higher than commercial cloud-based game streaming. Drizzle image streaming has bitrates around that of video conferencing. Drizzle graphics streaming has bitrates between video conferencing and traditional network games, but closer to the latter.

TABLE I. Network bitrate comparison

System	Bitrate (Kb/s)	Citation
Traditional network game	5 to 67	[16], [25], [26]
Drizzle graphics streaming	320	
Drizzle video streaming	1520	
Video conference	2222	
Commercial cloud-based game	6339	[1]
Drizzle image streaming	7950	

The game frames captured and sent by the server vary in size based on the game scene complexity. Game scenes with more game objects tend to be visually complex, not compressing as well for image and video streaming and requiring more commands for graphics streaming.

Figure 10 depicts the average network bitrate versus number of game objects for different Drizzle streaming approaches. The x-axis is the number of game objects and the y-axis is the network bitrate. Each point is the mean bitrate required for rendering a Cloud Saucer Shoot game frame with the indicated number of objects. From the graph, the bitrate requirements grow linearly with the number of objects. In all cases, image streaming has the highest bitrate by far, followed by video streaming and then graphics streaming.

Supporting games with a lot of game objects can make the performance bottleneck the server instead of the network.

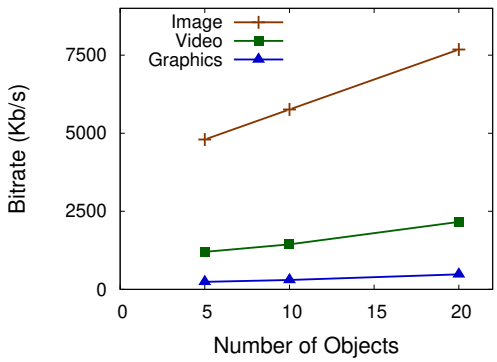

Fig. 10. Network bitrates versus number of game objects with trendlines for each Drizzle streaming approach.

Using the same computer as for our user study, we analyzed the CPU load for the Drizzle server for image streaming scenarios from Figure 11 with 200 milliseconds of added latency and time warp. The breakdown is as follows:

Time warp - each game loop (30 Hz), the server rolls back the game world 200 milliseconds to compensate for client latency, applies the user input, and rolls the world forward again.

Update - the server then updates the game world (moving game objects and resolving collisions).

Copy - the server copies the current game world, effectively replicating every game object in the game to preserve it for future time warping.

Stream - lastly, the server renders the game world and sends it to the client.

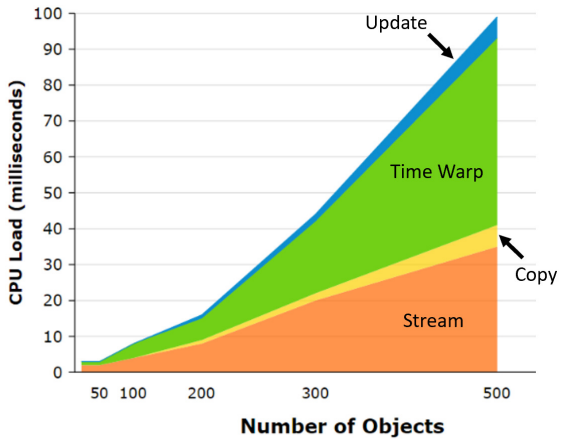

Fig. 11. CPU load breakdown versus number of game objects.

From the graph, as expected, total CPU load increases with number of game objects. Once the CPU load exceeds 33 milliseconds (at about 260 game objects), the game engine on can no longer keep up with the 30 Hz game loop rate. Under these conditions, the player experience would degrade from a reduced frame rate and overall sluggish performance. Time warp and streaming have the highest processing load by far. Many traditional game servers have latency compensation, but streaming adds an additional processing overhead unique to cloud-based game servers. This suggests the processing requirements for cloud-based servers are significantly higher than that of traditional game servers. Future work might look to optimize the processing of streaming as well as latency compensation.

V. Conclusion

The growth in games and networking has provided the opportunity for cloud-based games, where the server handles most of the game processing and rendering, streaming game frames to the lightweight client that primarily gathers and transmits player input. While cloud-based games have some advantages over traditional games, the remote processing of gameplay presents challenges in accommodating latency and network bitrate requirements.

This paper presents *Drizzle*, a lightweight cloud-based game system that allows for the study of latency compensation and game streaming approaches. Drizzle is written in C++ using the Dragonfly [5] game engine, adding a networking component and a lightweight client for full cloud-based game functionality.

Addressing network bitrates, Drizzle is used to evaluate different cloud-based streaming approaches, comparing the bitrates required by image streaming, video streaming and graphics streaming. Results from our systems experiments show video streaming, the state of the art for most commercial systems, provides a significant bitrate reduction over image streaming but graphics streaming can reduce bitrates even more.

Addressing latency, Drizzle is used to evaluate a well-known (but not well-evaluated) latency compensation technique – time warp – wherein the game server rolls back time to when the client provided input in order to accommodate the server to client latency. While time warp is often used by traditional game servers, e.g., Overwatch [24] (Blizzard, 2016), it has not been scientifically evaluated much nor has it been applied to cloud-based games. Results from our 30-participant user study show time warp can mitigate some of the effects of latency in terms of player performance, but time warp with projectile weapons has a noticeably more inconsistent game state. Analysis of CPU load shows time warp and streaming dominate, suggesting cloud-based game servers need more resources than traditional game servers.

Since time warp in cloud-based games was studied for projectile weapons in this paper, future work could evaluate time warp for hit scan (i.e., instant effect) weapons. Other latency compensation techniques such as aim assistance or time delay, could be evaluated in a cloud-based game system.

While Drizzle shows graphics streaming has potential to reduce bitrates more than video streaming, future work could apply graphics streaming techniques to systems other than Drizzle and explore the benefits for a wider range of games and game conditions.

REFERENCES

[1] M. Claypool, D. Finkel, A. Grant, and M. Solano, "On the Performance of OnLive Thin Client Games," *Springer Multimedia Systems Journal (MMSJ) - Special Issue on NetGames*, vol. 20, no. 5, 2014.

[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. 26, no. 2, Feb. 2014.

[3] D. D. Winter, P. Simoens, L. Deboosere, F. D. Turck, J. Moreau, B. Dhoedt, and P. Demeester, "A Hybrid Thin-client Protocol for Multimedia Streaming and Interactive Gaming Applications," in *Proceedings of NOSSDAV*, Newport, RI, USA, Jun. 2006.

[4] Y. W. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," in *Proceedings of GDC*, San Francisco, CA, USA, Feb. 2001, https://tinyurl.com/yan2yvs2 (accessed 1-17-2019).

[5] M. Claypool, *Dragonfly - Program a Game Engine from Scratch*. Worcester, MA, USA: Interactive Media and Game Development, Worcester Polytechnic Institute, 2014, online at: http://dragonfly.wpi.edu/book/.

[6] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Proceedings of Grid Computing Environments Workshop (GCE)*, Austin, TX, USA, Nov. 2008, pp. 1–10.

[7] C. Huang, C. Hsu, Y. Chang, and K. Chen, "GamingAnywhere: An Open Cloud Gaming System," in *Proceedings of ACM Multimedia Systems (MMSys)*, Oslo, Norway, Feb. 2013.

[8] R. Shea, L. Jiangchuan, E. Ngai, and Y. Cui, "Cloud Gaming: Architecture and Performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, Jul-Aug 2013.

[9] I. Slivar, M. Suznjevic, and L. Skorin-Kapov, "Game Categorization for Deriving QoE-Driven Video Encoding Configuration Strategies for Cloud Gaming," *ACM Transactions on Multimedia Computuing Communications and Applications*, vol. 14, no. 3s, pp. 56:1–56:24, Jun. 2018.

[10] R. Shea, J. Liu, E. Ngai, and Y. Cui, "Cloud Gaming: Architecture and Performance," *IEEE Network*, vol. 27, pp. 16–21, Aug. 2013.

[11] T. Fritsch, H. Ritter, and J. H. Schiller, "The Effect of Latency and Network Limitations on MMORPGs: a Field Study of Everquest 2," in *Proceedings of NetGames*, Hawthorne, NY, USA, Oct. 2005.

[12] G. Armitage, "An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3," in *Proceedings of the 11th IEEE International Conference on Networks (ICON)*, Sydney, Australia, Sep. 2003.

[13] K. Chen, P. Haung, G. Wang, C. Huang, and C. Lee, "On the Sensitivity of Online Game Playing Time to Network QoS," in *Proceedings of IEEE Infocom*, Barcelona, Spain, Apr. 2006.

[14] R. Amin, F. Jackson, J. E. Gilbert, J. Martin, and T. Shaw, "Assessing the Impact of Latency and Jitter on the Perceived Quality of Call of Duty Modern Warfare 2," in *Proceedings of HCI – Users and Contexts of Use*, Las Vegas, NV, USA, Jul. 2013, pp. 97–106.

[15] O. Hohlfeld, H. Fiedler, E. Pujol, and D. Guse, "Insensitivity to Network Delay: Minecraft Gaming Experience of Casual Gamers," in *Proceedings of the International Teletraffic Congress (ITC)*, Wurzburg, Germany, Sep. 2016.

[16] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The Effect of Latency on User Performance in Warcraft III," in *Proceedings of NetGames*, Redwood City, CA, USA, May 2003.

[17] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hossfeld, "An Evaluation of QoE in Cloud Gaming Based on Subjective Tests," in *Proceedings of Innovative Mobile and Internet Services in Ubiquitous Computing*, Seoul, Korea, 2011.

[18] M. Claypool and D. Finkel, "The Effects of Latency on Player Performance in Cloud-based Games," in *Proceedings of NetGames*, Nagoya, Japan, Dec. 2014.

[19] A. Sackl, R. Schatz, T. Hossfeld, F. Metzger, D. Lister, and R. Irmer, "QoE Management Made Uneasy: The Case of Cloud Gaming," in *IEEE Conference on Communications Workshops (ICC)*, Kuala Lumpur, Malaysia, May 2016.

[20] Y. W. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," in *Proceedings of the Game Developers Conference*, San Francisco, CA, USA, Feb. 2001, [Online] https://www.gamedevs.org/uploads/latency-compensation-in-client-server-protocols.pdf.

[21] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe, "Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games," in *Proceedings of the Conference on Human Factors in Computing Systems*, Seoul, Korea, 2015.

[22] W.-K. Lee and R. K. C. Chang, "Evaluation of Lag-related Configurations in First-person Shooter Games," in *Proceedings of NetGames*, Zagreb, Croatia, 2015.

[23] S. W. K. Lee and R. K. C. Chang, "On 'Shot Around a Corner' in First-Person Shooter Games," in *Proceedings of NetGames*, Jun. 2017.

[24] T. Ford and P. Orwig, "Developer Update - Let's Talk Netcode," Online, apr 2016, https://www.youtube.com/watch?v=vTH2ZPgYujQ (accessed: 1-17-2018).

[25] J. Nichols and M. Claypool, "The Effects of Latency on Online Madden NFL Football," in *Proceedings of NOSSDAV*, Kinsale, County Cork, Ireland, Jun. 2004.

[26] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," in *Proceedings of NetGames*, Portland, OG, USA, Sep. 2004.