WPI-CS-TR-20-01 May 2020

Online Games and Network Latency Compensation Techniques

by

Shengmei Liu

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Online Games and Network Latency Compensation Techniques
Research Qualifier Report

Shengmei Liu

ABSTRACT

The growth in networking and cloud services provides opportunities to host multimedia on remote servers, but also brings challenges to developers who must deal with added delays that degrade interactivity. Latency compensation techniques are developed to reduce the impact of latency on players. Analytic models and simulations have the potential to enable exploration of the space of player game performance with delay using fewer resources than time-intensive user studies. However, in order to be effective, such models and simulations need accurate representations of user performance for core game actions. This paper presents an analytic model for the distributions of player elapsed times for a common game task – selecting a moving target with a mouse with delay – derived from results from prior user studies. We demonstrate use of our model in simulation, exploring player performance in games with several different latency compensation techniques, configurations and delays.

1. INTRODUCTION

Computer games require timely responses to player actions in order to provide an immersive interactive experience. Unfortunately, computer input hardware and software always has some delay from when a player inputs a game command until the result is processed and rendered on the screen. Delays on the local computer system are still at least 20 milliseconds and can range much higher, to about 250 milliseconds for some platforms and game systems [18]. Games played over a network, such as multi-player game systems or cloud-based games, have additional delays from the network processing as game data has to be transmitted to and from a server. There have been many studies of latency and games [7, 9–11, 13, 16, 17, 19, 24, 25, 27] that show that both local delays and network delays impact the player, degrading both player performance and player Quality of Experience (QoE) as total delay increases [4, 5, 8].

Some of the limitations of latency are fundamental, but progress can be made through the deployment of latency compensation techniques [3]. Latency compensation techniques attempt to mitigate the effects of network delay. The techniques trade-off consistency for better responsiveness. For example, for intense shooting games, techniques that improve responsiveness, like time warp, are used. For some slower RTS games, techniques that improve consistency, like bucket synchronization, are used. There are many existing compensation techniques which are deployed on the client, the server, or both. Since the server-side techniques play a dominant role in online gaming, we focus on two server-side techniques - time warp and bucket synchronization.

To evaluate latency compensation techniques, we build models from previous user studies [10]. The users studies measured user performance for an atomic game action – selecting a moving target with a mouse. Users played a game that had them select targets that moved around the screen with different speeds and with different amounts of added delay. The studies recorded the elapsed time to select the target and the number of clicks, coupled with a player-provided self-rating of skill.

We use the data from these studies in multivariate, multiple regression to derive novel models of the expected elapsed times for selecting a moving target with a mouse, and the *distribution* of the elapsed times. Used together, the models provide an accurate representation of target selection times over a range of delays and target speeds and two levels of player skill. We demonstrate the use of the models in analytic models of player performance and simulations of basic shooting games to predict the impact of delay and compensation techniques for several game and system configurations.

We propose a novel representation of the consistency-responsiveness space to evaluate latency compensation techniques across game configurations. One axis is inconsistency measured by the differences in the game world across players, and the other axis is unresponsiveness measured by the input delay. We can use our model and simulations to quantify the responsiveness and inconsistency for a given game and system configuration. The closer the compensation places the configuration to the left bottom corner, the better the QoE.

The rest of this paper is organized as follows: Section 2 introduces latency compensation techniques. Section 3 describes the methodology exploring gaming and latency spaces. Section 4 presents prior user study data. Section 5 pre-processes data before modeling. Section 6 depicts the process of finding and building the most parsimonious model. Section 7 depicts how player skills are clustered, and the process of building models with skill as a parameter. Section 8 includes multiple simulations on games, delay, and compensation techniques. Section 9 proposes a novel model to measure compensation techniques. Section 10 summarizes and concludes. And Section 11 provides future work.

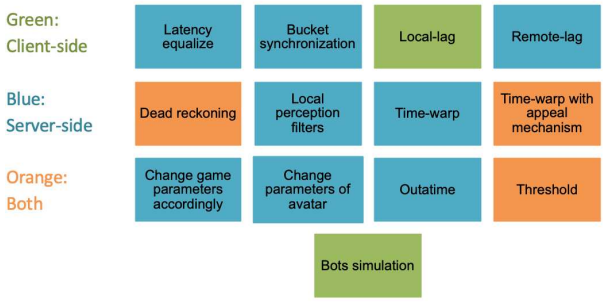

FIG. 1. Compensation techniques

2. LATENCY COMPENSATION TECHNIQUES

Latency compensation techniques are designed to mitigate the negative effects of network delay. We classify existing techniques by where they are applied in the client-server architecture. Figure 1 includes thirteen existing compensation techniques. The green are deployed on client, the blue are deployed on server and the orange are deployed on both client and server.

A. Client-side techniques

With *bots simulation* [15], the client stores data on opponents. When high latency is detected, bots replace real players to play the game, and actual states are synchronized later. This technique is only suitable for multiple player games with a few players. This technique claims it can conceal jitter seamlessly, but can cause information exposure because of states sharing, which can lead to cheating. Moreover, It requires extensive computation and extra memory for the client. If the latency persists, players compete against the bots not humans , which is not most of players' purpose for playing multiple player games.

Local-lag [23] delays the execution of local commands long enough that the commands have time to propagate to all remote sites and can then be executed simultaneously at all locations. For example, if a player presses the "W" key to move forward, the avatar does not move until after the delay period, providing time for the input to be sent over the network to remote game clients. Programming local lag requires mechanisms for delaying inputs and for estimating message delivery time between the different nodes. Delayed input algorithms improve consistency, at the cost of slower response to player actions.

B. Server-side techniques

With *bucket synchronization* [14], the game server puts all actions over a period in a buffer. When all actions in the bucket are to be processed, the server updates states for all players. This technique equalises latency between players, and is useful in games requiring high consistency but low latency sensitivity, like RTS games. Players share the same view of the game world. It's not effective in games with many players, since players with high latency can degrade the gaming experience of other players.

With *remote-lag* [3], each client applies a constant lag to the actions of other players and the server arbitrates hit decisions based on the state of the shooter's client at the time the shot was made. This means that the shooter can aim directly at the target. For example, in the Half-Life first person shooter game, remote players' avatars are lagged by a constant 100ms. It increases inconsistency (since the remote state updates are not applied immediately), but makes the degree of consistency predictable, something to which players may be able to adapt.

Local perception filters [28] continually adjust the amount of delay applied to non-player controlled entities depending on their positions. The nearer the entity to local players, the closer the entity to the current state. Players can directly interact with entities close to them. However, this technique is complicated to implement. Besides, high delay jitter may cause entities to move back and forth. These filters cannot be applied to direct interactions between players (like melee) and can cause inconsistency in that the players have different views of the game world.

With *time-warp* [23], when the server gets a player action, the server rolls the state back to the time when the action occurred on client, and resolves the game action. Players can directly act on their visible game world. There is no need for the client to extrapolate game states. However, time warp is known to cause 'shot around corner' problems. In a shooter, a player with lower latency who has hidden behind a cover or a corner could be rolled out of the cover. The game world is not consistent between the players. Also, the server may need large storage and processing for the game states corresponding to the timestamps.

Outatime [6] combines input prediction with a Markov model on the server, and game world correction with view interpolation and bandwidth compression by sharing information. However, extensive calculations are needed at the server.

Change game parameters [26] changes the game parameters according to precision-deadline model proposed by Claypool and Claypool [8], and adjusts difficulty levels. Players with high latency can perform similar to players with low latency. However, it does not work in games with a world view shared by multiple players. Players still act on their predicted game state, but it may affect game fairness.

Change parameters of avatar [18] changes parameters of a player's avatar, like size, velocity, or parameters in the aim assistant function. This technique can mitigate the effects of latency. However, it only works in games

with an avatar. Players still have to act on their predicted game state, but it may affect game fairness.

C. Client and Server techniques

Time warp with appeal mechanism is a refinement of time warp proposed [20] to avoid 'shot around corner'. Once players get shot, they can appeal the shot. If the client approves the appeal and denies the shot, the authoritative server makes a final decision. This technique can mitigate the 'shot around the corner' problem of time warp. However, the client involvement in the decision can lead to cheating. Also this technique is only useful for shooting games, and maybe ineffective for high latency situations.

Threshold [20] is a refinement to existing lag compensation techniques adding a threshold for applying compensation, which is normally 250 ms. It can be combined with many other compensation techniques. It is useful in games where the original compensation technique brings negative effects to lower latency players. For example, time warp is known to cause 'shot around corner' problems for lower latency players. Thresholds have been applied in FPS games like Overwatch [20], effectively reducing the negative effect of compensation techniques on lower latency players. However, players with high latency over 250ms are not compensated and may not able to play the game.

Dead-reckoning [2] predicts changes to remote states before those changes are propagated over the network. The client extrapolates the current state of an object by its velocity and direction. If the predicted position is wrong, there could be 3 ways of correction: 1. No correction, 2. Immediate correction - the object is moved to the correct position immediately, 3. Smooth correction, the object is moved to the correct position smoothly. This technique is suitable for games with stable target velocity and direction, but may not perform well in some fast action games where players move unpredictably.

3. METHODOLOGY

There are two primary ways exploring the effects of latency on players – user study and simulation. A user study obtains real user data, and the data can be used to build models for the base of many simulations. Simulation allows for thorough exploration of a wide range of latencies, which is unrealistic for user studies. Simulation can also help decide an area of focus for a full user study. Hence, my approach is to build models with data from the existing user studies, then run simulations of latency compensation techniques and games based on the models.

Our methodology is:

1. Select prior user study data (Section 4)
2. Pre-process data before modeling (Section 5)
3. Find and build the most parsimonious model for the selection time distribution using the data (Section 6)
4. Cluster player skill, and refine model with player skill as a term (Section 7)
5. Build simulations and analytic models of games, delay, and compensation techniques using the models (Section 8)

4. DATASETS

We use two sets of data, Set-A and Set-B, obtained from prior user studies [10]. Each data set was obtained from users playing a game with controlled amounts of delay where the game focused on a single player's action – selecting a moving target with a mouse. Selecting a moving target is an action common to many PC game genres. Some examples include: 1) top-down shooters (e.g., *Nuclear Throne*, Vlambeer, 2015) where the player aims a projectile at opponents by moving the mouse to the intended target; 2) first person shooters (FPS) (e.g., *Call of Duty*, Activision, 2003) where players use the mouse to pan the game world to align a reticle over a moving opponent and shoot; and 3) multiplayer online battle arenas (MOBAs) (e.g., *League of Legends*, Riot Games, 2009) where players move a skill shot indicator with a mouse to target a moving opponent with a spell.

A. Game

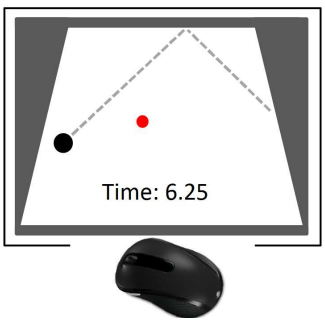

FIG. 2. *Puck Hunt.* Users click on a moving target (the puck) with the mouse cursor (a red ball). The game adds delay to the mouse input and varies the target speed between rounds.

Both data sets are obtained from users playing a custom game called *Puck Hunt* that allows for the study of a moving target selection with controlled amounts of delay. In Puck Hunt, depicted in Figure 2, the user proceeds through a series of short rounds, where each round has a large black ball, the puck/target, that moves with kinematics, bouncing off the edges of the screen. The user

moves the mouse to control the small red ball (a.k.a., the cursor) and attempts to select the target by moving the ball over the target and clicking the mouse button. Once the user has successfully selected the target, the target disappears and a notification pops up telling the user to prepare for the next round. Thereupon pressing any key, a new round starts, with the target at a new starting location with a new orientation and speed. The user is scored via a timer that counts up from zero at the beginning of each round, stopping when the target is selected. The target is a constant 28 mm.

In dataset Set-A, users select targets with three different speeds (150, 300 and 450 pixels/s) under 11 different delays (100, 125, 150, 175, 200, 225, 250, 275, 300, 400, and 500 ms), with each combination of delay and speed encountered 5 times.

In dataset Set-B, users select targets with three different speeds (550, 1100 and 1550 pixels/s) under 11 different delays (20, 45, 70, 95, 120, 145, 160, 195, 220, 320, and 420 ms), with each combination of delay and speed encountered 5 times.

For both Set-A and Set-B, users played Puck Hunt with a mouse.

Objective measures of performance recorded are the elapsed time to select the target and the number of clicks required to do so.

B. Procedure

All user studies were conducted in dedicated computer labs with LCD monitors and computer hardware more than adequate to support the games.

For each study, participants first completed informed consent and demographic questionnaire forms before starting the game. The demographic questionnaire include the question "rate yourself as a computer gamer" with responses given on a 5 point scale (1 - low to 5 - high). The self-rating question was mandatory. The demographic questionnaire also included a gender question with options for "male", "female", "other" and "prefer not to say" – only one user did not specify either male or female.

TABLE 1. Summary of dataset variables

Dataset	Usrs	Gender	Rounds	Conditions
Set-A	51	43 ♂, 8 ♀	167	3 speeds, 11 delays
Set-B	32	23 ♂, 8 ♀, 1 ?	167	3 speeds, 11 delays
Combined	83	66 ♂, 16 ♀, 1 ?	167	6 speeds, 22 delays

Table 1 summarizes the major dataset variables, with the bottom row, "Combined", showing the users, gender and rounds of both datasets combined into one.

Table 2 shows the breakdown of self-rated skills for each dataset, with the mean and standard deviation reported by $\bar{x}$ and s in the last two columns, respectively. The bottom row shows the breakdown of both datasets combined into one. The datasets have a slight skew towards higher skills (mean skill slightly above 3 and mode 4 for each) but there are players of all skill levels in both sets.

TABLE 2. Dataset breakdown of self-rated skills

	Self-rated skill						
Dataset	1	2	3	4	5	$\bar{x}$	s
Set-A	1	3	5	24	18	4.1	0.9
Set-B	4	2	9	8	9	3.5	1.3
Combined	5	5	14	32	27	3.9	1.1

5. PREPROCESSING AND STANDARDIZATION

Before modeling, the user study data needs to be preprocessed and standardized.

A. Preprocessing

In Puck Hunt, if the user's time to select the target surpasses 30 seconds, the round ends, and the elapsed time for that round is recorded as a 30. These 30 second values are not the elapsed times that would have been recorded if the trial continued, and so artificially impact any model of selection time that includes them. Thus, we look to replace values of exactly 30 seconds with estimates of the larger values they likely would have had if the round had continued (and the user had kept trying) until the target was selected.

In total, the game has 66 different combinations of speed and delay, called *difficulty levels*. For most difficulty levels, there are no elapsed times of 30 seconds. However, the higher difficulty levels (speeds above 500 px/s, and delay above 120 ms) have one or more 30 second values.

TABLE 3. Elapsed times above 30 seconds

Speed	Delay (ms)					
(px/s)	145	170	195	220	320	420
550	0 -	0 -	0 -	0 -	1 (0.4%)	*5* *(2.0%)*
1100	2 (0.8%)	1 (0.4%)	3 (1.2%)	2 (0.8%)	*18* *(7.1%)*	**56** **(22.0%)**
1550	3 (1.2%)	3 (1.2%)	*8* *(3.1%)*	**12** **(4.7%)**	**70** **(27.5%)**	**109** **(42.8%)**

Table 3 includes the number and percent of trials with elapsed times recorded as 30 seconds for high difficulty levels. All difficulties not in the table (i.e., easier levels)

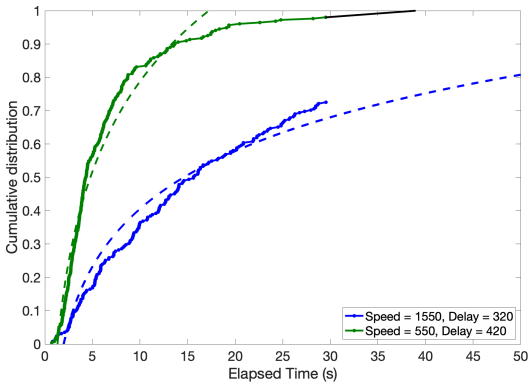

FIG. 3. Example above 30 data

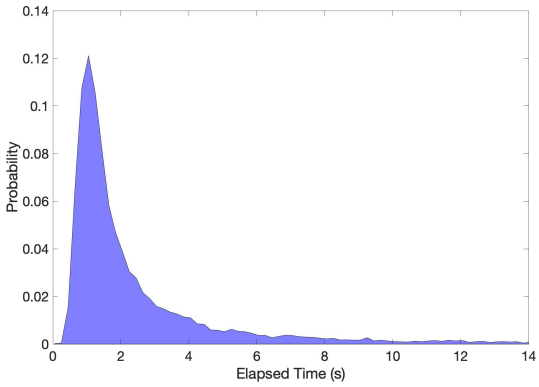

FIG. 4. Probability density distribution of elapsed time

have no values recorded as 30. For the numbers with a normal font, only about 1% (or fewer) of the times are a 30, so we leave them as is.

TABLE 4. Analytic models for CDFs for speed & delay combinations not including failures.

Speed	Delay	Equation	R^2	Mean
550	420	$0.39 \cdot ln(x) - 0.11$	0.96	6.2
1100	320	$0.34 \cdot ln(x) - 0.09$	0.98	8.2
1100	420	$0.27 \cdot ln(x) - 0.20$	0.94	23.1
1550	195	$0.29 \cdot ln(x) + 0.05$	1.00	7.4
1550	220	$0.30 \cdot ln(x) - 0.07$	0.97	10.3
1550	320	$0.25 \cdot ln(x) - 0.17$	0.94	30.0
1550	420	$0.21 \cdot ln(x) - 0.22$	0.93	66.8

For the difficulty levels in italics and bold, we estimate the elapsed times as if they were not artificially constrained, and generate synthetic points. Figure 3 depicts CDFs of the empirical data, along with previously derived models at several example difficulty levels to illustrate our methods of generating elapsed times larger than 30. The x-axis is the elapsed time and the y-axis is the cumulative distribution. The solid lines are CDFs of the empirical data, and the dashed lines with the same color are the corresponding models listed in Table 4. For the 4 difficulty levels with bold numbers in Table 3, we replace the original data with randomly generated points above 30 seconds using the models. For the 3 difficulty levels in italics, the corresponding CDF model does not surpass 30 seconds. For these difficulty levels, we model the tail of the distribution as a linear regression using the last 5 data points, and extend the CDF to 1 on the y-axis. We replace the original data with points evenly distributed on those extension lines.

B. Standardization

Before modeling, we standardized the mean and speed by subtracting the means and dividing by the standard deviations in Table 5.

TABLE 5. Values used for standardization

Factor	Mean	Std Dev
Delay (D)	206 ms	122
Speed (S)	683 px/s	488

6. MODELING

This section describes our methods used to process and analyze the user study data to derive models for the distribution of elapsed times to select moving targets with delay.

Figure 4 shows the probability distribution of all elapsed times from the user studies. The x-axis is the elapsed time and the y-axis is the probability density. The blue region is the distribution of all elapsed times, and appears log-normal, which makes sense since human actions are impacted by many individual factors that, when put together, have an exponential distribution.

A. Gamma distribution fitting

The gamma distribution is a two-parameter family of continuous probability distributions. Some gamma distributions can be visually similar to the right-skewed distribution of the elapsed time of all the data. Figure 5 shows how the gamma function fits for the elapsed time. The x-axis is the elapsed time in seconds, and the y-axis is the probability density. The blue shape is the probability distribution of all elapsed times. The red shape is generated from the gamma function fitted for the elapsed

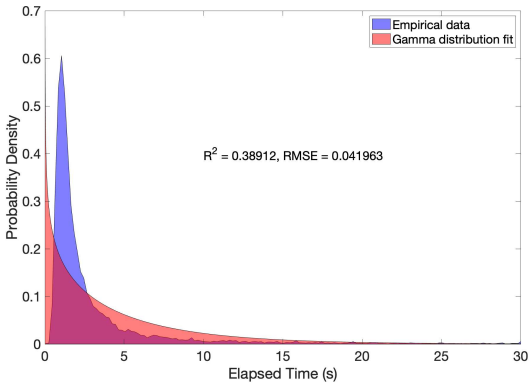

FIG. 5. Modeled data with gamma distribution and empirical data

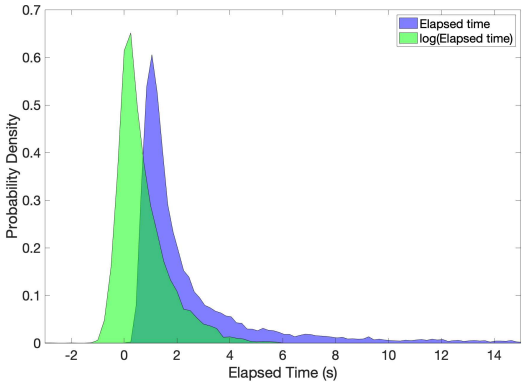

FIG. 6. Probability distribution of elapsed time

time. The red shape does not fit the blue shape well due to the long tail of elapsed time, with R^2 at 0.39 and RMSE at 0.04. This suggests that a gamma distribution is not the right model for the data.

B. Normal fitting

1. *Transformation*

Prior literature points out that human response time are right skewed and log-normal [12][21] . Ma et al. [22] fit human response time with a generalized inverse gamma (GIGa) function that belongs to a family of distributions. They argue that the family, which includes log-normal, is the most appropriate for modeling human reaction times. Action time is an important factor of moving target selection. Thus we take the logarithm of the elapsed time to obtain a probability distribution in the green region. It appears less right skewed than the empirical data as indicated in Figure 6.

Figure 7 depicts how normal distribution fits for natural log of elapsed time. The x-axis is elapsed time and

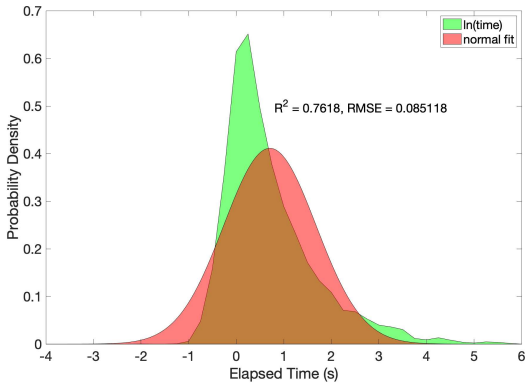

FIG. 7. Probability distribution of elapsed time

the y-axis is probability density. The green region is the distribution of the natural log of the elapsed time, and red region is the distribution of normal fit of the natural log of the elapsed time. The normal distribution fits better than the gamma function, with R^2 at 0.76 and RMSE at 0.09. However, it is still not the ideal model that can be used for simulation.

The low R^2 of normal fit, and the significant skewness of the natural log of the elapsed time are likely caused by the combination of many low difficulties and a few high difficulties.

2. *Skewness measurement*

We measure the skewness of the elapsed time distribution. If skewness of a distribution is less than -1 or greater than 1, the distribution is highly skewed. If skewness is between -1 and -0.5 or between 0.5 and 1, the distribution is moderately skewed. If skewness is between -0.5 and 0.5, the distribution is approximately symmetric [1]. The skewness of the natural log of the elapsed time equals to 1.45 indicates that distribution of ln(elapsed time) (the green shape in Figure 6) is significant highly skewed.

However, when the data is divided into groups for each difficulty, the data is mostly symmetric. Skewnesses of Set-A difficulties are included in Table 6. The skewness for most difficulties are between 0 and 0.6, which indicates that most difficulties in Set-A have approximately symmetric or at most moderately skewed distributions. Skewnesses of Set-B difficulties are included in Table 7. The skewness for most difficulties are between -0.2 and 0.5, which indicates that most difficulties in Set-B dataset have approximately symmetric distributions.

3. *Kolmogorov-Smirnov test (K-S) and Shapiro-Wilk test (S-W)*

In statistics, the Kolmogorov–Smirnov(K-S) test is a nonparametric test of the equality of continuous, one-

TABLE 6. Skewness of Set-A difficulties

	delays										
Speed	20	45	70	95	120	145	170	195	220	320	420
550	0.35	0.25	0.66	0.35	0.59	0.31	0.54	0.20	0.51	0.56	0.39
1100	0.58	0.41	0.29	0.42	0.52	0.62	0.58	0.46	0.27	0.17	-0.84
1550	0.31	0.24	0.37	0.03	0.27	0.24	0.14	-0.01	-0.36	-0.88	-1.36

TABLE 7. Skewness of Set-B difficulties

	delays										
Speed	100	125	150	175	200	225	250	275	300	400	500
150	0.12	0.06	-0.02	0.08	0.22	-0.16	0.30	-0.03	0.43	-0.08	0.02
300	0.52	0.31	0.03	0.01	-0.18	0.52	0.37	-1.30	0.21	0.40	0.42
450	0.24	0.95	0.55	0.12	0.01	0.35	0.54	0.22	1.33	-1.50	-0.02

dimensional probability distributions that can be used to compare a sample with a reference probability distribution [30]. If the resulting p value is less than 0.05, it indicates that there is significant difference between sample distribution and reference probability. Otherwise there is no statistically significant difference between the two distributions. The Shapiro–Wilk(S-W) test tests the null hypothesis that a sample x1, ..., xn came from a normally distributed population. If the test is not significant, this indicates the data are normal. Both of the tests can be used to test normality. It has been suggested that if the sample size is less than 50, S-W test is more persuasive than the K-S test. Otherwise, the K-S test is more suitable. When the sample size is greater than 1500, the S-W test is not reasonable anymore. For each difficulty in Set-A, there are 255 trails. For each difficulty in Set-B, there are 160 trails. Thus, we use the K-S test to test normality. (For completeness, the results of the S-W tests are included in the appendix).

All elapsed time are transformed and standardized for each difficulty. We then conduct K-S tests on the resulting data values. The results of the K-S tests for most difficulties in both datasets are not significant, indicating that the distributions of the natural log of the elapsed times for most difficulty levels are approximate normal.

Results of the K-S test of the Set-A difficulties are included in Table 8. Results of the K-S test of the Set-B difficulties are included in Table 9.

For most difficulties levels, the distribution of the natural log of the elapsed time is approximately symmetric or at most moderately skewed indicated by the skewness value, and there is no significant difference with a normal distribution indicated by K-S test results. Hence this suggests that the probability distributions of the natural log of the elapsed time at most difficulty levels can be approximated by a normal distribution.

4. Example difficulty fitting

To better illustrate how the models fit for data at specific difficulties, we pick one difficulty (speed=150, delay= 150) and fit models on the data.

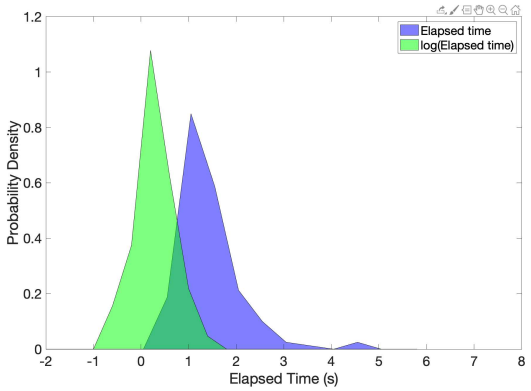

FIG. 8. PDF for elapsed time and natural log of elapsed time with speed= 150, delay=150

Figure 8 depicts the probability density distribution of elapsed time and the probability density distribution of the natural log of elapsed time, both with speed at 150 px/s and delay 150 ms. The x-axis is the elapsed time in seconds and the y-axis is the probability density. The blue shape is the probability density distribution of elapsed time, and the green shape is the probability density distribution of the natural log of elapsed time. The skewness of the green shape is smaller than the skewness of the green shape in Figure 5.

Figure 9 depicts a gamma distribution fit for the elapsed time with speed at 150 px/s and delay at 150 ms. The x-axis is the elapsed time in seconds at the difficulty, and the y-axis is the probability density at the difficulty. The blue shape is the probability density distribution of all elapsed times at the difficulty. The red shape is the gamma function fitted for the elapsed time. The gamma distribution fits the empirical data well with R^2 at 0.97 and RMSE at 0.04.

Figure 7 depicts a normal distribution fit for the natural log of the elapsed time with speed at 150 px/s and delay at 150 ms. The x-axis is elapsed time and the y-axis is probability density. The green region is the dis-

TABLE 8. K-S test of Set-A difficulties

	delays										
Speed	20	45	70	95	120	145	170	195	220	320	420
550	0.59	0.82	0.13	0.59	0.07	0.78	0.34	0.47	0.78	0.21	0.07
1100	0.16	0.43	0.69	0.29	0.27	0.29	0.07	0.29	0.84	0.44	**<0.001**
1550	0.40	0.95	0.16	0.97	0.46	0.76	0.84	0.19	0.32	**<0.001**	**<0.001**

TABLE 9. K-S test of Set-B difficulties

	delays										
Speed	100	125	150	175	200	225	250	275	300	400	500
150	0.56	0.17	0.96	0.70	0.60	0.82	0.68	0.68	0.52	0.58	0.69
300	0.25	0.56	0.80	0.74	0.68	0.07	0.67	0.27	0.84	0.23	0.33
450	0.51	0.20	0.34	0.78	0.94	0.34	0.13	0.62	**0.03**	0.26	0.94

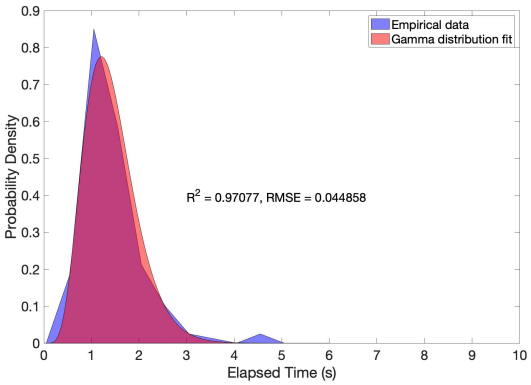

FIG. 9. Gamma fit for elapsed time with speed= 150, delay= 150

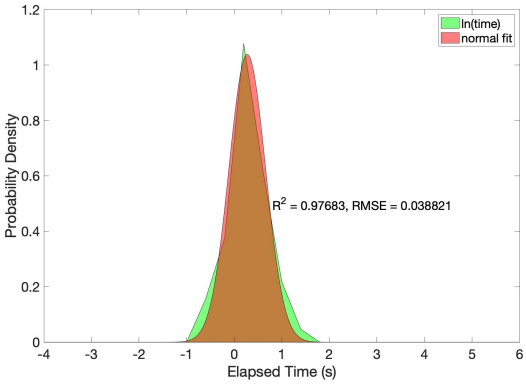

FIG. 10. Normal fit for the natural log of elapsed time with speed= 150, delay= 150

tribution of the natural log of the elapsed time at the difficulty, and red region is the distribution of normal fit of the natural log of the elapsed time at the difficulty. The normal distribution fits the data well with R^2 at 0.98 and RMSE at 0.04, which is a little bit better than the gamma distribution.

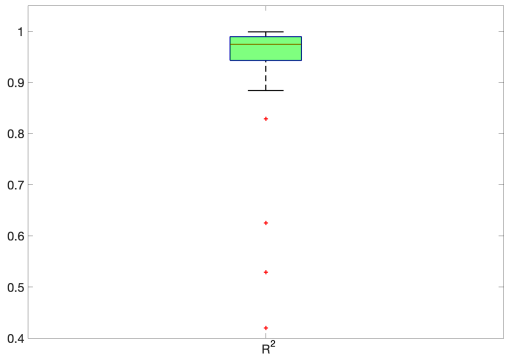

FIG. 11. Boxplot of R^2 for all difficulties

Figure 11 is the boxplot for the R^2 value of the normal fit for the 66 difficulties in dataset Set-A and Set-B. The mean of the R^2 is 0.95 and median is 0.97. 75% of the R^2 values are above 0.93 with only few outliers at high difficulty. This figure indicates that the normal distribution fits well for the natural log of elapsed time with almost all difficulties.

The values of R^2 and RMSE for all difficulties are included in the appendix.

5. Modeling results

Inspired by the analysis above, we use multivariate, multiple regression to model the mean and standard deviation of the natural logarithm of the elapsed time for each difficulty level. This will allow us to generate distributions of elapsed times for a given difficulty level, making it usable for analytically modeling and for simulating game performance.

There are many possible models that fit the elapsed time data. We compare different regression models by using the coefficient of determination (R^2) as a measurement of how well observed outcomes are replicated by the

model. In doing such a comparison, it might be tempting to choose the model with the maximum R^2, but this can over-fit the model to the data (i.e., the R^2 value can be increased by adding more terms). However, the adjusted R^2 modifies the R^2 based on the number of predictors in the model, increasing if the new term improves the model more than would be expected by chance and decreasing it otherwise. Overall, we want a parsimonious model – one that provides the desired prediction with as few terms as possible.

Table 10 summarizes the models explored. For the equations, the k parameters (e.g., k_1) are constants, e is the base of the natural logarithm (2.7182), d is the standardized delay and s is the standardized speed, where $d = \frac{D-206}{122}$ and $s = \frac{S-683}{488}$.

While model 10 has the highest R^2 for both mean and standard deviation, it is only slightly higher than model 8 while having significantly more terms (9 versus 4). Thus, we propose predicting mean and standard deviation for the natural log of the elapsed time:

$$\begin{aligned} mean(ln\ T) &= k_1 + k_2 d + k_3 s + k_4 ds \\ stddev(ln\ T) &= k_5 + k_6 d + k_7 s + k_8 ds \end{aligned} \quad (1)$$

where d and s are standardized from the delay (D) and target speed (S), respectively: k_1 through k_8 are constants derived from data gathered through the user studies.

Using our standardized user study data in Table 5 yields an adjusted R^2 for mean and standard deviation both at 0.96. The final model for the mean and standard deviation of the natural log of elapsed time is:

$$\begin{aligned} mean(ln\ T) &= 0.685 + 0.506d + 0.605s + 0.196ds \\ stddev(ln\ T) &= 0.567 + 0.126d + 0.225s + 0.029ds \end{aligned} \quad (2)$$

With the models predicting mean and standard deviation for the natural log of the elapsed time, given speed and delay, the normal distribution with the predicted mean and standard deviation can be used to generate a distribution of logarithmic elapsed times, taking the exponent to get the elapsed time. Figure 12 shows the model generated data compared to the actual data. The x-axis is the elapsed time in seconds, and the y-axis is the cumulative distribution. The blue shape is the probability distribution of the elapsed time for the empirical data, and the red shape is generated from the normal distribution using the modeled mean and standard deviation from Equation 2. Our model has an excellent fit for the data with R^2 at 0.99 and RMSE at 0.03.

Figure 13 shows example CDFs of our model and data for speed at 1550 px/s and delay 70 ms (top) and speed 550 px/s and delay 170 ms (bottom). The blue lines are the empirical elapsed times from the user study data and the dashed black lines are the corresponding models at the respective difficulty. In all cases, the model fits the data well, with R^2 both at 0.98 and RMSE both at 0.04.

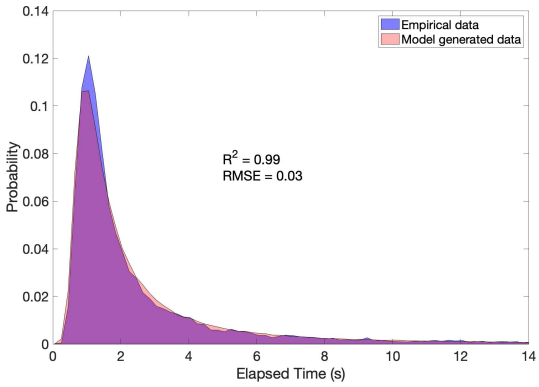

FIG. 12. PDF of modeled data and empirical data

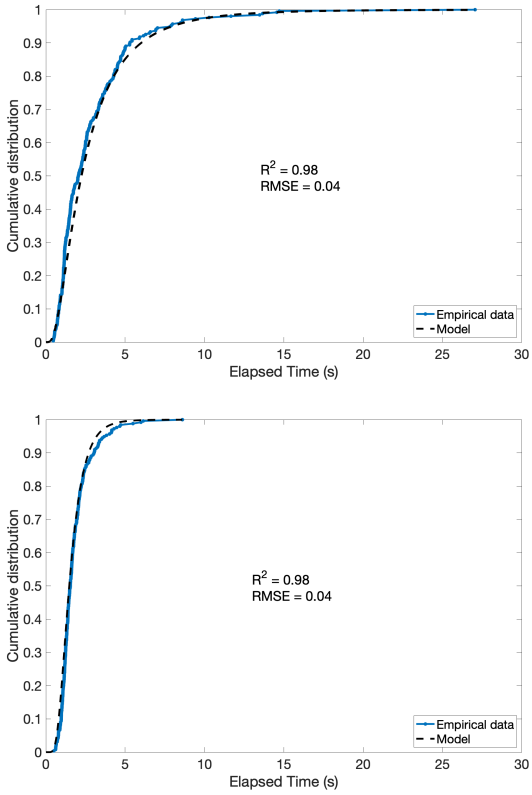

FIG. 13. Example CDF data and models. Top: speed 1550 px/s and delay 70 ms. Bottom: speed 550 px/s and delay 170 ms

7. MODEL WITH PLAYER SKILL

Player skill can be an important factor for analytical models and simulations for games. This section describes our methods to derive models for the distribution of elapsed times with player skill as a parameter.

TABLE 10. Models predicting mean and standard deviation of selection time of a moving target with delay (d) and speed (s)

	Model	Mean R^2	Std Dev R^2
1.	$k_1 + k_2 s$	0.41	0.67
2.	$k_1 + k_2 e^s$	0.39	0.61
3.	$k_1 + k_2 d$	0.23	0.07
4.	$k_1 + k_2 d + k_3 s$	0.89	0.94
5.	$k_1 + k_2 e^d$	0.18	0.07
6.	$k_1 + k_2 e^d + k_3 e^s$	0.68	0.79
7.	$k_1 + k_2 d + k_3 d^2 + k_4 s + k_5 s^2$	0.88	0.94
8.	$\mathbf{k_1 + k_2 d + k_3 s + k_4 ds}$	**0.96**	**0.96**
9.	$k_1 + k_2 e^d + k_3 e^s + k_4 e^d e^s$	0.84	0.83
10.	$k_1 + k_2 d + k_3 d^2 + k_4 s + k_5 s^2 + k_6 ds + k_7 d^2 s + k_8 ds^2 + k_9 d^2 s^2$	0.97	0.96

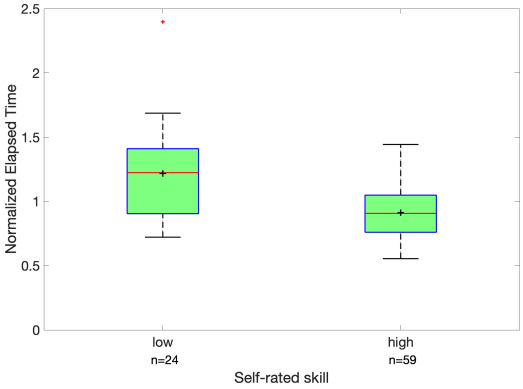

FIG. 14. Combined skill groups

A. Player skill

In the user studies, players rated their skills as computer gamers from 1 (low) to 5 (high). The mean self-rating was about 3.9, showing a slight skew to having "high ability". Based on our user sample, we divided users into low skill (24 users with self-rating 1-3) and high skill (59 users with self-rating 4-5).

The performance of each user is the average of their results across all trials in their user study (Set-A or Set-B). Since the games and tested conditions are slightly different between the two user studies, user results from one study cannot be directly compared (or combined) with the results from another. Hence, we normalize the data for each user study based on the average performance of all users in the same dataset. For example, since the average elapsed time to select a target across all users and all trials for the Set-B dataset is 1.6 seconds, each individual user in the Set-B dataset has their average elapsed time divided by 1.6. Users with normalized values below 1 are better than average and values above 1 are worse than average – e.g., a normalized score of 0.9 is 10% better than the average while a 2.0 is twice as bad as the average.

Figure 14 shows boxplots of normalized elapsed time on the y-axis for users clustered by skill group on the x-axis. Each box depicts quartiles and median with the mean shown with a '+'. Points higher or lower than 1.4 × the inter-quartile range are outliers, depicted by the dots. The whiskers span from the minimum non-outlier to the maximum non-outlier. The x-axis "n=" labels indicate the number of participants in each skill group.

From the figure, the mean and median of normalized elapsed times decrease (improve) with self-rated skill. The spread also indicates that some individuals with lower self-ratings perform better than users with higher self-ratings.

Since the elapsed time data was observed to be skewed right, comparisons of the two skill groups was done using a Mann-Whitney U test – a non-parametric test of the null hypothesis that it is equally likely that a randomly selected value from one group will be greater than or less than a randomly selected value from a second group. Effectively, this tests whether two independent self-rated skill group samples come from populations having the same distribution.

TABLE 11. Mann-Whitney U test for elapsed time by self-rated skill

Users		Median			
low	high	low	high	U	p value
24	59	1.22	0.91	321	**<.001**

Table 11 gives the results of the Mann-Whitney U tests. The "Users" and "Median" columns show the number of corresponding participants and median normalized elapsed times for the respective skill groups. The "U" and "p value" columns depict the test results. The test indicates that the difference in median elapsed of low skill compared to high skill is significant.

B. Modeling

We derive models for ln T parameterized by skill as we did for all users before, again finding the form of model 8 in Table 10 to be the most parsimonious.

TABLE 12. Models of moving target selection time with delay

Skill	Model	Adjusted R^2
All	$mean(ln\ T) = 0.685 + 0.506d + 0.605s + 0.196ds$	0.96
	$stddev(ln.T) = 0.567 + 0.126d + 0.225s + 0.029ds$	0.96
Low	$mean(ln\ T) = 0.850 + 0.560d + 0.672s + 0.212ds$	0.96
	$stddev(ln.T) = 0.589 + 0.118d + 0.253s + 0.009ds$	0.88
High	$mean(ln\ T) = 0.605 + 0.468d + 0.625s + 0.208ds$	0.95
	$stddev(ln\ T) = 0.539 + 0.109d + 0.227s + 0.041ds$	0.96

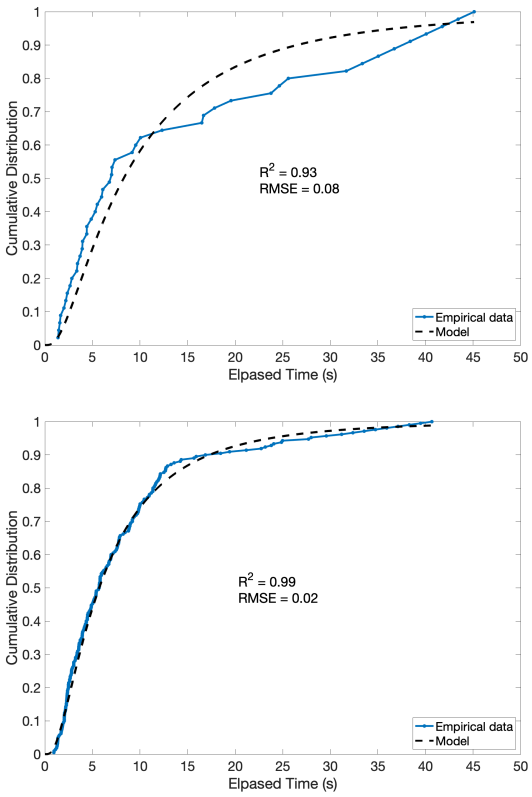

FIG. 15. Example CDF for speed 1100 px/s and delay 320 ms. Top: low skill. Bottom: high skill

The final models for the mean and standard deviation of the natural log of the elapsed time parameterized by player skill are shown in Table 12.

Figure 15 shows example CDF models for a 1100 px/s target and delay 320ms for a low skill player (top) and a high skill player (bottom). For both graphs, the x-axis is the elapsed time and the y-axis is the cumulative distribution. The blue dots are the empirical elapsed times and the black dashed lines are the corresponding models. Our model fits both sets of data well, with R^2 at 0.93, RMSE at 0.08 for the low skill player, and R^2 at 0.99, RMSE at 0.02 for the high skill player.

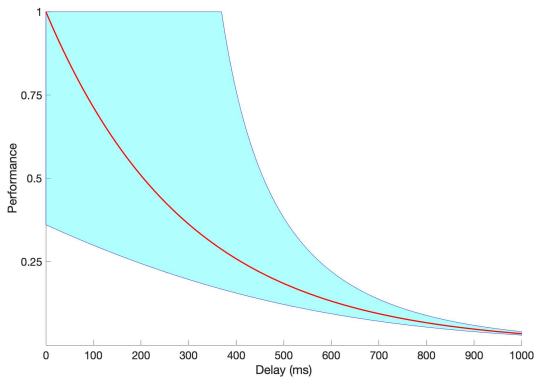

FIG. 16. Performance versus delay for target selection (Analytic model)

8. ANALYTIC MODELS AND SIMULATIONS

As a demonstration of the models use and to evaluate several specific game configurations, we use our model as the basis for analytically modeling and simulating player performance. This allows us to explore the impact of latency compensation techniques and system configuration on game performance over a range of delays.

A. Performance versus Delay- Selection

We begin by modeling player performance of selection versus latency. Using our model, selection time performance with delay is modeled analytically by $\frac{T(0)}{T(n)}$, where T(n) is the mean of elapsed time with delay at an average target speed at 450 px/s, calculated with our model. Figure 16 demonstrates how player performance decays with delay. The x-axis is delay in milliseconds, with a "0" representing the ideal case, and the y-axis is the normalized performance with a "1" representing performance in the best case (0 delay). The red line shows how player performance selecting a 450 px/s target decays with an increase in delay. The light blue region above and below the red line is within one standard deviation of the mean. The graph depicts that player performance and the standard deviation of player performance decreases with delay. Player performance is always poor regardless

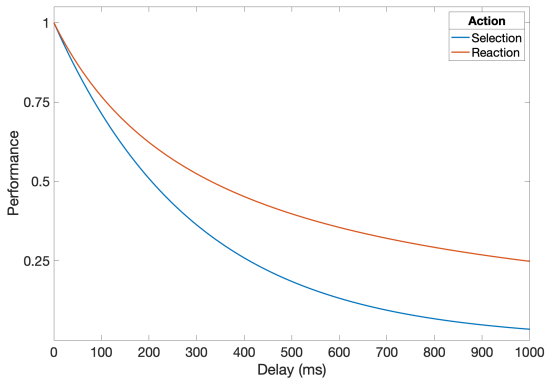

FIG. 17. Performance versus delay for selection and reaction (Analytic model)

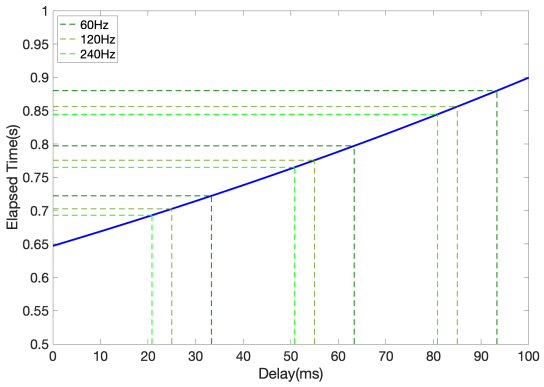

FIG. 18. Frame rate and delay (Analytic model)

of player skill and game difficulty when delay is high. When delay is low, player performance varies due to difference in gaming skill and individual performance.

B. Performance versus Delay– Selection and Reaction

We then build analytic models comparing the impact of delay on performance for target selection compared to reaction actions where a player responds immediately (e.g., by a key press or mouse click) to an event in the game. Selection time performance is calculated the same way as with the last model. Reaction time actions are similarly modeled, but using the mathematical response time derivation by Ma et al. [22].

Figure 17 depicts the effects of delay on player performance for both actions. The x-axis is delay, in milliseconds, and the y-axis is player performance. The blue line shows how player performance selecting a 450 px/s target decays with an increase in delay. The orange line shows how player reaction time performance decays with an increase in delay. From the figure, both actions degrade by about 25% at a modest delay of 100 ms. The blue line has a steeper decreasing trend than the orange line, indicating that delay has more impact on selection actions than on reaction actions.

C. High display frame rate

We next simulate the potential benefits high framerate displays have on game play. We assume a configuration with a total delay of 55 ms and a 120 Hz display, and then evaluate the impact on performance for alternate setups using the same shooting game used above with target speeds of 300 px/s.

Figure 18 depicts the results. The blue line simulates an average skill player, the darkest green dashed line depicts perform if a 60 Hz display were used instead, and the lightest green dashed line if a 240 Hz display were used instead.

TABLE 13. Display frame rate and elapsed time

Delay(ms)	Hz	Elapsed Time(s)	Δ w/Hz	Δ w/Delay
25	60	0.722	-2.70%	9.41%
	120	0.703	—	
	240	0.693	1.42%	
55	60	0.797	-2.71%	—
	120	0.776	—	
	240	0.765	1.42%	
85	60	0.880	-2.80%	-10.31%
	120	0.856	—	
	240	0.845	1.29%	

Table 13 quantifies the results. Column #1 includes three total base delays, each with 120 Hz display. Column #2 shows the display frame rates simulated (60 Hz, 120 Hz, and 240 Hz). Column #3 has the simulated elapsed time at each base delay and Hz display. Column #4 provides the change in performance for the different displays from the base delay. And column #5 provides the change in performance for a 30 ms change in the base delay.

From the table and figure, the performance difference to downgrade to the 60 Hz display is between -2.8% and -2.7%, and the performance difference for an upgrade to the 240 Hz display is between 1.29% and 1.42%. Thus, an upgrade from 60Hz to 120Hz improves player performance more than an upgrade from 120Hz to 240Hz. The performance difference from changing the base delay from 55ms to 25ms is 9.41%, and from 55ms to 85ms is -10.31%. In total, this suggests that base delay and network delay impacts player performance more than does the display frame rate. This confirms user study results measured by Spjut et al. [29].

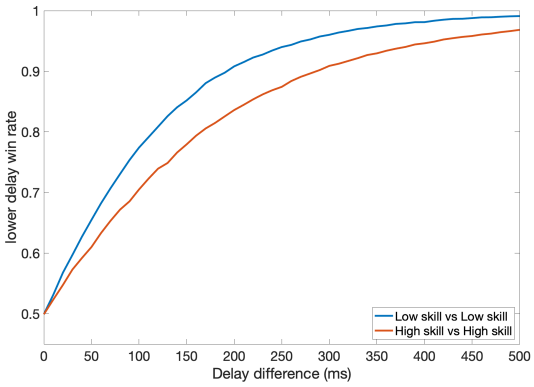

FIG. 19. Win rate versus delay for different skill players (Simulation)

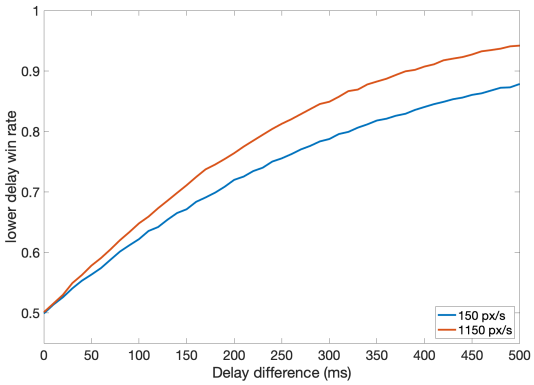

FIG. 20. Win rate versus delay for different target speeds (Simulation)

D. Win Rate versus Delay- Skill

We simulate a shooter game where two players try to select (shoot) a target before their opponent. Both players have an equal base delay, but one player has extra network delay to evaluate the effects of unequal amounts of delay on matches with players of equal skill. We simulate 100k iterations of the game for each combination of added delay and player skill. Figure 19 depicts the results. The x-axis is the delay difference for the two players, and the y-axis is the win rate of the player with lower delay. The blue line is for games with two low skill players, while the orange line is for games with two high skill players. From the figure, even modest delay differences of 100 ms makes it about 50% harder for the delayed player to win. The blue line increases faster than the orange line, indicating delay impacts the games with higher skill players less.

Listing 1. variables

```
base - local delay
speed - speed of the target
w1 - Player1 victories
w2 - Player2 victories
t1 - Player1 selection time
t2 - Player2 selection time
d1 - Player1 round trip delay to server
d2 - Player2 round trip delay to server
```

Listing 2. Game simulation without compensation

```
// Determine player selection times.
t1 = model (speed, base + d1)
t2 = model (speed, base + d2)

// Player that selects first wins.
if (t1 < t2) then
 w1 += 1
else
 w2 += 1
end if
```

E. Win Rate versus Delay- Target Speed

Figure 20 depicts how delay impacts win rate with different target speeds. This simulation is the same as before, but both players are of average skill and the target speed varies. The x-axis is the delay difference for the two players, and the y-axis is the win rate of the player with lower delay. The blue line is games with slow target speeds of 150 px/s, while the orange line is games with fast target speeds of 1150 px/s. The orange line increases faster than the blue line, indicating delay impacts player performance more for higher target speeds (i.e. harder games).

Listing 1 includes variables that will be using in selection simulation. Listing 2 depicts the selection simulation without compensation techniques. We then simulate bucket synchronization and time warp on the selection game.

F. Simulation for Bucket Synchronization

1. *Performance*

Listing 3 depicts how bucket synchronization is simulated in selection. Figure 21 depicts how players' performances change after applying bucket synchronization. The graph on the top depicts the condition without compensation. The x-axis is delay in milliseconds and the y-axis is player performance. The blue line describes how player performance decays with delay, and it contains the same data as the simulation on performance for selection above. The red rectangle represents player 1 with delay at 100 ms, and performance at around 0.7. The blue star represents player 2 with delay at 400 ms, and per-

Listing 3. Bucket synchronization

```
0
1  // Compute bucket latency.
2  b1 = max(d2 - d1, 0)
3  b2 = max(d1 - d2, 0)
4
5  // Determine player selection times with bucket.
6  t1 = model (speed, base + d1 + b1)
7  t2 = model (speed, base + d2 + b2)
8
9  // Player that selects first, wins.
10 if (t1 < t2) then
11  w1 += 1
12 else
13  w2 += 1
14 end if
```

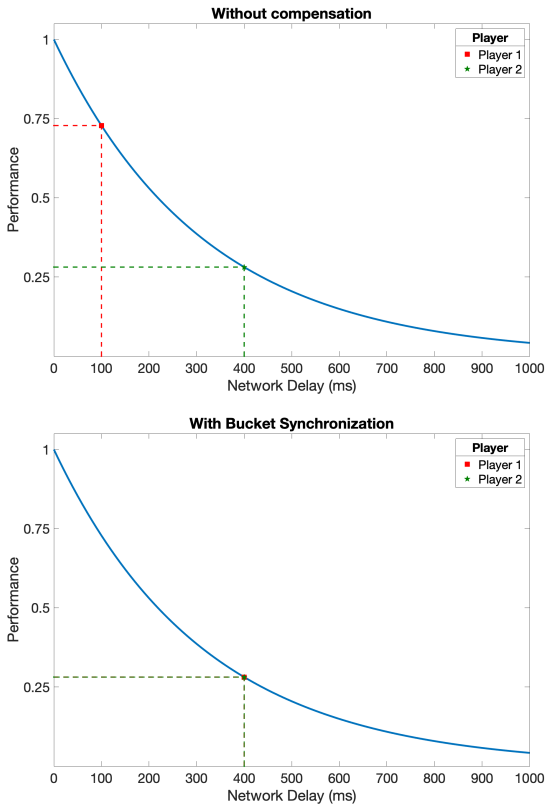

FIG. 21. Performance versus delay (Analytic model) Top: without compensation. Bottom: After using Bucket synchronization

formance at around 0.3. The graph on the bottom is the condition after applying bucket synchronization. The x-axis, the y-axis, and the blue line are the same as the graph on top. With bucket synchronization, both players would experience the maximum delay of player 1 and player 2, which is 400 ms. Thus, they would have the same performance at around 0.3.

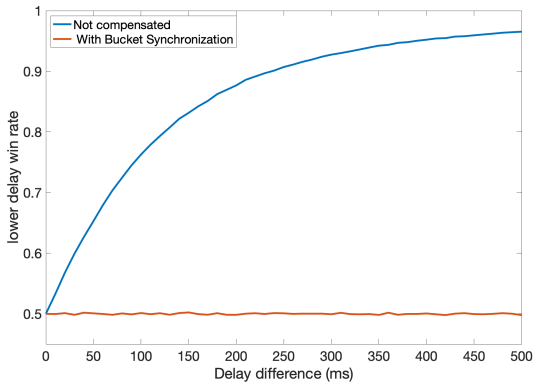

FIG. 22. Win rate versus delay difference for bucket synchronization (Simulation)

2. *Win rate*

Next, we simulate a game where there are two players playing a selection game, and the player with the shortest time to select wins. The two players have equal latency in the beginning, and we add extra latency to one player. First, we simulate the game without any compensation technique, and then we simulate the game with bucket synchronization.

Figure 22 depicts how players win rates change after applying bucket synchronization. The x-axis is delay difference of two players, and the y-axis is win rate of lower delay player. The blue line represents how the lower latency player win rate increases with delay difference when the players are not compensated. The orange line represents the condition when bucket synchronization is applied. Without compensation, when the delay difference is high, the lower latency player would almost always win. After applying bucket synchronization, the two players would have the same chance to win since they experience the same amount of delay.

G. Simulation for Time Warp

1. *Performance*

Listing 4 depicts how time warp is simulated in selection. Figure 23 depicts how time warp improves player performance for selection. The graph on the top depicts the condition without compensation. It is the same graph as the top graph in Figure 21. The graph on the bottom is the condition after applying time warp. The x-axis, the-y axis, and the blue line are the same with the graph on top. Both players would experience no extra delay from the network, and thus have performance at 1.

Listing 4. Time warp

```
// Compute time warp latency.
b1 = -d1
b2 = -d2

// Determine player selection times with time warp.
t1 = model (speed, base + d1 + b1)
   = model(speed, base)
t2 = model (speed, base + d2 + b2)
   = model(speed, base)

// Player that selects first, wins.
if (t1 < t2) then
 w1 += 1
else
 w2 += 1
end if
```

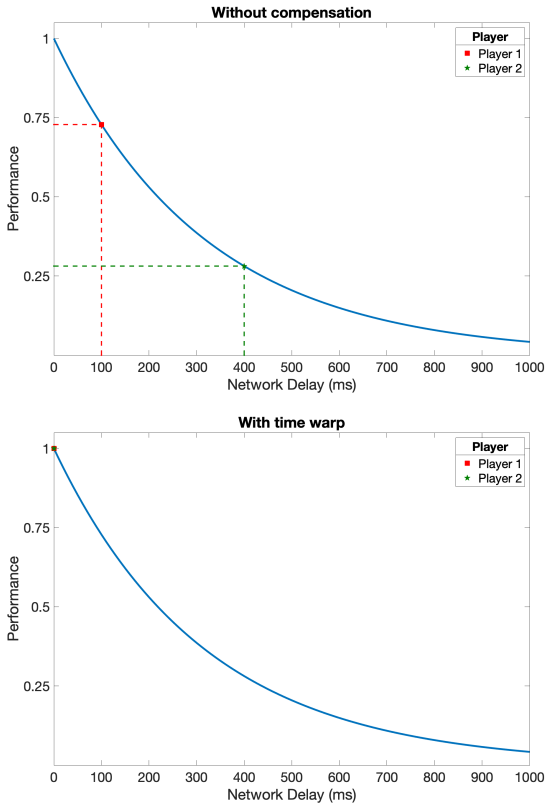

FIG. 23. Performance versus delay (Analytic model)
Top: without compensation. Bottom: After using time warp

2. *Win rate*

In this simulation, the game is still the same as the win rate of bucket synchronization, but we are applying time warp instead of bucket synchronization.

Figure 24 depicts how players' win rates change after applying time warp. The x-axis is the delay difference between the two players, and the y-axis is the win rate of the lower delay player. The blue line depicts the lower latency player win rate increase with the delay difference when the players are not compensated. The orange line represents the condition when time warp is applied. Without compensation, when the delay difference is high, the lower latency player would always win. After applying time warp, the two players would have the same chance to win.

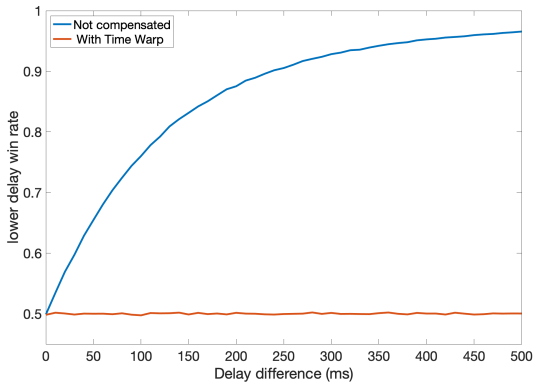

FIG. 24. Win rate versus delay difference for time warp (Simulation)

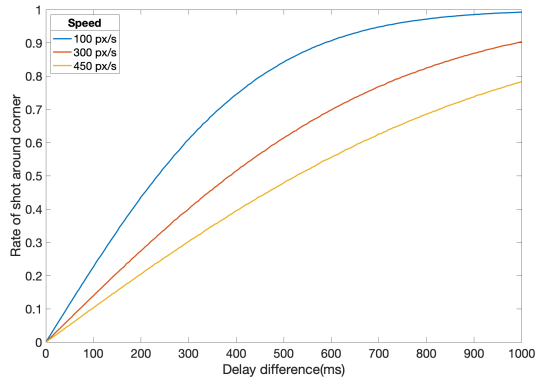

FIG. 25. Rate of shot around corner (Simulation)

3. *Shot around corner*

Shot around corner is known to be a problem caused by time warp, where a player gets shot when in a position of cover. This simulation explores the probability of 'shot around corner' and delay. We simulate a game where there are two players playing a target selection game with the same amount of delay under 3 different target speeds. The player with the shorter time to select the target wins the trial. If a player hits a target before getting informed of losing, it is recorded as a 'shot around corner'.

Listing 5 depicts how 'shot around corner' defined by pseudocode. In a two players selection game, if a player selects the target before being informed of losing, the player gets 'shot around corner'. Figure 25 depicts how the rate of 'shot around corner' increases with delay.

Listing 5. 'shot around corner' simulation

```
0
1  //Determine player selection times with time warp.
2  t1 = model(speed, base)
3  t2 = model(speed, base)
4
5  //Determine when the player selection event arrives at
       server
6  s1 = t1 arrives at server
7     = t1 + d1/2
8  s2 = t2 arrives at server
9     = t2 + d2/2
10
11 //Determine when the player selection event arrives at
       the other client
12 c1 = Player1 gets player2 selection event
13    = s2 + d1 / 2
14 c2 = Player2 gets player1 selection event
15    = s1 + d2 / 2
16
17
18
19 // Player that selects first wins.
20 if t1 < t2 then
21  player1 wins
22 else if t2<t1
23  player2 wins
24
25 //Determine if 'shot around corner' happens to the
       players
26 sac1 = Player 1 is shot around the corner
27 sac2 = Player 2 is shot around the corner
28
29 if player1 wins and t2<c2
30   sac2
31 if player2 wins and t1<c1
32   sac1
```

The x-axis is delay and the y-axis is probability of 'shot around corner'. Three colors denote three different target speeds. Overall, for a higher network latency there is a higher probability of 'shot around corner'. The probability of 'shot around corner' increases faster with low target speeds.

9. UNRESPONSIVENESS-INCONSISTENCY MODEL

There is a trade-off between consistency and responsiveness with most latency compensation techniques.

Inconsistency causes players to see the world differently [19]. It is the degree of mismatch between different users' views of the virtual world. In multiplayer games, inconsistencies arise due to a combination of network latency and the use of prediction by the client to determine the local game state.

To avoid inconsistencies, it would be desirable to synchronize the game states. However, this synchronization can lead to another unwanted latency-related artifact, increased response time. Response time or responsiveness is the time difference between when an operation is issued and when it is executed. In many situations, minimizing of response time and avoiding inconsistencies are conflicting goals.

We depict the space of consistency and responsiveness

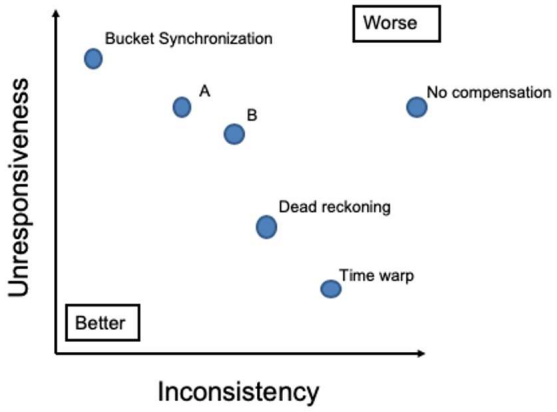

FIG. 26. The 'Responsiveness and Consistency' space

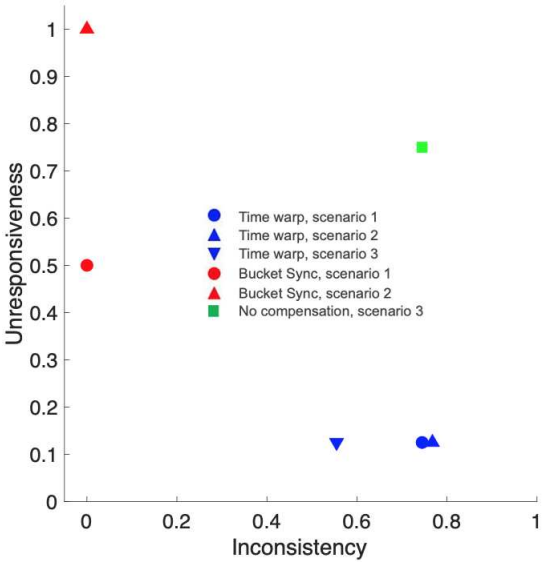

FIG. 27. The 'Responsiveness and Consistency' space
The red dots are bucket synchronization examples, and blue dots are time warp examples.

in Figure 26. The x-axis is inconsistency and the y-axis is unresponsiveness. The upper rightmost blue dot denotes no compensation. Without using compensation techniques, players would experience high unresponsiveness and high inconsistency. The other blue dots denote some existing latency compensation techniques. The shorter the distance to the bottom left corner, better the performance of the compensation technique.

We use our model to simulate the location of game configuration and compensation techniques in the consistency-responsiveness space. Figure 27 shows the model with time warp at three difference scenarios, and bucket synchronization at two different scenarios. The x-axis is the normalized inconsistency, measured by the game states differences between clients. For time warp, it can be measured by the normalized rate of 'shot around

corner'. For bucket synchronization, the inconsistency is always 0 because the techniques enable the players to share the same view of the game world. The y-axis is the normalized unresponsiveness, measured by the normalized delay perceived by players. For time warp, the unresponsiveness is always 0 because the server rolls players' state back hence they can act at the game state on their screen. The red dots are bucket synchronization examples, and blue dots are time warp examples. Circles are simulations at scenario 1 (target speed= 100 px/s, and players' network delay of 100 ms and 200 ms), triangles are simulations at scenario 2 (target speed= 300 px/s, and players' network delays of 200 ms and 400 ms), and inverted triangles are simulations at scenario 3 (target speed= 450 px/s, and players' network delays of 100 ms and 400 ms). All scenarios have a local delay of 50 ms. This figure depicts that time warp compensates unresponsiveness well, while bucket synchronization compensates inconsistency well.

10. CONCLUSION

With the growth in networking and cloud services, applications are increasingly hosted on servers, adding significant delay between user input and rendered results. This is true of computer games, where user input in response to game events can be delayed by the local system and the networking and processing by the server before rendered results are displayed on the screen. Latency compensation techniques are designed to compensate the negative effects latency brings to players. Understanding the impact of delay and the compensation techniques on game input can help build systems that better deal with the delays.

While user studies can be accurate for assess games and latency compensation techniques, they are time intensive and, by necessity, typically have a limited range of parameters they can test. Analytic models and simulations that mimic the behavior of players and compensation techniques in games can complement user studies, providing for a broader range of study. This latter approach is most effective if the game modeled and simulated incorporates observed user behavior. Our work leverages data gathered from two previous user studies to build a model that can be used for just such an approach – analytic models and simulations.

Eighty-three users each playing 167 rounds of a game provided data on the time to select a moving target, the target moving with 6 different speeds and the mouse input subjected to 22 different amounts of delay. For each difficulty level, the resultant elapsed time data on the time to select the targets appears log-normal. We used multivariate, multiple regression to model the mean and standard deviation of natural log of the elapsed time, with additive linear terms for delay and speed and a multiplicative interaction term. This use of delay and speed is critical for accuracy of the models, as is the interaction term as there is an interplay of high delay and high speed that impacts elapsed times. The models can be used to generate a normal distribution of logarithmic elapsed times, then expanded by taking the exponential to get a distribution of elapsed times. Furthermore, the same approach is used to model elapsed time distributions based on self-rated user skill, which we ascertained could be differentiated into two tiers: low (self-rated skill 1-3) and high (self-rated skill 4-5). As proposed, our models have an excellent fit (R^2 around 0.99 and low root mean square error).

In addition to the contribution of a model for the distribution of target selection times with delay, we demonstrate use of the model by analytically modeling and simulating player performance and latency compensation techniques in a game that features target selection. The analytic modeling and simulation results show that for the evaluated game:

1. When delay is low, player performance varies. Player performance is similarly always poor with high delay.

2. Even delays of only 100ms make it about 50% harder to win.

3. High skill players are less affected by delay than low skill players.

4. Delay affects players more for faster targets.

5. With bucket synchronization and time warp, players at the same skill level with different delay would have the same chance to win.

6. With time warp, players can perform as if there is no delay. With bucket synchronization, players all perceive the same amount delay as the highest delay player.

7. The rate of 'shot around corner' caused by time warp increases with delay and target speed.

8. Improving delays by as little as 30 ms can improve player performance more than increasing frame above 60 Hz.

There is a trade-off between consistency and responsiveness with most latency compensation techniques. For example, time warp compensates unresponsiveness well and bucket synchronization compensates inconsistency well. We propose a novel representation of the space of the latency techniques based on normalized inconsistency and normalized unresponsiveness. The closer a game configuration with latency compensation techniques is to the bottom left corner, the better the player quality of experience.

11. FUTURE WORK

While the models presented in this paper provide insights into a meaningful and fundamental measure of performance – the elapsed time to select – another important measure of performance for selection is accuracy. Accuracy in the datasets used in this work manifests itself in the number of clicks needed to select the target – any number greater than 1 is a "miss". Future work could model accuracy in a manner similar to the elapsed time models in this work. These accuracy models could be combined with our elapsed time models to simulate shooter games with ammunition, wherein accuracy matters in terms of number of shots taken or allowed.

The atomic action of moving target selection is only one of many fundamental game actions that are affected by delay. Future work could design and conduct user studies to gather data on the effects of delay on other atomic actions. Data from these studies could the modeled as in this paper, and those models combined with our models of target selection to simulate the effects of delay for a richer set of games. For example, a user study that assessed the impact of delay on avatar navigation could provide data for models of the impact of delay on player avatar movement. These navigation models could be used in conjunction with our models of target selection to simulate a wide range of shooter games, allowing study of the effects of delay for the many commercial shooter games that have not been evaluated, nor possibly even built.

The x-axis and y-axis in the consistency-responsiveness model are normalized separately with different terms. They could be normalized by the tolerance of players for unresponsiveness and inconsistency, making them comparable. Future work could include a user study to determine these limits of the player tolerance for unresponsiveness and inconsistency.

[1] Normality testing - skewness and kurtosis. `https://tinyurl.com/ybacnyx7`.

[2] Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 161–165, 2004.

[3] Yahn W Bernier. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033, 2001.

[4] De-Yu Chen and Hao-Tsung Yang dn Kuan-Ta Chen. Dude, the source of lags is on your Computer. In *Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, Denver, CO, USA, December 2013.

[5] Kuan-Ta Chen, Yu-Chun Chang, Hwai-Jung Hsu, De-Yu Chen, Chun-Ying Huang, and Cheng-Hsin Hsu. On the Quality of Service of Cloud Gaming Systems. *IEEE Transactions on Multimedia*, 26(2), February 2014.

[6] Kyungmin Lee David Chu, Eduardo Cuervo, Johannes Kopf, Sergey Grizan, Alec Wolman, and Jason Flinn. Outatime: Using speculation to enable low-latency continuous interaction for cloud gaming. 2014.

[7] Mark Claypool. Game input with delay—moving target selection with a game controller thumbstick. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(3s):1–22, 2018.

[8] Mark Claypool and Kajal Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[9] Mark Claypool and Kajal Claypool. Latency can kill: precision and deadline in online games. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 215–222, 2010.

[10] Mark Claypool, Ragnhild Eg, and Kjetil Raaen. Modeling user performance for moving target selection with a delayed mouse. In *International Conference on Multimedia Modeling*, pages 226–237. Springer, 2017.

[11] Mark Claypool and David Finkel. The effects of latency on player performance in cloud-based games. In *13th Annual Workshop on Network and Systems Support for Games*, pages 1–6. IEEE, 2014.

[12] Dr Fermin Moscoso del Prado Martin. A theory of reaction time distributions. January 2009.

[13] Matthias Dick, Oliver Wellnitz, and Lars Wolf. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–7, 2005.

[14] Christophe Diot and Laurent Gautier. A distributed architecture for multiplayer interactive applications on the internet. *IEEE network*, 13(4):6–15, 1999.

[15] Chen Gao, Haifeng Shen, and M Ali Babar. Concealing jitter in multi-player online games through predictive behaviour modeling. In *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 62–67. IEEE, 2016.

[16] Eben Howard, Clint Cooper, Mike P Wittie, Steven Swinford, and Qing Yang. Cascading impact of lag on quality of experience in cooperative multiplayer games. In *13th Annual Workshop on Network and Systems Support for Games*, pages 1–6. IEEE, 2014.

[17] Aaron Isaksen, Daniel Gopstein, and Andrew Nealen. Exploring game space using survival analysis. In *FDG*, 2015.

[18] Zenja Ivkovic, Ian Stavness, Carl Gutwin, and Steven Sutcliffe. Quantifying and mitigating the negative effects of local latencies on aiming in 3d shooter games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 135–144, 2015.

[19] Xinbo Jiang, Farzad Safaei, and Paul Boustead. Latency and scalability: a survey of issues and techniques for supporting networked games. In *13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic*, volume 1, pages 6–pp. IEEE, 2005.

[20] Steven WK Lee and Rocky KC Chang. Enhancing the experience of multiplayer shooter games via advanced lag

compensation. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 284–293, 2018.

[21] Eckhard Limpert, Werner A Stahel, and Markus Abbt. Log-normal distributions across the sciences: keys and clues: on the charms of statistics, and how mechanical models resembling gambling machines offer a link to a handy way to characterize log-normal distributions, which can provide deeper insight into variability and probability—normal or log-normal: that is the question. *BioScience*, 51(5):341–352, 2001.

[22] Tao Ma, John Holden, and R. A. Serota. Distribution of Human Response Times. *Complexity*, 21(6):61–69, 2013.

[23] Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE transactions on Multimedia*, 6(1):47–57, 2004.

[24] Manuel Oliveira and Tristan Henderson. What online gamers really think of the internet? In *Proceedings of the 2nd workshop on Network and system support for games*, pages 185–193, 2003.

[25] Saeed Shafiee Sabet, Steven Schmidt, Carsten Griwodz, and Sebastian Möller. Towards the impact of gamers' adaptation to delay variation on gaming quality of experience. In *2019 Eleventh International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2019.

[26] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Carsten Griwodz, and Sebastian Moller. Towards applying game adaptation to decrease the impact of delay on quality of experience. In *IEEE International Symposium on Multimedia (ISM)*, pages 114–121. IEEE, 2018.

[27] Steven Schmidt, Saman Zadtootaghaj, and Sebastian Möller. Towards the delay sensitivity of games: There is more than genres. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2017.

[28] Paul M Sharkey, Matthew D Ryan, and David J Roberts. A local perception filter for distributed virtual environments. In *Proceedings. IEEE Virtual Reality Annual International Symposium (Cat. No. 98CB36180)*, pages 242–249. IEEE, 1998.

[29] Josef Spjut, Ben Boudaoud, Kamran Binaee, Jonghyun Kim, Alexander Majercik, Morgan McGuire, David Luebke, and Joohwan Kim. Latency of 30 ms benefits first person targeting tasks more than refresh rate above 60 Hz. In *SIGGRAPH Asia 2019 Technical Briefs*, pages 110–113. 2019.

[30] Wikipedia. Kolmogorov–Smirnov test — Wikipedia, the free encyclopedia. `https://tinyurl.com/y6uwjzz5`, 2020. [Online; accessed 06-May-2020].

Appendix A: Appendix

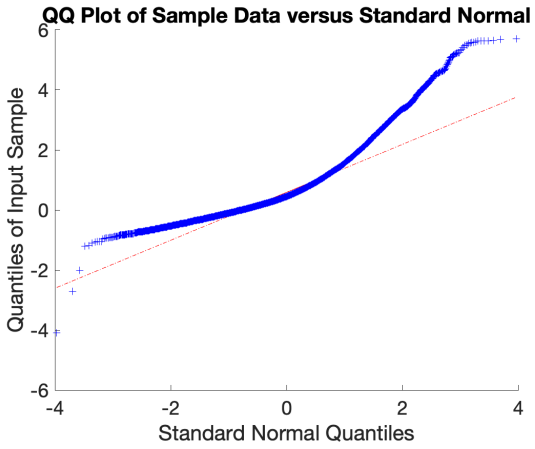

FIG. 28. QQ plot of log(Elapsed time)

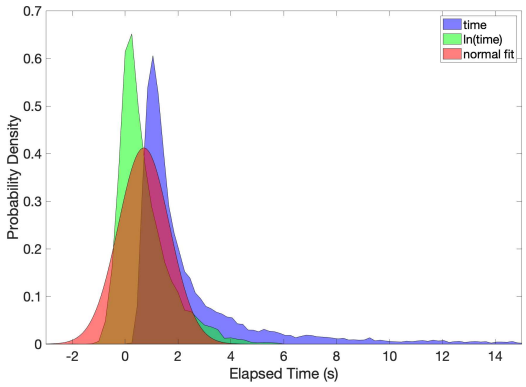

FIG. 29. PDFs of original data, log data, and normal fit of log data

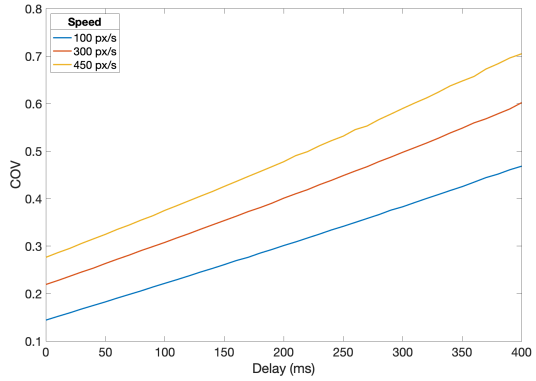

FIG. 30. Coefficient of variation vs delay

TABLE 14. S-W test of WPI difficulties

	delays										
Speed	100	125	150	175	200	225	250	275	300	400	500
150	0.16	**0.00**	0.59	0.37	0.15	0.28	0.03	0.13	**0.01**	0.15	0.08
300	0.00	0.07	0.80	0.64	0.49	**0.01**	0.05	**<0.001**	0.14	**0.01**	**0.00**
450	0.36	**<0.001**	**0.00**	0.34	0.67	0.24	**0.00**	0.33	**<0.001**	**<0.001**	**0.01**

TABLE 15. S-W test of Oslo difficulties

	delays										
Speed	20	45	70	95	120	145	170	195	220	320	420
550	**0.04**	0.33	**<0.001**	**0.01**	**<0.001**	0.17	**0.00**	0.08	**0.01**	**0.00**	**0.00**
1100	**<0.001**	**0.01**	0.10	**0.00**	**<0.001**	**<0.001**	**<0.001**	**0.00**	0.11	**<0.001**	**<0.001**
1550	0.05	0.36	**0.00**	0.29	**0.01**	0.05	**0.01**	**<0.001**	**<0.001**	**<0.001**	**<0.001**

TABLE 16. R^2 of Set-B difficulties

	delays										
Speed	100	125	150	175	200	225	250	275	300	400	500
150	1.00	0.98	1.00	0.98	1.00	0.96	1.00	0.98	0.97	0.95	0.99
300	1.00	0.98	0.97	0.99	0.94	0.97	0.97	0.98	0.98	0.94	0.95
450	0.98	0.98	0.99	0.99	0.99	0.98	1.00	0.97	0.99	0.96	0.98

TABLE 17. RMSE of Set-B difficulties

	delays										
Speed	100	125	150	175	200	225	250	275	300	400	500
150	0.01	0.04	0.02	0.04	0.01	0.05	0.01	0.04	0.05	0.06	0.03
300	0.01	0.04	0.04	0.02	0.05	0.04	0.03	0.03	0.03	0.05	0.04
450	0.03	0.03	0.03	0.02	0.02	0.03	0.02	0.04	0.03	0.04	0.04

TABLE 18. R^2 of Set-A difficulties

	delays										
Speed	20	45	70	95	120	145	170	195	220	320	420
550	0.95	0.98	0.95	0.96	0.93	0.97	0.97	0.94	0.93	0.88	0.63
1100	0.99	0.94	0.97	0.99	0.96	0.92	0.99	0.97	1.00	0.96	1.00
1550	0.98	0.93	0.93	0.99	0.89	0.83	0.92	0.92	0.53	0.97	0.42

TABLE 19. RMSE of Set-A difficulties

	delays										
Speed	20	45	70	95	120	145	170	195	220	320	420
550	0.04	0.04	0.04	0.05	0.04	0.04	0.03	0.06	0.04	0.07	0.12
1100	0.03	0.04	0.05	0.02	0.06	0.05	0.02	0.02	0.02	0.03	0.01
1550	0.02	0.07	0.04	0.02	0.04	0.10	0.04	0.06	0.14	0.03	0.21

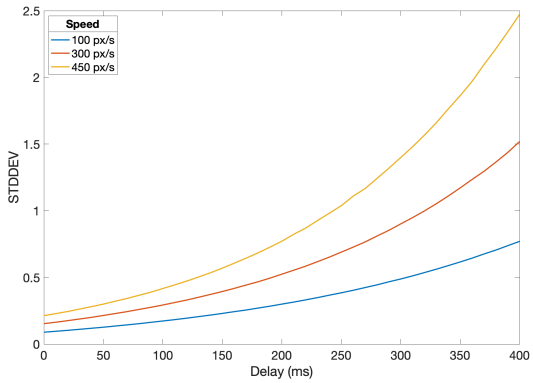

FIG. 31. *stddev* vs delay