The Impact of Latency on Players in First-person Shooter Games

A Dissertation Submitted to the Faculty

by

Shengmei Liu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Computer Science Department
Worcester Polytechnic Institute
Dec 2022

Ph.D. Advisor (Mark Claypool)	Department Head (Craig Shue)

Abstract

The first-person shooter game is the most popular genre in esports [Mur21] and among the most affected by latency [CC10]. In general, the lower the latency, the sooner a player sees the outcome of their actions. Techniques to study latency and individual games cannot scale to cover all possible first-person shooter games. Our approach is to study and model the primary actions in first-person shooter games, and use the models as building blocks to simulate first-person shooter scenarios with latency. We focus on the two main actions players take in first-person shooter games – navigation (get in position to shoot or avoid being shot), and selection (shoot at a moving or stationary target). We gather data on each action via user studies and build mathematical models for player performance. By incorporating the models for different actions, we simulate game scenarios by sampling in-sight windows and shots in the windows and varying parameters for different game scenarios. We validate the simulation results using data from first-person shooter games, finding that our simulation predicts outcomes from a custom FPS game well, but less well on CS:GO data probably due to difference in game modes and built-in latency compensation. Once validated, we simulate first-person shooter games with a broad range of games and game configurations including latency, latency compensation, player skill, room size, firing rate and target size. We find that latency compensation, number of hits required to kill an opponent and player skill have large effects on the performance of players with latency, but map size, effect size and firing rate do not.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer games are one of the world's most popular forms of entertainment, with global sales increasing at an annual rate of 10% or more [wep22]. Among the different game genres, first-person shooter (FPS) games are some of the most popular. Over 20% of sales of computer games were FPS games in 2019 and about one-third of all gamers like to play FPS games [rai21]. In all FPS games, players take on the first-person perspective of an avatar and move and shoot targets to accomplish tasks, but there is a wide range of play modes, weapons and maps.

FPS games are among the most sensitive to latency [CC06]. Latency between a player's input and the game output can impact the responsiveness and consistency of the game, hurting player performance and degrading quality of experience. To reduce latencies, serious gamers typically want fast computers with upgraded processing, memory and graphics capabilities, and fast networks with low round-trip times between the client and the game server. There are two main sources of latency in first-person shooter games – from the local system, such as from the mouse, operating system and monitor, and from the network between the client and the server. While both sources of latency affect the player, they manifest differently – local latency lags all player input until game output, while network latency lags communication with the server. This means local latency makes game controls feel unresponsive, while network latency makes player actions resolved later by the server.

Many latency compensation techniques [LXC21] were developed to mitigate the effects of latency on gamers, usually designed for network latency. In general, the techniques either mask latency from the player perspective or adjust game states between clients for more consistency and fairness. Most online commercial first-person shooter games deploy several latency compensation technique to improve player performance and quality of experience: 1) With time warp, the server makes decisions based on game states when the client acted so players can play as if there is no network latency; and 2) with self-prediction, the client predicts the game state based on player input, but before getting confirmation from the server.

There have been studies on latency and commercial games [FRS05, HFPG16], especially network latency and FPS games [Arm03, BCL+04, QML+04, AJG+13] owing to the sensitivity of FPS games to network latency and the popularity of FPS games in the competitive and esports scenes. However, such studies often evaluate games with built-in latency compensation techniques, thus the results might not generalize to other games. Moreover, theses studies do not evaluate how the latency compensation techniques improve player performance and quality of experience. Other games research has studied local latency, usually focusing on a subset of a full game [LC15, LG18, LG19] - for example, a gaming action or task instead of the full game. While this prior research has been valuable for understanding latency and games, it has focused on a specific game task or a specific game with pre-set conditions. There has been no good answer as to how latency degrades performance for first-person shooter gamers across different games and game configurations (e.g., weapons, avatar size and map size), latency types, latency values and latency compensation techniques.

A common approach to study latency and games is with user studies, which can be expensive and time-consuming. A user study for one game and a limited set of conditions normally takes at least 6 months and tens of thousands of dollars. Given the wide range of games and possible configurations, player skills, latency types and values, and latency compensation techniques, it is not practical to iterate over all possible FPS configurations with user studies. Plus, there are future first-person shooter games with different configurations that have not yet been developed.

Instead of studying each possible FPS configuration, our approach is to isolate and model game actions in FPS games and then use those models to simulate full games. Game actions are types of player interactions within the game. There are 2 primary game actions in FPS games: navigation and selection. Navigation is moving an avatar to a position to avoid being shot or to better shoot enemies. Selection is pointing to a target with an input device and clicking to shoot, e.g., moving a reticle over an opponent avatar and clicking the mouse to shoot the opponent. We collect player performance and QoE data on the two FPS game actions with user studies. In the user studies, the latency types and values and compensation techniques are controlled and applied to the game rounds.

From the user study data, we build mathematical models for the distribution of player performance versus latency considering latency compensation techniques and player skill. We integrate the models of navigation and selection into simulations on different FPS game scenarios and validate the simulation results using data from FPS games. The validated simulations are used to explore a wide range of game scenarios and game configurations.

Analysis of the results from the navigation user study indicates local latency and network latency have similar effects on navigation in the absence of latency compensation. Both subjective quality of experience (QoE) and objective player performance degrade linearly with total latency, where 100 ms increase in latency results in about an 8 percent decrease in player score and a 20 percent decrease in QoE. The impact of latency depends upon the navigation goal, however, with latency hindering seeking far more than hiding. We derive an analytic model that uses a Weibull equation to describe the time intervals that is of use for generalizing navigation with latency and is used in our first-person shooter simulations. Analysis of the results from the selection user study indicates both subjective quality of experience (QoE) and objective player performance degrade linearly with total latency. A 100 ms increase in latency results in about 30 percent improvement in time to select (shoot a target) and a 12 percent decrease in QoE. We derive analytic models of the time to select that generalize target selection with latency and latency compensation that uses an exponential equation and is used in our first-person shooter simulations. Both latency compensation techniques investigated – time warp and self-prediction – can improve player performance and QoE and, when applied together, can nearly completely overcome the effects of latency on performance and QoE.

We use our models in simulations of first-person shooter scenarios, validating them with data from a custom first-person shooter game and CS:GO. The simulation predicts game scenarios from the the custom first-person shooter game well for most conditions evaluated. There are some differences between simulation results and CS:GO results, probably due to difference in game mode (in CS:GO, players play against 20 other bots) and built-in latency compensation techniques, but overall trends with latency hold.

Finally, we are able to explore the rich space of first-person shooter games with the validated simulations. The exploration reveals that latency, latency compensation, number of hits required to kill, firing rate (when number of hits required to kill is high), and player skill have a large effect on player performance in the presence of latency, while target size, map size and firing rate (when number of hits required to kill is low) have a lower effect.

The rest of this thesis is organized as follows: Chapter 2 gives background on work related to games, latency and latency compensation techniques; Chapter 3 describes the related work and literature on the actions of selection and navigation based on latency and player skills; Chapter 4 provides our approach to understand the impact of latency on first-person shooter game players; Chapter 5 and Chapter 6 details user studies and model of navigation and target selection in first-person shooter games respectively; Chapter 7 describes the simulations where we integrate the models and simulate first-person shooter scenarios; Chapter 9 validates the simulations using a custom first-person shooter game; Chapter 8 validates the simulations using a commercial first-person shooter game - Counter Strike: Global Offensive; Chapter 10 explores the impacts of latency on player performance with different latency compensation techniques, number of hits required, firing rate, map size, target size and player skill using the validated simulations; Chapter 11 summarizes our conclusions and contributions in this dissertation; and Chapter 12 includes possible future work.

Chapter 2

Background

This chapter presents the background of our research in First-person shooter games (Section 2.1), client-server architectures (Section 2.2) and latency compensation techniques (Section 2.3).

2.1 First-person shooter game

A computer game involves player interactions in a virtual world using an input device – such as a mouse or a controller – to update the game states, which are then rendered and displayed on a screen, such as a monitor or a TV. Players typically execute one or more tasks to accomplish missions for fun and satisfaction. In many games, players can also easily interact with other players over the Internet. Computer games have been one of the most popular entertainments. The computer game industry is estimated to be worth $197 billion USD in 2022, with a steady upward trend over the past 10 years. Among all gaming platforms, PCs continue to be one of the most popular platforms for computer games despite the increasing competition from console and mobile gaming platforms [wep22]. First-person shooter games are among the most popular game genres. Table 2.1 lists the top 10 esports games by total prize pool in 2019 [Mur21], of which 6 are first-person shooter games.

In a first-person shooter game, players take on the first-person perspective of an avatar, moving and shooting targets with various types of weapons to accomplish goals in a 3D environment. The game mode can be either single player, multiple players playing against each other (free for all), or multiple players forming a team and collaborating. Most first-person shooter games are beginner-friendly and challenging at the same time [rai21]. Jansz and Tanis [JT07] point out that the socialization in first-person shooter games makes people spend even more time playing them. First-person shooters have been one of the most popular game genres. Valorant (Riot Games, 2020), a first-person shooter game launched in June 2020, was played by about 3 million people per day [Sha21]; Apex Legends (Electronic Arts, 2019) has over 70 million total registered players and the "old school" game Counter Strike: Global Offensive (Valve, 2012) first released in April 2012 still has about 600,000 concurrent players on average [gam20].

Table 2.1: Top 10 Esports Games in 2019 [Mur21]

Rank	Prize pool(USD)	Game	Year	Publisher	Game genre
1	64.4M	Fortnite	2017	Epic Games	First/Third-person Shooter
2	46.7M	DOTA 2	2013	Valve Corporation	Multiplayer Online Battle Arena
3	21M	Counter Strike: Global Offensive	2012	Valve Corporation	First-person Shooter
4	12.7M	PUBG	2017	Bluehole Inc.	First/Third-person Shooter
5	9.1M	Overwatch	2016	Blizzard Entertainment	First-person Shooter
6	9M	League of Legends	2009	Riot Games	Multiplayer Online Battle Arena
7	8.9M	Magic: The Gathering	1993	Wizards of the Coast	Strategy
8	6.5M	Call of Duty: Black Ops	2010	Activision	First-person Shooter
9	5.8M	Arena of Valor	2016	Garena	Multiplayer Online Battle Arena
10	4.1M	Rainbow Six Siege	2015	Ubisoft	First-person Shooter

Moreover, first-person shooter games are among the game genres which are most impacted by latency. Related research has proposed game classification approaches and models to help identify the sensitivity of games to latency.

Claypool and Claypool [CC06] define the precision-deadline model motivated by the effects of latency. Figure 2.1 depicts the model where the x-axis is deadline and the y-axis is precision. With the model, for a given game action, the higher the

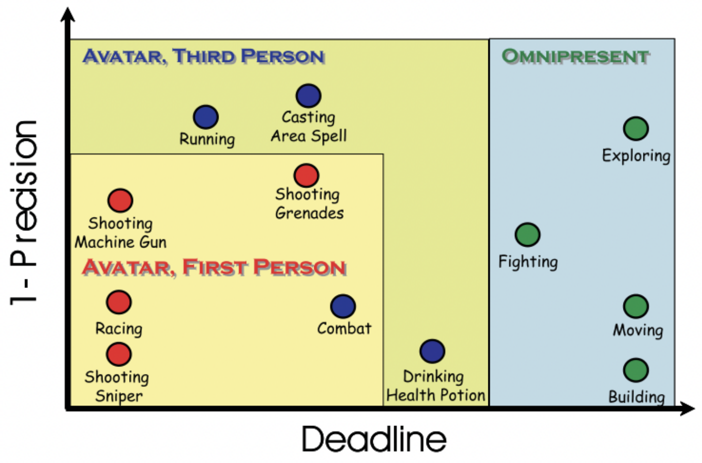


Figure 2.1: Game genres – precision-deadline classification [CC10]

precision required and the tighter the deadline the greater the impact of latency on performance. The closer the game to the origin in the model (the bottom left corner), the greater the impact of latency on performance. First-person perspective games normally have the highest precision and tightest deadline, third person perspective games follow and omnipresent perspective games generally have the lowest precision and loosest deadline. Within first-person perspective games, shooting games with high-precision weapons are among the most heavily impacted by latency.

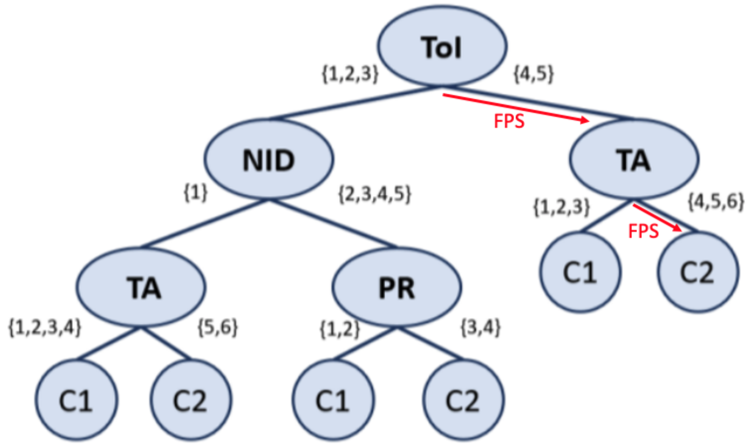


Figure 2.2: Delay sensitivity decision tree depending upon different gaming characteristics [SSZ+20]

Based on the precision-deadline model, Sabet et al. [SSZ$^+$20] present an evaluation method to classify games with respect to their delay sensitivity. They define 9 characteristics which influence the sensitivity of a game towards delay (without the presence of latency compensation techniques), and provide the approaches to quantify the characteristics. The 9 characteristics are Temporal Accuracy (TA) – the available time interval for a player to perform a desired interaction, Spatial Accuracy (SA) – the degree of precision required to complete an interaction successfully, Predictability – if a player is able to estimate the upcoming events in the game, number of Input Directions (NID): – the number of possible input directions in a game scenario, Consequences (CQ) – the negative consequences due to failing to perform the desired action, Importance of Actions (IoA) – how much each action for a game scenario can change its outcome, number of Required Actions (NRA), Feedback Frequency (FF) – how often the game gives feedback to the player and Type of Input (ToI) – the temporal aspects of player inputs on a spectrum of discrete to continuous. The characteristics were quantified with 30 different games, and the games were mapped by means of a decision tree, as indicated in Figure 2.2. With the decision tree, two classes of sensitivity (low and high) were defined based on four characteristics, ToI, NID, PR and TA.

First-person shooter games typically have both continuous and discrete inputs which have a TOI of 5. Moreover, first-person shooter games often require immediate interactions which have a TA of 6. According to the decision tree model, first-person shooter games have high sensitivity to latency, indicated by the red arrows and FPS labels in Figure 2.2.

In my research, we focus on first-person shooter games on a PC, with a mouse and a keyboard as input devices, and a monitor as a display; the mouse and keyboard are best for aiming [gee21] and are the most common peripherals for professional first-person shooter game players in esports.

2.2 Client-server Architecture

While underlying network game architectures can be peer-to-peer [SKH02], most use a client-server architecture with an authoritative game server. With the authoritative game server, most player actions are required to be approved on the server for their effects to change the game state. Moreover, the consequences of actions are often processed on the server and then synchronized to clients. For example, in a first-person shooter game, once a player pulls the trigger of a weapon, the client will notify the server about the action information and server will then decide if this shot is a hit or a miss before sending the updated game states to the client. This architecture is popular for network games since firewall rules can make it difficult for clients to connect to each other. Having a single server can also help a game scale with number of players. Architectures with a trusted server (e.g., run by the game publisher) can reduce the risk of players cheating.

Typically, the server is at a “well-known” IP address and port and is public – reachable by all clients. In some cases, a “server browser” setup allows game servers to register their individual IP addresses and ports with a sort of central server, allowing clients to connect to the central server and browse available game servers, the individual game servers begin differentiated based on configuration parameters (e.g., a certain game mode, map or latency). Once an individual game server is chosen, the central server provides the client with the game server IP address and port whereupon the client connects to the game server to play the game. Some network game architectures have one client act as the host to which the other clients connect. While these have peer-to-peer connections, they can still be viewed as a client-server architecture in that the host acts as a server, even though it also acts as a client for that player.

2.3 Latency Compensation

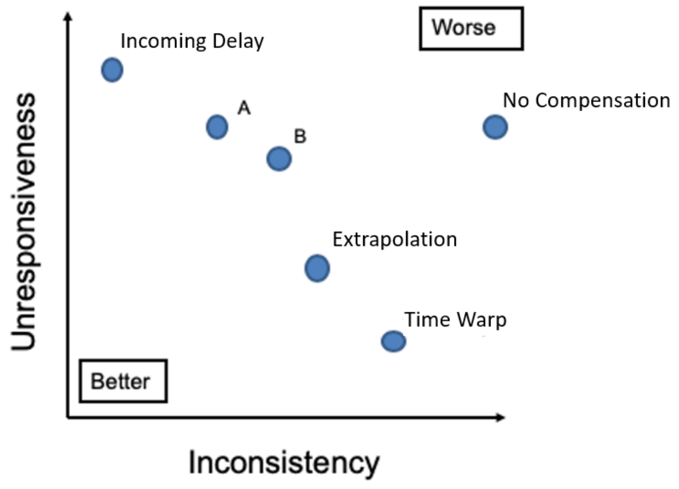


Figure 2.3: Trade-offs between responsiveness and consistency of latency compensation techniques

Latency affects the responsiveness and consistency of a game. Responsiveness denotes how fast the game gives feedback to player actions. Consistency denotes how tightly the game states synchronize between the clients. Latency compensation techniques have been widely used in most commercial games to mitigate the effects of latency on players. Latency compensation techniques often provide a trade-off between responsiveness and consistency, as indicated in Figure 2.3. In the figure, the x-axis is inconsistency and the y-axis is unresponsiveness. The closer the game to the bottom left corner, the better the responsiveness and consistency. Games with "no compensation" would be at the top right. Different game genres may use different sets of compensation techniques to find the best balance between responsiveness and consistency for their player actions. For example, real time strategy (RTS) games often use synchronization to ensure fairness between players and may trade-off worse responsiveness for better consistency. In contrast, first-person shooter games require low response time to help with aiming but may care less about consistency.

Table 2.2 includes eleven compensation techniques [LXC21] applied to first-person shooter games. The first column is the name of the technique. The second column is the usage of the technique in first-person shooter games. The third column is the impact of the technique in first-person shooter games. The forth column is whether we study the technique with the thesis.

The first technique in the list is latency concealment. Latency concealment visually masks latency from the client to the server so as to minimize the perception of unresponsiveness. In first-person shooter games, clients can show an animation and play sounds of movement and shooting before the action is resolved by the server. This technique may improve QoE. Since this technique likely does not change player performance, which is the main focus of our research, we do not study it.

Latency exposure gives a visual indicator of the magnitude of the latency from the client to the server. In first-person shooter games, the latency value is often shown on the screen. Players may change their weapon choice and play style upon observing the latency. However, latency exposure may not directly change player performance and QoE and so is excluded from our research.

Self-prediction predicts game state based on player input but before getting confirmation from the server. In first-person shooter games, movement and aiming are typically predicted on the client; the client does not wait on the server before rendering the result of these actions. Self-prediction is widely used in first-person shooter games. Self-prediction may change both player performance and QoE. We implement self-prediction on customized games made with Unity and study the tech-

Table 2.2: Latency compensation techniques

Technique	Use in FPS	Primary Player Impact	Study?
Latency concealment	Clients immediately show animations, sounds	QoE	No
Latency exposure	Latency value shows on screen	Play style	No
Self-prediction	Client predicts movement and orientation	Performance and QoE	**Yes**
Interpolation	Client predicts previous states for other avatars	QoE	No
Extrapolation	Client predicts future states of other clients	Performance and QoE	No
Speculative execution	Server predicts game state based on possible player inputs	Performance and QoE	No
Time warp	Server resolves actions based on previous client game states	Performance and QoE	**Yes**
Incoming delay	Equalize delay after receiving a message (synchronization)	Fairness	No
Outgoing delay	Adds delay before sending message (synchronization)	Fairness	No
Control assistance	Server or client assists aim and movement	Performance and QoE	No
Attribute scaling	Server changes game world attributes (e.g., AoE)	Performance and QoE	No

nique with user studies. We then incorporate the user study results into subsequent models and simulations.

Interpolation predicts past states for objects controlled by other players based on the current state and previously known states. In first-person shooter games, the client predicts previous states for other avatars in order to smooth visuals between server updates. Interpolation may improves QoE since avatar movement are smoothed, but since it may not affect player performance which is the main focus of our research, we do not study it.

Extrapolation predicts future states for objects controlled by other players assuming current behaviors continue. In first-person shooter games, extrapolation improves consistency between players and allows players to aim at opponents when shooting instead of aiming ahead of them. Extrapolation may improve both player performance and QoE with accurate prediction, but the effect of extrapolation relies on the accuracy of the prediction algorithm. Errors in prediction may actually degrade player performance and QoE. The variety of possible prediction algorithms also complicates its study. We do not study this technique in our research due to these challenges.

Speculative execution computes the game world state based on possible player input before it has actually happened and adopts this pre-computed state if/when the input is provided. Speculative execution may improve both player performance and QoE with accurate prediction on player input. To the best of our knowledge, speculative execution is not used in any of commercial first-person shooter games, perhaps due to the difficulties in implementation. Because its lack of use in first-person shooter games, we do not include this technique in our thesis.

Time warp rolls back game state on the server to when the player action occurred on the client, applies the action, then rolls the game state forward to the current time. In first-person shooter games, the server decides whether a player hits a target based on the previous game state when the player fired the shot. This technique improves game responsiveness and lets players aim directly at the target as if there is no latency. Time warp may improve both player performance and QoE and is widely used in commercial first-person shooter games. We implement time warp on customized games made with Unity and evaluate its impact on player performance and quality of experience with user studies. We then incorporate the user study results into subsequent models and simulations.

Incoming time delay buffers player actions before applying them so that actions arrive (and are applied) at all clients simultaneously. Outgoing delay works similarly to incoming delay but adds delay before sending a message instead. Both incoming delay and outgoing delay improve fairness but may not improve player performance and QoE since they may increase latency. These two techniques are normally not used in multi-player first-person shooter games. We do not study incoming time delay and outgoing time delay because they may not improve player performance and QoE.

Control assistance adjusts the outcome of player input to accommodate for inaccuracies due to latency. In first-person shooters, control assistance normally refers to aim assistance. It improves performance and QoE but it is not typically used in games with a PC with mouse. Since our research focus is on first-person shooter games on PCs with a mouse, we do not include it.

Attribute scaling increases or decreases numeric attributes of objects and other game world parameters to adjust game difficulty so as to make player actions easier to complete with higher latency. For example, in first-person shooter games, attribute scaling can increase target size and lower target moving speed with higher latency. This should improve both player performance and QoE since it lowers game difficulty. However, attribute scaling is not typically used in multi-player games because clients may have different latency values, so we do not include this technique in our research.

Some latency compensation techniques may change player style or improve fairness but may not improve player performance and QoE (e.g., latency exposure). Some techniques may only improve QoE but may not improve performance (e.g., latency concealment). Some techniques such as control assistance are typically used in console games. Given this, we focus on techniques that improve both performance and QoE and are widely used in first-person shooter games on PCs - self-prediction and time warp - with user studies on both actions. We then incorporate the user study results into subsequent models and simulations.

Sections 2.3.1 and 2.3.2 introduce the two techniques - self-prediction [LSGH17] and time warp [SG13] in detail.

2.3.1 Self-prediction

Self-prediction predicts game state based on player input, but before getting confirmation from the server. Self-prediction is a natural technique to use for network game programmers since most clients run a full-featured game engine able to incorporate player input and compute game object interactions (e.g., physics, including collisions), and doing so can provide immediate feedback for the player.

Figure 2.4 depicts an example of self-prediction. A time t_0 on the left, the player has a view of the game world that is consistent with the server's (not shown). At time t_1, the player has provided some game input (e.g., press the right arrow key) in order to move the green avatar to the right. The client assumes that this movement will be allowed by the server and renders the world with the green avatar in the predicted location. Once the server receives and then responds to the player input

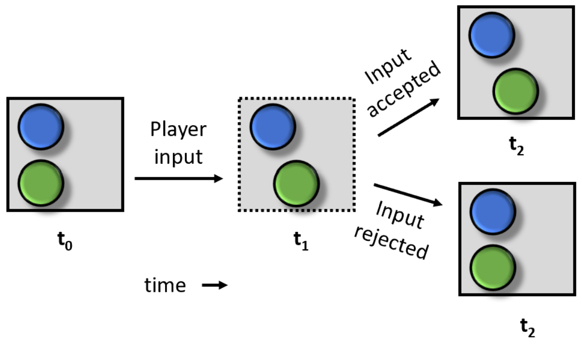


Figure 2.4: An example of self-prediction.

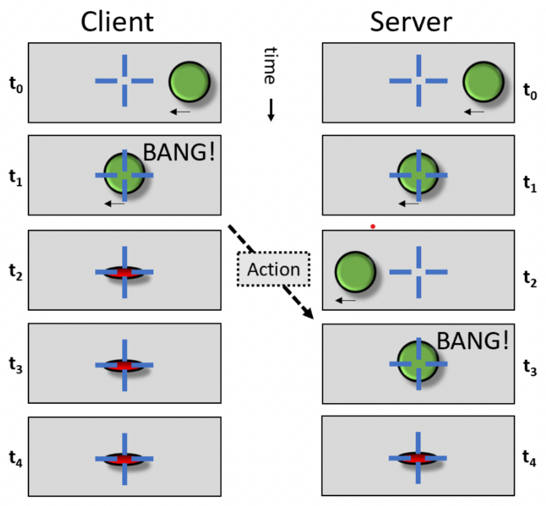


Figure 2.5: Time warp

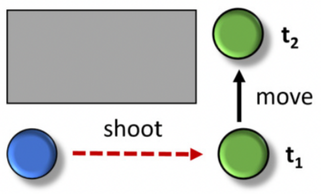


Figure 2.6: Shot around corner

at time t_2 there are two possibilities: in the first case, shown at the top, the server has accepted the input and the green avatar's new position is confirmed; in the second case, shown at the bottom, the server has rejected the player input (e.g., if the avatar is blocked by another object unknown to the client) and the client renders the world as specified by the server.

2.3.2 Time Warp

Time warp rolls back game state on the server to when the player action occurred on the client, applies the action, then rolls the game state forward to the current time.

Figure 2.5 depicts an example of time warp. The figure shows the game world for a shooter game on a Client and the Server, with time advancing from top to bottom. The player on the client is shooting at a green avatar that is moving right to left, with the "plus" sign in the middle representing a weapon reticle. At time t_0 at the client, the green avatar is to the right of the reticle, moving into the reticle at time $t1$ where the player pulls the trigger and that action is sent to the server arriving just after time $t2$. Meanwhile, on the server, the green avatar moved past the reticle at time $t1$ and has continued right at time $t2$. When the action arrives at the server, the server "warps" time back to when the action occurred at time $t1$, applying the action to the world representation at that time.

However, resolving an event in the past and rolling it forward may cause already rendered client game states to be inconsistent with the new view. This is a well-known artifact of some shooting games, known as "shot around the corner", shown in Figure 2.6. At time t_1, at the blue avatar's client, the green avatar is in sight and the blue player fires. However, by time t_2, the green avatar has reached a safe position around the corner from the blue player and cannot be targeted. However, with time warp, the server, upon receiving the blue player's action, rolls back time to the green avatar's position at time t_1 and applies the action. This hits the green avatar. Rolling the game world forward, with the green avatar hurt or killed, may feel like being "shot around the corner" for the green player.

Time warp is often called *latency compensation* or *lag compensation* in some papers and, more often, in online blogs and player posts.

Chapter 3

Related Research

This chapter presents previous research in related topics - gaming actions (Section 3.1), impact of latency (Section 3.2), first-person shooter games (Section 3.3) and players (Section 3.4).

3.1 Gaming Actions

Game actions are the main actions player take within game. There are mainly two game actions in first-person shooters - navigation and selection. The navigation action is moving an avatar to a position to shoot an opponent or avoid been shot. The selection action is pointing to a moving or a stationary target and clicking the "fire" button. This section presents related work about these two actions.

3.1.1 Selection

Selection refers to a player's ability to click on a target with an input device. Long and Gutwin [LG18] study the effects of latency on selecting a moving target. They find target speed directly affects the impact of latency, with fast targets affected by latency as low as 50 ms, but slower targets resilient to latency as high as 150 ms. Long and Gutwin [LG19] measure selection time for different sized moving targets. They find that the effects of delay are exacerbated by fast target speeds. Claypool et al. [CER17] investigate selecting a moving target with a mouse in the presence of latency. Their analysis showed target selection time is impacted exponentially by latency and target speed for constant-velocity targets. Janzen and Teather [JT14] conduct a study with 12 users playing a target selection game with frame rates from 15 to 60 f/s and latencies from 0 to 100 ms. The work revealed that in the lowest frame rate conditions, latency does not significantly affect performance. However, Latency degrades player performance significantly with decent frame rate.

There have been many studies on selection related tasks, while almost all of them are for 2D objects. In general, while these approaches and previous work have helped understand latency and the selection action, they generally have not applied a model to the data gathered, or if they have, the models are for average (expected) values and not the distributions of the values. In our research, we model the distribution of elapsed times for selection. With the models, we can simulate the range of player target selection actions which can help us better understand the impact of latency on first-person shooter players.

3.1.2 Navigation

Navigation refers to controlling and moving an avatar, for example, steering a vehicle in racing car games. Drury [Dru71] simulates vehicle steering on a path with straight lines and circles and confirmed a linear relationship between the average velocity and the width of the tolerance band. Accot and Zhai [AZ97] propose the Steering Law, a predictive model of human movement, concerning the speed and total time with which a user may steer a pointing device (e.g., a mouse) through a tunnel presented on a screen (i.e., with a bird's eye view of the tunnel), where the user travels from one end of the path to the other as quickly as possible while staying within the confines of the path. The total time is a function of path width and index of difficulty. The authors later evaluate the model with 5 different input devices on 2 steering tasks [AZ99], and the model has a 0.98 or greater R^2 for all devices. Accot et al. [AZ01] study the scale effect in the framework of the Steering Law, and found a significant scale effect in path steering performance. There are extensions to the Steering Law since then. Kattinakere et al. [KGS07] enrich the model with the condition of steering through constrained and unconstrained paths in above-the-surface layers. Zhai et al. [ZAW04] extend the Steering Law to Virtual Reality (VR) games by examining the applicability in a VR locomotion task. Friston et al. [FKS16] conduct a user study on pointing and steering tasks and found that latency low as 16 ms can affect player performance and the effect is non-linear.

Although the previous work is helpful in understanding the steering task - moving an avatar to a designated position following path constraints - navigation in first-person shooter games, where the purpose of movement is to get into position to shoot opponents or avoid being shot, remains unexplored. Navigation in first-person shooter games generally has no specific path constraints and the performance metrics are also different. To the best of our knowledge, there is no such work evaluating the impact of latency on player movement in first-person games.

3.2 Impact of Latency

This section introduces related research on latency and games, where Section 3.2.1 focuses on local latency - the latency induced by hardware like input devices and displays; Section 3.2.2 focuses on network latency and Section 3.2.3 focuses on quality of experience. Section 3.2.4 describes related work in three latency compensation techniques.

3.2.1 Local Latency

There are several works that have characterized the effects of local latency on game player performance [ISGS15, CER17, ERC18, LG18, LG19]. These papers have generally focused on a single player action, isolating and analyzing the effects of latency without considering the broader set of interactions of a typical game [LKS$^+$21a].

Claypool et al. [CER17] show local latency and target speed exponentially impact target selection time. Ivkovic et al. [ISGS15] find significant main effects for local latency on target tracking and acquisition tasks, both with and without latency compensation (aim assistance), and with a greater effect for higher target speeds. Long and Gutwin [LG18] find target speed directly affects target acquisition with latency, with fast targets affected by latencies as low as 50 ms but slower targets resilient to latencies as high as 150 ms. Eg et al. [ERC18] show local latencies from 40 ms to 400 ms negatively affect player performance for moving target selection, but that performance does not co-vary with self-reported game skill. Long and Gutwin [LG19] compare the effects of local latency across 4 different gaming devices, demonstrating that latency affects each device differently for moving target selection [LKS$^+$21a]. Durnez et al. [DZC$^+$21] show that while player performance and quality of experience degrades with latency, exergame actions are fairly tolerant of even hundreds of milliseconds of latency.

Understanding the effect of local latency can help understand how improvement on systems can benefit players. Better systems can help reduce local latency. For example, frame rate has been crucial in game-related analysis (e.g., analysis of CPU performance) and affects player performance. Higher frame rate helps render feedback faster and provides a lower local latency. Previous studies generally focus on frame rate and resolution as independent variables in their analysis. Claypool and Claypool [CC07] show that player actions that require precise, rapid response, such as shooting, are greatly impacted by degradation in frame rates below 30 f/s for a first-person shooter game [LKS$^+$21b].

While helpful for ascertaining the impact of system level configurations on game players, previous research do not provide a model on how frame rate affect player performance. However, with our model in local latency and player performance, we are able to predict how change in system configurations affect performance of first-person shooter players through simulations.

3.2.2 Network Latency

Numerous studies have detailed the effects of network latency and games [PW02, Arm03, QML$^+$04, DWW05, FRS05, CC06, AJG$^+$13, HCW$^+$14, HFPG16]. Most of these studies utilize commercial games with controlled amounts of network latency in a laboratory environment, rather than observing players in gaming actions [LKS$^+$21b].

Dick et al. [DWW05] show via a survey that players generally think about 120 ms is the maximum tolerable latency for a network game, regardless of game genre, but their user study shows players find 150 ms acceptable for the two First-Person Shooter games and racing game tested. Pantel and Wolf [PW02] show latencies of about 100 ms can affect car racing games. Fritsch et al. [FRS05] find players of the role-playing game Everquest 2 can tolerate hundreds of milliseconds of network latency, while Hoßfeld et al. [HFPG16] find players of the casual game Minecraft are insensitive to network latencies of up to 1 second. Howard et al. [HCW$^+$14] indicate that for online cooperative games, a player can be affected by latency for a teammate due to cascading effects on the game outcome. Based on this body of work, Claypool and Claypool [CC06] suggest game action sensitivity to latency can be classified by precision and deadline – higher precision and tighter deadlines mean more sensitivity to latency. Halbhuber et al. [HKS^{+}22] investigate the effects of auditory latency (0 - 500 ms) on experienced first-person shooter players via a 24-player user study. They find that player experience degrades significantly with latency and high skill players are more impacted by latency.

However, while such works have been instrumental in better understanding the effects of network latency on players of online games, they do not necessarily represent the effects of latencies on player performance since they combine local latency with network latency. As such, they typically deal with a high range of latencies which are avoided by computer gamers. Also, the results in the previous work were on specific games and the results may not generalize to other first-person shooter games. Moreover, most network games have latency compensation techniques that help mitigate the effects of network latency [LXC22] which can cause inaccuracy to the results. Instead, we build games without latency compensation techniques to evaluate the impact of network latency accurately.

3.2.3 Quality of Experience (QoE)

In game related area, Quality of experience (QoE) is a measure of player satisfaction of a player's experiences in a game round. More broadly, it can be a measure of enjoyment and pleasure. The effect of latency on a gamer's quality of experience (QoE) is widely investigated in many papers [AGC^{+}18]. Quax et al. [QBV^{+}13] show that latency has different influence in different game genres; for instance, first-person shooter (FPS) games are more sensitive to delay than platform games. Also, Beyer et al. [BM14] show that within the same genre, depending on the game rules and implementation, the sensitivity of games may differ significantly. Moreover, Schmidt et al. [SZM17] show that even within the same game, different scenarios might lead to different latency sensitivities. Sabel et al. find that player's adaptation to latency can mitigate the impact of latency on players' quality of experience [SSSZ$^+$18].

Quality of experience is an important metric in gaming. Players might quit the game when they perform well but have a poor quality of experience. We access and analyze the impact of latency on player's quality of experience in our user studies of games and game actions.

3.2.4 Latency Compensation Techniques

As indicated in Section 2.3, latency compensation techniques are software algorithms that run on the game client or game server (or both) and have been widely used in first-person shooter games to mitigate the impact of network latency on players. Such techniques might improve responsiveness or increase consistency (or both), but often sacrifice one for the other (i.e., increased consistency at the cost of reduced responsiveness or vice versa).

Much of the previous work in latency compensation techniques propose new techniques or improve upon existing techniques but with limited evaluation. Jefferson [Jef85] first proposed virtual time as a paradigm for distributed computation, with time warp as an implementation. Although multi-player games were not identified as a use at that time, distributed discrete event simulations were. Mauve [Mau00b] describes how timestamps and time warp can be used to overcome mistakes made using extrapolation. In particular, an extrapolated state update may miss a key event, such as a player being killed that time warp can roll back to correct. Mauve [Mau00a] and Mauve et al. [MVHE04] provide a formalization of time warp in the context of continuous, interactive media, such as computer games. Jiang et al. [JSB05] mention time warp in their survey as a means to overcome inconsistencies caused by prediction. Savery et al. [SGG10] mention the use of time warp and the "shot around the corner" problem in commercial first-person shooter games, whereby time warp can undo a players move to a safe location to instead being damaged. Lee and Chang [LC15, LC17] describe and evaluate the "shot around the corner" problem. Subsequently, they [LC18] describe how commercial first-person shooters provide a limit on how far back a server rolls back time and propose an advanced time warp technique to prevent "shot around the corner" whereby a client can identify the player is currently safe (using their local time) and prevent rollback. Sun and Claypool [SC19] implement and evaluate time warp for a cloud-based custom game streaming system running a 2d arcade game. They test 5 latency values on 30 players finding that time warp can mitigate the effects of latency on player performance. Although this work helps evaluating time warp, the results may not generalize to first-person shooter games. Brun et al. [BSB06] mention self-prediction as it affects game state consistency in multiplayer games. Burgess and Shelly [SK05, BK06] describe the use of a software design pattern based on the notion of "optimism" where a prediction on the client is assumed to be valid until the server does the official computation and returns the results. They provide additional techniques for synchronization and consistency checking. Chen et al. [CCC$^+$07] use a form of self-prediction called an "echo" to immediately show the player the effects of an action, even if additional delay is incurred before the official confirmation. Wu and Ouhyoung [WO00] study "look-ahead" algorithms for 3d, head-mounted displays that predict object position and orientation. They study algorithms with different prediction complexities – simple to more complex. Also for a virtual environment with a head-mounted display, Tumanov et al. [TAS07] describe predictions of "poses" for players based on the position and orientation in the motion space. Le et al. [LSGH17] capture movements of the hand in a touch interface, use these movements in a neural network, and predict the location of future touch positions. Similarly, Antoine et al. [AMC18] use high-frequency data gathered from a computer mouse to predict the future velocity and position. They provide specific context for fairness in multi-player games, where players may have different amounts of latency. Halbhuber et al. [HHS21] investigates if latency can be compensated with data-driven ANNs predicting user inputs within a live first-person action game. They trained the system with 24 participants and evaluated the system in a second user study with 96 participants. The results show that the prediction technique significantly improves player performance and experience.

Though the work may help understand the impact of the compensation techniques, the evaluations do not generalize to FPS games. To the best of our knowledge, none of them provide a model with the techniques on first-person shooter game players. In our research, we study self-prediction and time warp, and model the effects of latency on player performance self-prediction and/or time warp.

3.3 First-person Shooter Games

First-person shooter games have been a popular topic in game related research. Amin et al [AJG$^+$13] show player experience defines and determines the sensitivity to latency for the FPS game Call of Duty, with competitive gamers more adept at compensating for impaired conditions. Armitage et al. [Arm03] estimate the latency tolerance threshold for Quake 3 to be about 150-180 ms. Quax et al. [QML$^+$04] show players for UT2003 that latency and latency jitter under 100 ms can degrade player performance and quality of experience. Lee and Chang [LC15] evaluate how interpolation in Counter-Strike: Global Offensive (Valve, 2012) improves player accuracy.

They test two latency values - 0 and 150 ms - with 4 users and find that interpolation can increase player accuracy significantly. Spjut et al. [SBB+19] show a reduction in 30 ms of latency benefits first-person targeting tasks more than frame rates above 60 f/s. Claypool et al. [CCD06] find frame rate has a marked impact on both player performance and game enjoyment while frame resolution has little impact on performance and some impact on enjoyment for a first-person shooter game

While beneficial in understanding first-person shooter games, these papers generally focus on a specific game, and the results may not generalize to other first-person shooter games with different configurations. In our research, we study and model gaming actions as building blocks, simulate first-person shooter scenarios and validate the simulations. Then, we explore general first-person shooter games with the validated simulations.

3.4 Players

This section includes related work regarding game players, especially reaction time (3.4.1) and player skill assessment and evaluation (3.4.2).

3.4.1 Reaction Time

Reaction time refers to the speed in responding to a stimulus. Kosinski [Kos08] finds measured reaction time might differ because of the purpose of a study, age of participants and types of stimulus. The human benchmark [Ben] reports that the average human reaction time is 284 ms based on 81 million users. Related work shows that gamers tend to have faster reaction time than non-gamers. Richardson et al. [Ric14] depict that gamers who play over 4 hours of video game on average each week have about 45 ms shorter reaction time than non-gamers. Thornton and Gilden [TG05] point out that factors such as fatigue and sequential effects are generally assumed to be of negligible impact and therefore ignored in reaction time measurement.

Importantly, research shows response-time distributions are not Gaussian (normal) distributions but rather rise rapidly on the left and have a long positive tail on the right. Reaction-time distributions are similar to the ex-Gaussian distribution [Luc86], which is a convolution (mixture) of a Gaussian and an exponential distribution that has been shown to fit empirical reaction time distributions well [BS99]. This distribution has three parameters. The mean and the standard deviation of a Gaussian body are described by mu (μ) and sigma (σ), respectively [HPM91]. Tau (τ) describes both the mean and the standard deviation of the exponential component. Whelan [Whe08] simulates a distribution of reaction time to demonstrate the distribution shape but not to predict reaction time for certain group of people on certain task.

Although human reaction time has been a well-studied topic with a long history, there are fewer works on the correlation of reaction time and player skill in first-person shooter games. In our research, we assess the role of reaction time in first-person shooter player performance.

3.4.2 Player Skill

Game players generally want low network latency to maximize their chances of winning, but how much latency affects players with different levels of skill is not well-known. There is some, albeit limited, work investigating the effects of latency on players with consideration to players grouped by skill. Claypool [Cla18] triage 51 users into three skill groups, have them play a target selection game with latency and show that higher skill players are resilient to performance degredations for latencies above 350 milliseconds. While useful for some interactive applications, these delays are much higher than many gamers experience, and the game studied is for 2D target selection only. Amin et al. [AJG^{+}13] query two users with different amounts of skill after playing the first-person shooter game Call of Duty with latency and infer that higher-skilled players notice even small amounts of latency but are able to compensate for it better than lower-skill players. As the authors themselves note, their small sample size makes their objective evaluation hard to generalize. While useful for understanding the impact of latency on the game Call of Duty, the work did not provide a model, and the results may not represent the players in other first-person shooter games. Dick et al. [DWW05] separate 8 users into two teams, have them play four first-person shooter games with network latency and jitter to study the factors that impact players, finding skill impacts score but not mean opinion score (MOS). While useful for understanding the factors that affect players, player skill is differentiated by self-report score only with a total sample of only eight players total.

Although the related work might help understand latency and a specific first-person shooter game, there is no model on player skill and in-game performance. The results may not generalize to other first-person shooter games with different maps, weapons, target sizes and speeds.

In our research, we separate player skill by their in-game performance - players with top third performance are higher skill players and bottom third performance are lower skill players. We then model elapsed time and navigation windows separately for each skill group. With the models, we can simulate first-person shooter scenarios for players with different level of skill.

Chapter 4

Proposed Approach

Our goal is to develop a thorough understanding on the impact of latency on first-person shooter game players. The methodology is depicted in Figure 4.1, where the boxes show the main steps of our methodology, and the arrows indicate progression from one step to the next. To explore the space of first-person shooter games without iterating over all game and system configurations (e.g., weapons, maps, latencies), we first break first-person shooter games into two fundamental actions that are the basis for player interactions in the game: target selection and navigation. These two gaming actions can be seen as the building blocks for first-person shooter games that are impacted by latency and can be integrated into more complicated game scenarios with different game configurations.

We conduct user studies to obtain data on player performance for the isolated actions under different conditions (Sections 5.1 and 6.1). Data from the user studies are directly used in our modeling of the gaming actions, but is also a contribution to the game development and research communities by providing insights into the effects of latency on game interactions.

We build models from the user study data to explain the relationship between

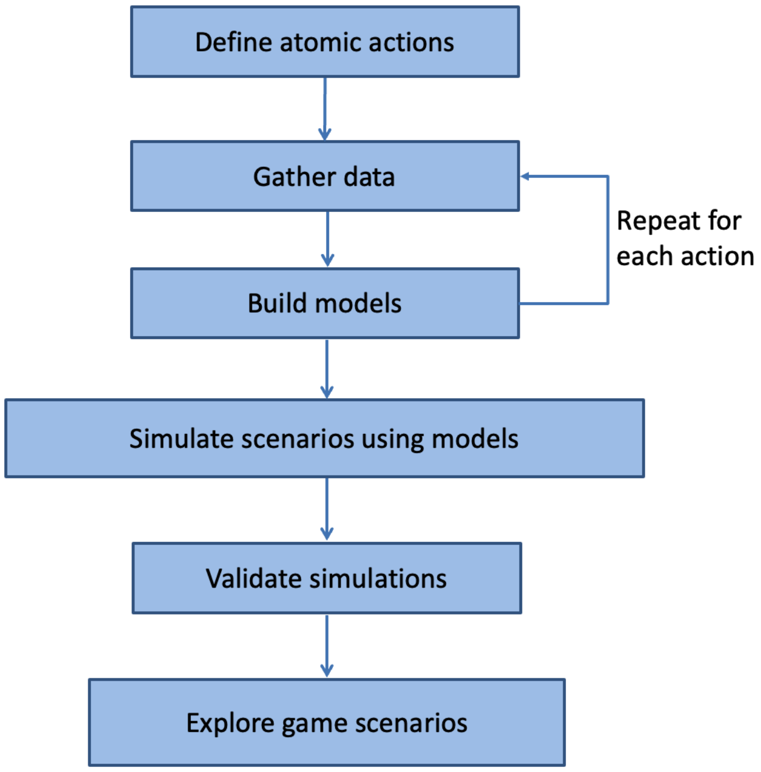


Figure 4.1: Methodology

latencies, player skills, game configurations and player performance (Sections 5.4 and 6.2.6). The models are used directly in simulations for first-person shooter games with different types and values of latency, latency compensation techniques, target speed, target size, map size and player skills in Chapter 7.

We then validate our simulations by comparing the simulated data with empirical data from first-person shooter games in Chapters 8 and 9. Finally, we explore the first-person shooter game space with additional simulations, varying latency types and values, weapon choices, latency compensation techniques and player skills in Chapter 10. The simulations reveal how player performance is affected by latency conditions across a broad set of first-person shooter game conditions.

Chapter 5

Navigation

The navigation action is used by players to move an avatar in position to shoot opponent(s) or to avoid being shot. To build the model for the impact of latency on player performance for navigation, we collect data with a user study where participants play a custom game that isolates the navigation action. Local latency, network latency and latency compensation techniques are controlled. Objective performance and subjective QoE are collected via game logs and surveys after game rounds. With player performance and QoE data collected, we analyze how player performance and QoE change with latency. We fit the distribution of time being seen by the seeker or seeing the hider with mathematical models. Finally, we get the models for time windows related to local latency, network latency and compensation techniques.

This section presents results from two user studies that evaluate the impact of latency and latency compensation techniques on first-person navigation using a custom "hide and seek" game that isolates avatar movement in first-person shooter games. Analysis of the results shows pronounced benefits to player performance (score and positioning to hide/seek), with subjective opinions on Quality of Experience following suit. Time warp and self-prediction both mitigate the effects of latency, and when applied together, can eliminate the effects of latency on player performance. We derive an analytic model for the distribution of the time intervals with opponents in sight, which can be combined with target selection to model and simulate player performance in a full-range of first-person shooter games.

5.1 Navigation Methodology

In order to assess the effects of latency on navigation in a first-person game, we built a custom game that isolates the navigation action, added controlled amounts of local and network latency, recruited participants for user studies, and measured player performance and quality of experience.

5.1.1 Hide and Seek Game

We designed and implemented a custom first-person game in Unity that isolated the action of first-person navigation in a first-person shooter-type setting. Our game is a two-player game called "hide and seek", shown via screen shot in Figure 5.2. At any given time, one player is the hider and the other player is the seeker. The goal for each player is the spot the opponent's avatar when the seeker, and hide from the opponent's avatar when the hider. These roles capture typical interactions in first-person shooter games where a player navigates to get an opponent in sight to shoot at and, similarly, navigates to hide from an opponent while being shot at. In our game, the roles switch every 2-6 seconds – this abrupt and random switching of roles captures the dynamics in a first-person shooter game where a player is both hunting (trying to shoot an opponent) and hunted (trying to avoid being shot) in a short amount of time. Anecdotally, several users said, unprompted, that the game tension felt like a first-person shooter game, albeit without the weapons.

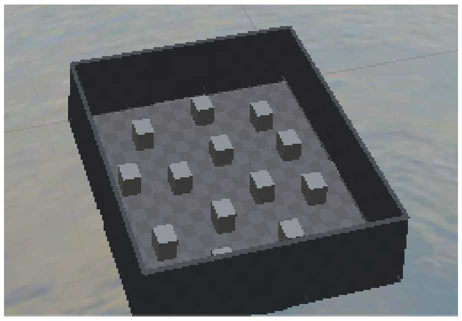


Figure 5.1: Hide and Seek map.

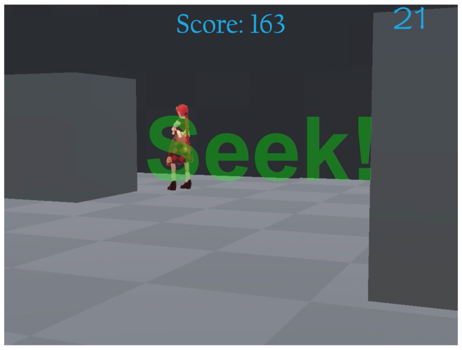


Figure 5.2: Hide and Seek screenshot.

The update rate for the game engine is fixed at 50 frames per second. In a frame, if the seeker can see any part of the hider, the seeker earns a point; otherwise, the hider earns a point. A game round terminates after 40 seconds. While the roles are switched randomly, we ensure each player is the hider for exactly half the time (i.e., 20 seconds per round) and the seeker for half the time. When the timer runs out, the player with more points wins. Each frame, the game logs whether the hider or the seeker gets points, the running score for both players, and the keyboard and mouse actions.

Hide and Seek has one map, depicted in Figure 5.1. The map is a single, square room, 36 meters in length and width, with multiple obstacles to mimic maps in typical first-person shooter games where terrain can play a role in the combat. The player avatars spawn at a random location on the map near, but not currently in view of, the opposing player. Upon spawning, the game provides a countdown for each player until the round starts. Figure 5.2 shows a Hide and Seek screenshot where the player is currently a seeker. The semi-transparent green "Seek!" or red "Hide!" message in the middle of the screen informs the player of their current role. The score and timer are shown in blue at the top of the screen. In the screenshot, the opponent is in sight at that moment, thus the player is gaining points as long as the opponent remains visible (or the roles switch).

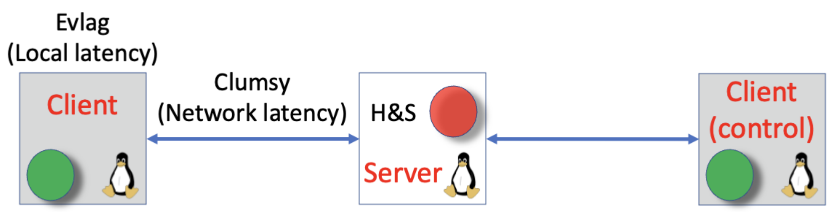


Figure 5.3: Hide and Seek computer configuration.

Hide and Seek has a client-server architecture typical of most network games where the authoritative server keeps the master world state and communicates state updates to the clients.

We conduct two studies that use the Hide and Seek game and a client server architecture - study A and study B.

5.1.2 Navigation Testbed Setup

We setup the game for our user studies in a dedicated, on-campus computer lab. The testbed setup is depicted in Figure 5.3. The server hosts the game and is connected via high-speed LAN to the clients. The clients and server are Alienware PCs with Intel i7-4790K CPUs @4 GHz with 16 GB RAM and an Intel HD 4600 graphics card. The clients are each equipped with a gaming mouse and monitor so as to minimize local system latency and maintain consistency. The clients have a 25" Lenovo Legion monitor running at 1920x1080 pixels displayed at 16:9 and 240 Hz, with AMD FreeSync and a 1 ms response time. The mouse is a Logitech G502 12k DPI with a 1000 Hz polling rate. The clients and the server run Ubuntu 20.04 LTS, with Linux kernel version 5.4.

To provide for accurate assessment of the latencies user experienced in the study described here, the base system latency was measured on the test system. The measurement method is depicted in Figure 5.4. A high-frame rate camera (a Casio

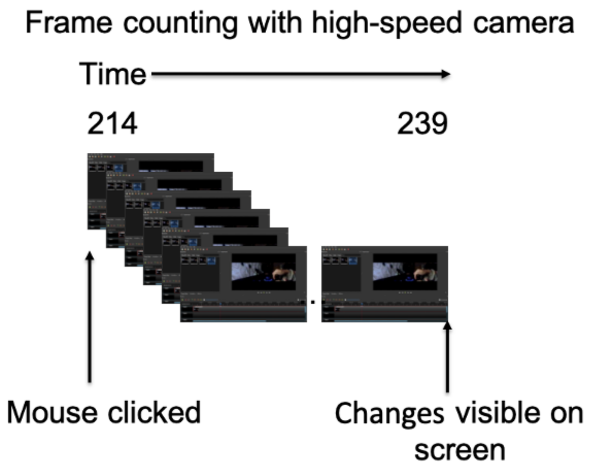

Frame counting with high-speed camera
Time
214
239
Mouse clicked
Changes visible on screen

Figure 5.4: Measuring local latency

EX-ZR100) was setup completely external to the game system and filmed a user at 1000 f/s, capturing the moment the mouse button was clicked. By manually examining the video frames, the frame number when the mouse was clicked (finger bent, frame number 214 in Figure 5.4) is subtracted from the frame number when the output was visible based on the user input click (frame number 239 in Figure 5.4), giving the base system latency (25 milliseconds in Figure 5.4). The measurement method was repeated 10 times on our system, resulting in an average base latency of 22 milliseconds (ms), with a standard deviation of 5 milliseconds.

Table 5.1: Testing conditions for user study A.

Parameters	Values
Local Latency	25, 100, 175 ms
Network Latency	0, 100, 200 ms
Latency Compensation	None

Local latency delays all input until resulting rendered output, whereas network latency delays receipt of the player's action at the server and subsequent server response to the client. Since the Hide and Seek server is authoritative, the client

Table 5.2: Testing conditions for user study B.

Parameter	Values
Local Latency	25 ms
Network Latency	0, 175, 350 ms
latency compensation	None, Self-prediction, Time warp, Both

cannot update the position of an avatar until the server response has arrived. Thus, for Hide and Seek (as for all client-server games without latency compensation), local latency manifests similarly to network latency. Player movement input until resulting avatar movement is seen on the screen is delayed by at least the sum of the local latency and the network latency.

Two user studies were conducted to study the impact of latencies and compensation techniques on players with the first study - study A focuses on network latency and local latency. Our intent is to assess local latencies over ranges that might typically be found in personal computers, which range from about 25 milliseconds for a fast gaming system, are around 100 milliseconds for a typical computer system, and can be 175 milliseconds for a slower gaming system [ISGS15]. In order to test the effects of latencies above the baseline 25 ms, additional latency was added to all keyboard and mouse user input with a custom C program we wrote called *EvLag* [LC21a]. EvLag is a stand-alone executable that adds a constant amount of latency to any input device in Linux using `evdev`, interfaces that generalize raw input events from device drivers as character devices. EvLag accesses the devices via `libevdev`, a user-space library that abstracts I/O calls through a type-safe interface. When enabled, EvLag intercepts and enqueues all input events from a selected device and, after the specified delay, dequeues and delivers the events. Timing in EvLag is maintained via the real-time clock drivers for Linux, accessed through `/dev/rtc`, giving fine-grained time resolution (less than a millisecond) for control.

Given our client has an average local latency of 22 milliseconds, EvLag adds either 3, 78 or 153 milliseconds of latency for resulting total local latencies of 25, 100 and 175 milliseconds, respectively, as shown in Table 5.1. Note that 25, 100 and 175 milliseconds are average values since the underlying system does not have a fixed, constant latency, consistent with all personal computers that do not have real-time control over devices, operating system scheduling and game computations.

Similarly, our intent is to assess network latencies over ranges typically experienced by PC network game players, which can be near 0 milliseconds for a local area network (LAN) game, 100 milliseconds for a reasonable Internet connection, and 200 milliseconds for a slower Internet connection [opt20]. We added network latency to the server uplink and downlink equally using Linux tc with Netem[1] – a network control tool. The total network latency added to the client was either of 0, 100, or 200 milliseconds as indicated in Table 5.1. The total latency is the round trip time between the client and the server - it equals to the summation of the uplink latency and downlink latency. For example, if the total latency is 100 ms, the uplink latency and downlink latency are equal at 50 ms in our user studies.

The second study - study B - focuses on latency compensation techniques. Latency compensation techniques can mitigate the impact of latency on players. Self-prediction and time warp are one of the most common compensation techniques applied to first-person shooter game. With self-prediction, the client predicts self movement and orientation. With time warp, the server resolves actions based on previous client game states when the actions triggered on the client. In user study B, we studied four latency compensation techniques conditions - none, only self-prediction, only time warp, both self-prediction and time warp. The total local latency is 25 ms across all game rounds. There are 3 network latency values - 0,

[1]`https://wiki.linuxfoundation.org/networking/netem`

175 and 350 ms. When the network latency is 0, there is no latency compensation. The latency values and compensation techniques tested are listed in Table 5.2. In commercial first-person shooter games played over the Internet, to avoid cheating, the server is always the one who makes decision on game outcomes instead of the clients. Our games take players in a closed lab and cheating is not an issue. In our Hide and Seek game, with time warp, the client calculates the outcome of the frame based on the game state on the client. The client then notifies the server of the outcome. Upon receive the notifications, the server updates the game state according to the outcome and synchronize to the clients. To avoid inaccuracy caused by inconsistency between clients and server during role switching synchronization time period (e.g., the player has been switched to seeker on server but remains hider on the client), the data during role switching synchronization time period is removed from further analysis. In both our implementation and commercial implementations, the players aim directly at the target in order to hit the target and players need to wait a round trip time to see the outcome displayed on the screen.

5.1.3 Navigation User Study Procedure

Before the launch of the formal user studies, a pilot study with 3 volunteers was conducted in order to test the viability of the procedure and tune the study settings. The pilot study results helped adjust round length, map size and layout, number of rounds, latency values and user instructions.

Interested participants first filled out a screener questionnaire with questions on first-person shooter game-related experience to help distinguish player skill.

The IRB-approved user studies was conducted during the COVID pandemic, so everyone wore masks and respected social distancing requirements. Before starting each study, all computer devices and touched surfaces were carefully sanitized. Selected users were invited to the lab at a pre-set time. Users then signed a consent form and positioned themselves at the test computer.

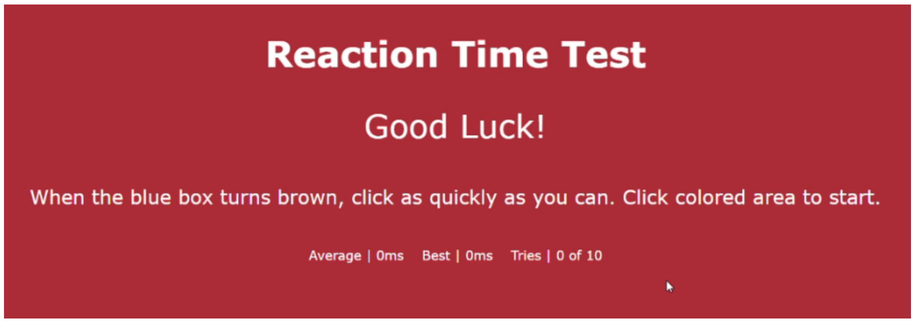
Before any sessions started, users first completed a reaction-time test written in Javascript and launched via a Chrome Web browser on the laptop. In the test, users click anywhere on the screen to start in Figure 5.5. Users then waited for a screen color change in Figure 5.6, and clicked the mouse as quickly as possible in Figure 5.7. The reaction time is then recorded and displayed on the screen as indicated in Figure 5.8. The screen also displays average and the best reaction time across the trials. Then users click anywhere on the screen for the next trail. Users did this 10 times (the total test time was about 30 seconds) before starting the game sessions. The average of the 10 values provides a measure of reaction time.

To ensure the consistency of play conditions across different users and the reliability of the between-subject results, all participants played against the same opponent, who served as a control. Network latency and local latency were only applied to the participant's avatar and not to the control avatar, as indicated in Figure 5.3.

In both of the studies, Users started by playing a practice round without any added latency to get familiar with the game. This data was not analyzed.

In the user study on network latency and local latency (user study A), users next played additional rounds, each with a different local latency (25, 100, or 175 milliseconds) and network latency (0, 100, or 200 ms), randomly shuffled. Each combination of local latency and network latency was repeated 3 times, for a total 27 rounds (plus the practice round) for each user.

In the user study on latency compensation techniques (user study B), users next played additional rounds, each with a different network latency (0, 175, or 350 ms) and latency compensation techniques (none, self-prediction only, time warp only,

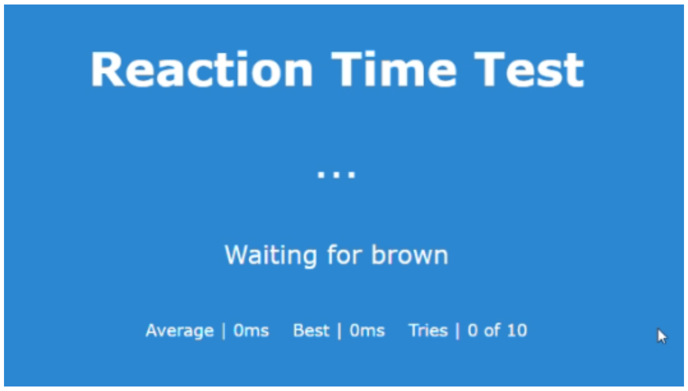

Reaction Time Test

Good Luck!

When the blue box turns brown, click as quickly as you can. Click colored area to start.

Average | 0ms Best | 0ms Tries | 0 of 10

Figure 5.5: Reaction trial - step 1

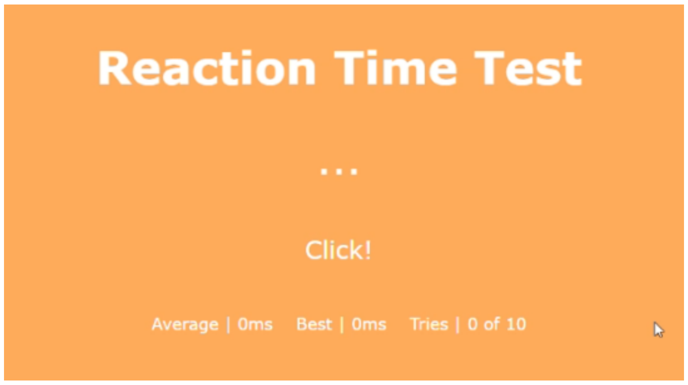

Reaction Time Test

...

Waiting for brown

Average | 0ms Best | 0ms Tries | 0 of 10

Figure 5.6: Reaction trial - step 2

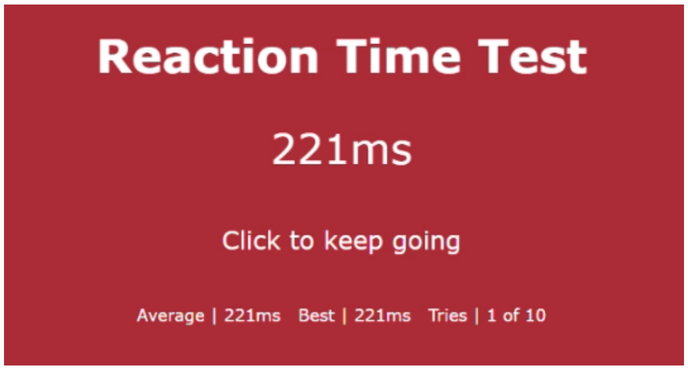

Reaction Time Test

...

Click!

Average | 0ms Best | 0ms Tries | 0 of 10

Figure 5.7: Reaction trial - step 3


Reaction Time Test

221ms

Click to keep going

Average | 221ms Best | 221ms Tries | 1 of 10

Figure 5.8: Reaction trial - step 4

both self-prediction and time warp), randomly shuffled. When network latency is 0, there is no latency compensation techniques. Each combination of network latency and latency compensation techniques was repeated 3 times, for a total 27 rounds (plus the practice round) for each user.

In both of the studies, after each round, users provided a subjective Mean Opinion Score (MOS) on a discrete 5-point Likert scale about the game experience in the preceding round. The question was "Rate the quality of the previous game round", and players chose an answer from 5 options: Excellent, Good, Fair, Poor or Bad. After completing the survey, the next round would commence when the user was ready, but users could take as much time as needed before starting the subsequent round,

It took each user about 30 minutes to complete all the tasks in each of the studies. A user study proctor was available for questions and trouble-shooting for the duration.

After completing all the game rounds, users were given a questionnaire with additional demographics questions about gamer experience – average time spent playing games and self-rated expertise with computer games.

In summary, the procedure each user followed was:

1. Fill out the screener questionnaire to ensure interest in participation and help understand player skill.

2. Come to the dedicated lab with pre-configured computers.

3. Adjust the computer chair height and monitor angle and height so as to be comfortably looking at the center of the screen.

4. Read the instructions regarding setup and game controls on the desktop.

Table 5.3: Study A demographic information

			Gaming per	Game		FPS	Reaction-
Study	Users	Age (yrs)	Gender	week (hours)	Self-rating	Self-rating	time (ms)
A	36	22.6 (3.4)	29 ♂7 ♀	11.2 (6.9)	3.5 (1.0)	3.3 (1.3)	206.9 (22.4)
B	30	23.1 (4.0)	26 ♂4 ♀	10.4 (8.3)	3.4 (1.1)	3.1 (1.0)	227.2 (40.0)

5. Complete the reaction-time test. (Takes about 30 seconds.)

6. Complete the hide and seek game rounds (1 practice round and 27 rounds with shuffled latencies), including the QoE surveys after each round. Take breaks between rounds if needed. (Takes a bit less than 30 minutes, total.)

7. Complete the final demographics questionnaire.

All users were eligible for a raffle to win a $25 USD Amazon gift card upon completion of the study, and many users received playtesting credit for relevant classes in which they were enrolled.

Study participants were solicited via university email lists. Thirty (30) users were recruited and participated in total for user study A and thirty-six (36) users were recruited and participated in total for user study B.

5.2 Navigation Results

This section first provides some summary demographics for the 30 participants from study A and 36 participants from study B (Section 5.2.1). Then, consistency of the human opponent is analyzed as a scrutiny of the methodology consistency (Section 5.2.2).

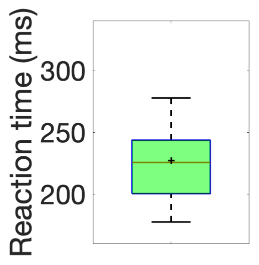


Figure 5.9: Reaction time (ms) - user study A

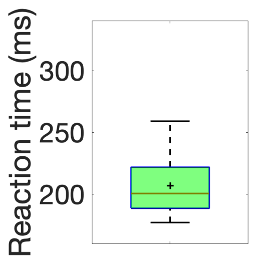


Figure 5.10: Reaction time (ms) - user study B

5.2.1 Navigation Demographics

Table 5.3 summarizes the demographic information for the user study participants. First-person shooter (FPS) self-rating is on a five-point scale, 1-low to 5-high. For age, FPS self-rating, and reaction times, the mean values are given with standard deviations in parentheses.

User study A had 30 participants, ranging from 18-31 years old but with the large majority of typical college age. Gender breakdown is predominantly male (26 males versus 4 female). We were slightly disappointed by the low number of female participants, but note that this reflects the gender breakdown of first-person shooter game players (about 7% of first-person shooter gamers are women [Lee17]) and our sample pool of university students skews male. Half of the participants played 10 or more hours of computer games per week. User self-ratings in general computer games slightly skews towards above the mid-point (mean 3.4), with self-rating in FPS games slightly lower (mean 3.1). Most participants majored in Robotics Engineering, Computer Science, or Game Development.

User study B had 36 participants, ranging from 18-28 years old but with the large majority of typical college age. Gender breakdown is predominantly male (29 males versus 7 female). Similar to study A, number of female participants is relatively low. Half of the participants played 10 or more hours of computer games per week. User self-ratings in general computer games slightly skews towards above the mid-point (mean 3.5), with self-rating in FPS games slightly lower (mean 3.3). Most participants majored in Robotics Engineering, Computer Science, or Game Development.

Figures 5.9 and 5.10 depicts the distribution of users' reaction times in user study A and user study B as a boxplot, respectively. The base local latency (22 ms) was subtracted from all reaction time trials and the resulting reaction times averaged for each user. The boxes depict quartiles and median for the distribution. The whiskers span from the minimum non-outlier to the maximum non-outlier. The black plus shows the mean value. From the graph, reaction times are mostly fast (about 230 ms for user study A and 200 ms for user study B), typical of computer game players [DGB09].

5.2.2 Opponent

The goal of our methodology is to have the same game conditions for each player, only varying the latency, hence our choice for the same opponent across all game rounds and all participants. However, there is a risk that the opponent either: a) gets better at the game over time, making the game more difficult for later participants, or b) gets fatigued and plays worse over time, making the game easier for later participants. Figure 5.11 depicts the performance of the opponent across the 30 users. The x-axis is the participant (player) number from one to thirty by participation order, and the y-axis is the score percent of the opponent – a score above 50 means the opponent got more than half the points and won, while a score below 50 means the participant got more than half the points and won. The circles are the mean scores of all game rounds against the opponent, and the dashed line is a linear regression through the mean values.

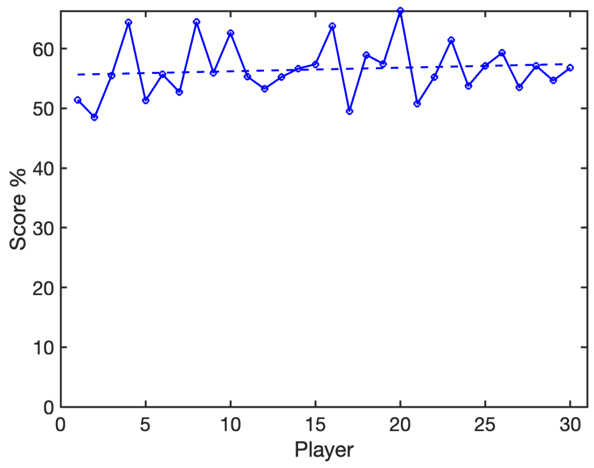


Figure 5.11: Opponent score versus player order.

From the graph, there is visible variation in performance across participants and while the opponent won more often than lost (the opponent never had added latency, only the participant), some participants beat the opponent. The p value of the linear regression is 0.54, indicating that there is no significant difference in the opponent's performance across the 30 players. Correspondingly, the regression line is visually flat, suggesting the opponent had consistent effort and skill over time. In other words, the participants likely faced a similar challenge regardless of their participation order.

Note, all analysis is done using the participant's data, not the opponent's data.

5.3 Navigation Analysis

This section first focuses on user study A. We compare the effects of local latency and network latency (Section 5.3.1), then present the core results – user performance (Section 5.3.2) and Quality of Experience (Section 5.3.3) in the presence of latency. Additional analysis examines the total time with the opponent in sight as a seeker and out of the opponent's sight as a hider (Section 5.3.4) and actions per minute

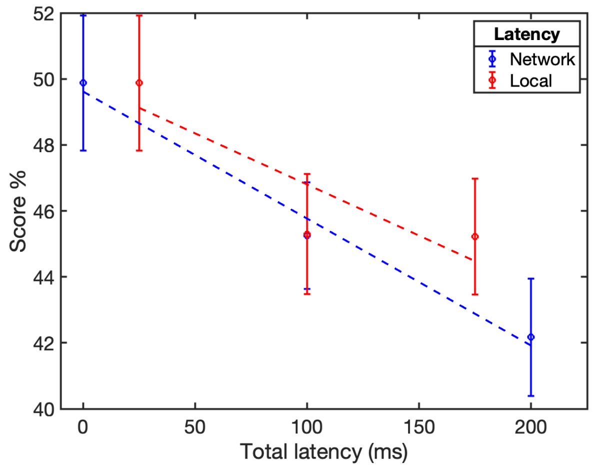


Figure 5.12: Score versus latency – comparing local latency and network latency.

(Section 5.3.5). Data from user study B are analyzed to compare player performance with different compensation techniques (Section 5.3.6).

5.3.1 Local Latency versus Network Latency

Previous work has shown that local latency has a higher impact on player performance than network latency [LKS+21a], but that was for a commercial game that has built-in latency compensation techniques. Our study allows comparison of player performance with local latency versus network latency in the absence of latency compensation.

We measure player performance based on the points earned. For each frame (50 f/s), if the player is a seeker and has the opponent in sight or if the player is hider and is not visible by the opponent, the player earns a point. For easier analysis, we convert the points to a score percentage which is the number of points earned divided by the maximum possible number of points (2000), multiplied by 100. A score percent above 50 means the participant won, while a score percent below 50 means the opponent won.

Figure 5.12 depicts score percent versus latency for network latency and local latency. The x-axis is the total latency (network plus local) in milliseconds. The y-axis is score percent for the player. The circles are the score percent means bounded by 95% confidence intervals and the dashed lines are linear regressions through the mean values. Blue is for rounds with network latency only, but without any added local latency and red is for rounds with local latency only without extra network latency. From the graph, player performance degrades with both types of network latency. While the slope for the network latency regression appears slightly steeper than the slope for the local latency regression, a regression for a unified model with latency (local/network) as a parameter shows the latency parameter is not statistically significant ($p = 0.36$). Given this, and since our expectation is that in the absence of latency compensation local latency and network latency impact navigation identically, for all subsequent analysis we do not differentiate the data by latency type but instead add local latency and network latency together. This provides a total of nine (9) different total latencies: 25, 100, 125, 175, 200, 225, 275, 300, and 375 ms.

5.3.2 Player Performance of Navigation

Figure 5.13 depicts player score percent versus total latency. The axes, data and trendlines are as for Figure 5.12 but the datasets are not differentiated based on latency type. From the graph, there is a generally linear downward trend in player performance with latency – i.e., players perform worse with higher latency. The linear regression fits the mean values well, with an R^2 of 0.93 and $p < .001$. As a take-away, an increase in total latency by 100 ms decreases score percent by 4.4 per minute of gameplay.

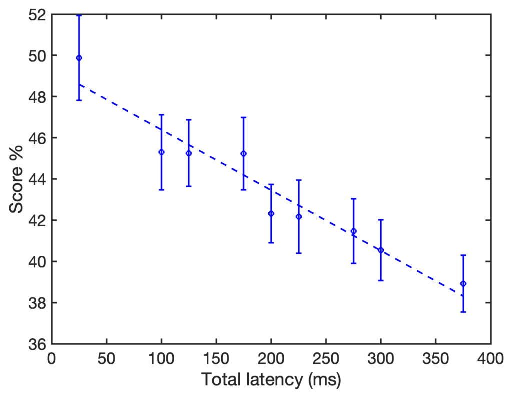


Figure 5.13: Score versus latency.

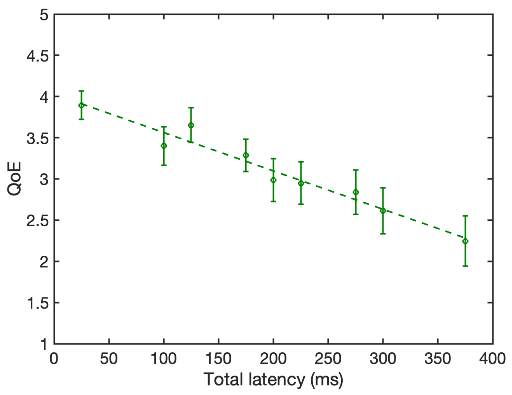


Figure 5.14: QoE versus latency.

5.3.3 QoE of Navigation

Quality of Experience (QoE) was assessed from the user responses to a Mean opinion Score (MOS) question filled out at the end of each round. Responses are converted to a 5 point scale, from 1-low to 5-high.

Figure 5.14 depicts QoE versus latency. The x-axes and trendline are as for Figure 5.13, but the data here are the QoE responses, shown on the y-axis, instead of the score. From the graph, mean user quality of experience degrades with latency. The linear regression fits the means well, with R^2 0.95 and $p < .001$. As a take-away, an increase in total latency by 100 ms decreases player QoE by 0.5 points on a 5-point scale.

While linear trends fit both Figure 5.13 and Figure 5.14 well, we note there are sub-regions where the linear trend does not clearly hold. For 100 - 175 ms and 200 - 275 ms of total latency, player performance does not vary much with latency, nor does QoE. Future work could analyze if and how sub-ranges of latency deviate from the overall linear trend.

5.3.4 Seeker Time and Hider Time

In first-person shooter games, what often matters is how long a player can see or be seen by an opponent. Longer time windows make it more likely to get a target in sight, aim, shoot and hit. For our game, seeker time is the total round time that a user has the opponent in sight while seeking, and hider time is the total round time that a user is out of the opponent's sight while hiding.

Figure 5.15 depicts seeker time versus latency and Figure 5.16 depicts hider time versus latency. The x-axes and trendlines are as for Figure 5.13, but the data and y-axes here are the seeker time in a round (Figure 5.15) and hider time in a round (Figure 5.16).

From Figure 5.15, the seeker time gets shorter with latency, meaning the player sees less of the opponent with an increase in latency. The linear regression fits the means well, with R^2 0.96 and $p < .001$. As a take-away, an increase in total latency by 100 ms degrades seeker time by 1.5 seconds per minute. Contrast this to the hider time in Figure 5.16. The hider time is relatively constant with latency, as indicated by the mostly flat regression line. The linear regression has an R^2 of only 0.34 and $p = 0.1$. This indicates that the ability of a player to hide from an opponent is not significantly impacted by latency.

Put together, latency would appear to have the strongest effect on navigation when a player is maneuvering to see an opponent but a much more limited effect on navigation when a player is avoiding being seen.

5.3.5 Actions per Minute

Actions per minute has been proposed as one metric to classify a game's sensitivity to latency [SSZ+20]. We analyze the converse – whether latency affects the player's

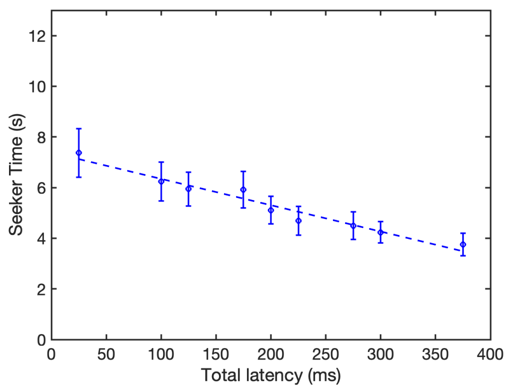


Figure 5.15: Seeker time versus latency.

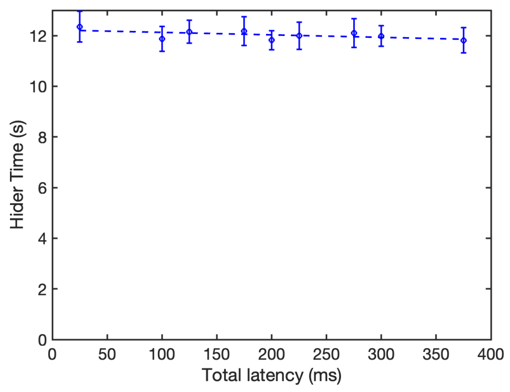


Figure 5.16: Hider time versus latency.

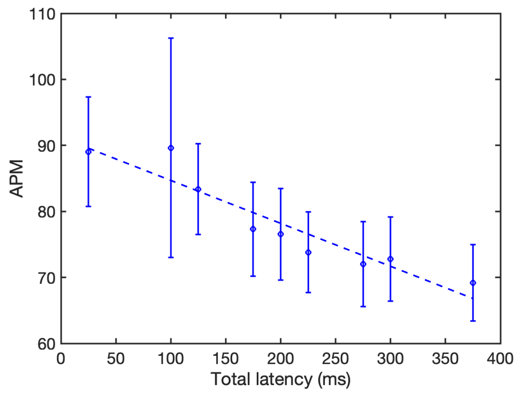


Figure 5.17: Actions per minute versus latency.

actions per minute. For navigation, the core parameter is how often the player intentionally moves in a particular direction, and for first-person navigation games on a PC this is via the WASD keys. For Hide and Seek, we compute actions per minute by the number of times a player presses a keyboard in a round divided by the round length (40 seconds).

Figure 5.17 depicts actions per minute versus latency. The x-axes and trendlines are as for Figure 5.13, but the data and y-axes here are the actions per minute. In general, there is considerable variation in actions per minute, more so than for score (Figure 5.13) and considerably more so than for QoE (Figure 5.14). There is also a noticeable downward trend as latency increases. The linear regression fits the means well with R^2 0.89 and $p < .001$. This indicates players move less often with an increase in latency, possibly because the lower responsiveness requires more deliberate movement actions by the players.

5.3.6 With Latency Compensation Techniques

Latency compensation techniques can mitigate the impact of network latency on players and are widely used in computer games, and self-prediction and time warp are commonly used latency compensation techniques in first-person shooter games [LXC22].

We compare four different latency compensation conditions: none, only self-prediction, only time warp, and both self-prediction and time warp. Local latency is kept at a minimum (25 milliseconds), and network latency varies: 0 ms, 175 ms and 350 ms.

Self-prediction [LSGH17] predicts game state based on player input, but before getting confirmation from the server. In first-person shooter games, self-prediction primarily helps a player's movement in the presence of network latency where an avatar will move and change direction as if there is no network latency. In our Hide

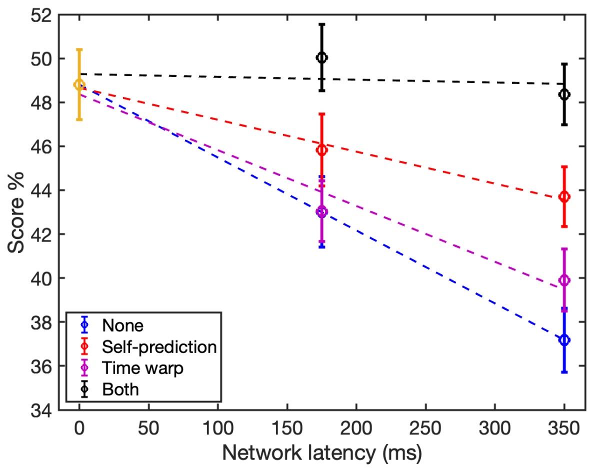


Figure 5.18: Score versus latency.

and Seek game, player prediction provides immediate feedback to the player for changing orientation (aiming) without having to get confirmation from the server.

Time warp [SG13] rolls back game state on the server to when the player action occurred on the client, applies the action, then rolls the game state forward to the current time. In our Hide and Seek game, the server decides whether the hider is in the seeker's sight based on the previous game state on the server when the game state sent by the client.

Figure 5.18 depicts score percent versus network latency where the axes and data are as for Figure 5.13. The data is separated out (means and trendlines) by the four latency compensation conditions: blue is without latency compensation, red is for self-prediction, purple is for time warp, and black is for both self-prediction and time warp. In the graph, the blue line has the steepest slope, showing that latency has the most impact on player performance when there is no latency compensation. The red line and the purple lines have shallower slopes with the green line decreasing faster than the red line, indicating that each technique individually can mitigate network latency and self-prediction helps players more. The black line is almost flat, indicating that both techniques together can nearly completely overcome the effects of network latency. The blue, red and purple lines fit their respective mean values well, with R^2 1, 0.99 and 0.97 and p = 0.001, 0.06, 0.11 respectively. The black line is almost flat with R^2 0.07 and p = 0.83, indicating that with both self-prediction and time warp, latency does not have statistically significant impact on player performance.

5.3.7 Navigation Summary

Table 5.4 summarizes the results from the study A in tabular form, providing the slope, y-intercept, adjusted coefficient of determination (R^2) and statistical significance (*p value*). Statistical significance is indicated in bold. Overall, navigation performance (score and seeker time) and perception (QoE) are impacted by total latency, both degrading with an increase in latency and accompanied by fewer actions per minute. Hider time, however, is not significantly affected by latency.

Table 5.5 summarizes the results from the study B in tabular form, providing the slope, y-intercept, adjusted coefficient of determination (R^2) and statistical significance (*p value*). Statistical significance is indicated in bold. Both self-prediction and time warp can mitigate the effects of latency on player performance.

Table 5.4: Results Summary - study A

metric	slope	y-intercept	R^2	p value
score	-0.029	49.3	0.93	<**.001**
QoE	-0.005	4.03	0.95	<**.001**
seeker time	-0.010	7.38	0.96	<**.001**
hider time	-0.001	12.23	0.34	0.10
APM	-0.065	91.20	0.89	<**.001**

Table 5.5: Results Summary - study B

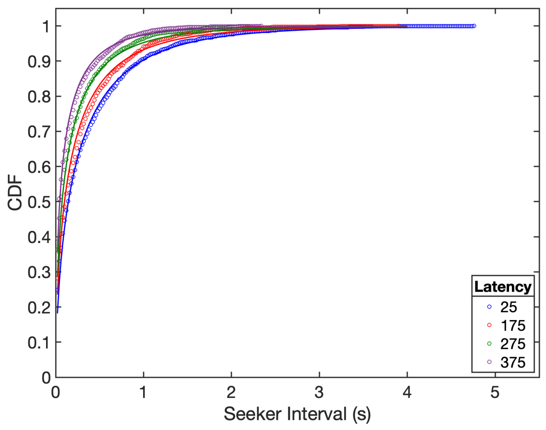
Compensation	slope	y-intercept	R^2	p value
none	-0.033	49.6	1	**<.001**
self-prediction	-0.015	49.0	0.99	0.06
time warp	-0.025	49.0	0.97	0.11
both	-0.001	49.3	0.07	0.83

5.4 Navigation Models

Analytic models can help generalize results beyond the necessarily narrow range of conditions tested in a user study. In our case, this means generalizing to latencies that are not one of the 9 discrete values used in our experiments. Analytic models can also be used to help with game design, where predicting player performance with latency can be used to adjust the game parameters [LG18] and change in-game attributes [SSSZ+18] in order to accommodate latency. Moreover, analytic models may be useful for discrete event simulations, where game performance can be selected using the model and applied to a simulated game outcome.

In our case, a model of navigation with latency is combined with models of target selection with latency [LC21b] in order to simulate moving and shooting in a first-person shooter game (Chapter 7). This enables predictions of player performance over a broad range of latency conditions, as well as other in-game conditions such as specific game parameters such as weapon attributes (Chapter 10).

Since the intent of navigation during first-person shooter combat is to position to shoot or not be shot, we analyze and then model the individual hider and seeker time intervals with latency. The former represents time windows when the player is hidden and cannot be shot, while the latter represents time windows when the player can see the opponent and potentially shoot them. In both cases, longer is better – more time being hidden or more time seeing a target.



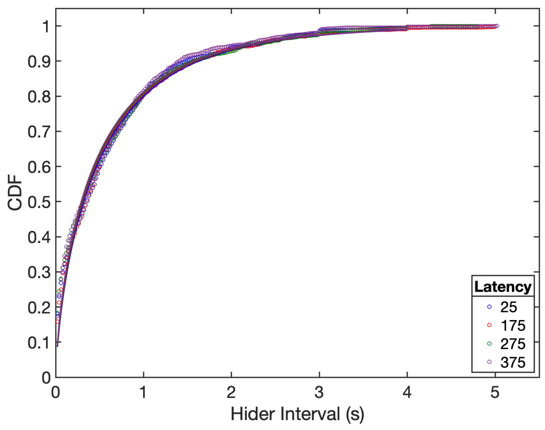


Figure 5.19: Seeker interval distribution. Figure 5.20: Hider interval distribution.

Figure 5.19 depicts the cumulative distribution function (CDF) of seeker intervals. The x-axis is length of the individual seeker intervals in seconds (i.e., a 2 second interval means the seeker had the hider in sight for 2 continuous seconds) and the y-axis is the cumulative distribution. The data is grouped for four[2] different total latency conditions. From the graph, the vast majority of the seeker intervals are below 2.5 seconds and the medians are less than 200 ms regardless of the latency. For reference, in first-person shooter games 200 ms is a relatively small time window to aim and shoot at an opponent. There is some visual separation of the lines based on latency, with lower latencies having slightly longer intervals (the lines are shifted down and to the right).

Figure 5.20 depicts the same data but for the hider intervals. From the graph, visually, the shape and distributions of the hider intervals looks similar to that of the seeker intervals (Figure 5.19). The hider interval distributions, however, do not separate based on latency. Put another way, the distribution of time durations where a hider cannot be seen is about the same regardless of the latency.

Since the CDF distributions appear to be an exponential fit but with a heavy

[2]The other latency conditions are not shown on the graph to make it readable.

Table 5.6: Models for seeker interval.

Name	Model	Adjusted R^2
Log gamma	$\frac{\gamma(\alpha,\beta\cdot T)}{\Gamma(\alpha)}$	0.53
Pareto	$1-\left(\frac{T_m}{T}\right)^{\alpha}$	0.80
Exponential	$1-e^{-\alpha*T}$	0.86
Stretched exponential	$1-e^{-\alpha*T^{\beta}}$	0.96
Weibull	$1-e^{-(x/\lambda)^k}$	**0.99**

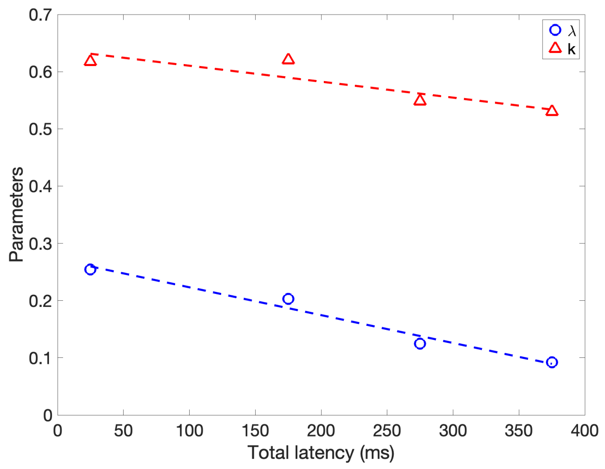


Figure 5.21: Weibull model parameters based on latency.

tail, we fit different heavy-tailed models to the data as indicated in Table 5.6 and find Weibull fits the data best. The CDF of a Weibull distribution is:

$$1-e^{-(x/\lambda)^k} \tag{5.1}$$

where k is the shape parameter and λ is the scale parameter.

Our goal is to have models representing performance with different latencies. Based on Figure 5.20, for the hider intervals, we do not parameterize the Weibull model based on latency. For the seeker intervals, rather than have a separate Weibull model for each latency value in our dataset, we model the Weibull parameters (k and λ) based on latency.

To integrate latency into the model, we use linear regression to fit the two parameters (λ and k) in Weibull distribution with latency and integrate linear regression to the Weibull distribution.

Figure 5.21 depicts a graph of the Weibull parameters fit to the seeker interval distributions for each latency. The dashed lines are a linear regression through these parameters, one line for each parameter. The red dashed line fits the λ values with R^2 0.97 and $p = 0.01$. The blue dashed line fits the k values with R^2 0.80 and $p = 0.11$. The slopes and intercepts are included in the second row in Table 6.8. These parameters provide a generalized Weibull model shown as lines for each latency group in Figure 5.19. Similarly, we overlay a Weibull model line for the hider interval distributions in Figure 5.20, albeit not parameterized based on latency. Details on these models is shown in Table 6.8. For both hider intervals and seeker intervals, the resulting model fits the data well with R^2 0.99 and low RMSE values.

Table 5.7: Modeling results for seeker and hider intervals with Equation 5.1.

interval distribution	λ	k	R^2	RMSE
All	0.29	0.61	0.99	0.01
Seeker	$-0.0005 \cdot l + 0.27$	$-0.0003 \cdot l + 0.63$	0.99	0.01
Hider	0.50	0.72	0.99	0.02

To assess whether there is time dependence in the intervals (e.g., does the length of one interval correlate to the length of the next), we computed an auto-correlation for all intervals for all rounds with τ from 1 to 20. Almost all the resulting auto-correlations were negative with an absolute value less than 0.05. These weak auto-correlations suggest that the interval lengths are independent. In other words, the length of an interval does not correspond to the length of intervals that follow.

Since seeker intervals are affected by latency, seeker intervals can be improved with latency compensation techniques. We build models of seeker intervals with different latency compensation techniques using data from user study B. To better fit the data with compensation techniques, we try different ways of improving the model. As a result, we modify equation in Section 5.4 with a parameter d:

$$1 - e^{-(x/\lambda)^k + d} \tag{5.2}$$

where d is a constant number. Details on these models with latency compensation techniques are shown in Table 5.8. For all the compensation techniques, the resulting model fits the data well with R^2 from 0.93 to 1 and RMSE values from 0.0001 to 0.0006.

Table 5.8: Seeker modeling results with latency compensation techniques.

Compensation technique	λ	k	d	R^2	RMSE
None	$-0.0004 \cdot l + 0.22$	$-0.0004 \cdot l + 0.69$	-0.27	0.99	0.0001
Self-prediction	$-0.0001 \cdot l + 0.22$	$-6.24e - 16 \cdot l + 0.69$	-0.23	1	0.0001
Time warp	$-0.0006 \cdot l + 0.25$	$-0.0008 \cdot l + 0.77$	-0.43	0.93	0.0005
Both	$-5.65e - 29 \cdot l + 0.34$	$-0.0001 \cdot l + 0.82$	-0.29	0.97	0.0006

The models in Table 5.7 and Table 5.8 can be used to reason about visibility intervals (hiding or seeking) in first-person shooter games (e.g., what is the likelihood of staying hidden for more than a second) and are incorporated in simulations (Chapter 7) of first-person shooter player performance.

5.5 Navigation Limitations

As noted in Section 5.2, both of the user studies had over 30 users (30 for user study A and 36 for user study B). While this sample size was large enough for statistically significant results for user performance and quality of experience with latency, more users would tighten the confidence bounds in Figure 5.13 and Figure 5.14. Similarly, potentially sampling more latencies, especially within the ranges we currently study, could help determine where linear trends do and do not hold.

Our sample is skewed towards males for both studies. While this may reflect the gender breakdown present in some first-person shooter games today, the results reported may not be representative of female performance. A follow-on study might also screen users for expertise in first-person games (e.g., using self-rated skill [LCD+20]) in order to provide for more focused analysis.

Our methodology intentionally had users compete against the same opponent in order to provide consistency across game conditions, save for the latency. This means, however, that our results are based on a specific opponent skill level – the impact of latency combined with different opponent skills was not assessed. Our use of the same human opponent for all users may limit the reproducibility of the study. While we would expect that trends would hold for other human opponents, the relative amounts may differ.

Serious game players often customize the software settings on their computers and games to suit their personal play preferences. For example, players may alter the mouse sensitivity or change the graphics resolution from the system defaults. These custom changes presumably improve that player's experience and/or performance. However, since customizations that deviated from our settings create a difference in test conditions between users, we did not allow any changes to the computer settings. This holds for other game configurations, too, such as using other mice, keyboards or monitors.

5.6 Navigation Summary

Analysis of results from our first-person navigation user studies shows that there is no significant difference in the impact of local latency versus network latency on player navigation performance in the absence of latency compensation techniques. Across the range of total latencies studied, player performance and quality of experience (QoE) both degrade linearly as latencies increase from 25 milliseconds to 375 milliseconds. Specifically, player scores at 25 milliseconds average over 25% better than player scores at 375 milliseconds. Over this same range, QoE decreases even more (nearly 60%), with the QoE at 25 milliseconds being about 4 (on a 5 point scale) and the QoE at 375 milliseconds falling to about 2.2. Player ability to move into position to see opponents decreases with latency; however player ability to hide to avoid being seen does not vary much with latency. The rate of player game actions decreases with latency by about 30% from 25 milliseconds to 375 milliseconds of latency. Time warp and self-prediction both mitigate the effects of latency, and, when applied together, can eliminate the effects of latency on player performance in navigation – i.e., players can perform as if there is no network latency when time warp and self-prediction are both used.

Chapter 6

Selection

The target selection action is a player pointing to and clicking on a moving or stationary target with an input device (e.g. a mouse). While target selection in a 2D space is fairly well-studied, target selection in a 3D space, such as shooting in first-person shooter (FPS) games, is not, nor are the benefits to players for many latency compensation techniques.

This chapter presents results from a user study that evaluates the impact of latency and latency compensation techniques on 3D target selection via a custom 3D selection game. User study participants play a game that isolates target selection. The target size, speed, latency values and latency compensation techniques vary between rounds. The combinations of testing conditions are shuffled and randomly applied to game rounds.

6.1 Selection Methodology

In order to assess the effects of latency on 3D target selection in a first-person game, we built a custom game that isolates the shooting action, implemented two commonly used latency compensation techniques, added controlled amounts of local and network latency, recruited participants for a user study, and measured player performance and quality of experience.

6.1.1 3D Target Selection Game

We designed and implemented a custom game in Unity that isolates the action of 3D target selection in a first-person shooter-type setting. In the game, the player stays at a fixed position at a corner of the map and can rotate to change orientation and aim, but cannot move or otherwise change positions. The opponent is a bot that spawns at a random location moving along one of three possible linear paths in the field of view and changes directions to avoid being hit. To mimic opponent movement in first-person shooter games, we extracted player movements patterns from data from a previous user study on Counter Strike: Global Offensive [LKS$^+$21b] (CS:GO, Valve, 2012). Specifically, we obtain the frequency of direction changes and jumps and the distribution of intervals between the same and use these values as the basis for our bot. From the data, the bot changes direction randomly every 3.02 - 8.70 seconds with a standard deviation of 1.43 seconds and jumps randomly every 1.41 - 9.95 seconds with a standard deviation of 2.43 seconds. The player tries to shoot the bot on the screen as fast as possible using a pistol with unlimited ammunition and a firing rate of 1 shot every 250 ms. It takes two hits to kill the bot. While the bot cannot shoot or otherwise damage the player, in order to provide some urgency for the player to shoot the bot quickly, the player's health is shown to decrease the longer the player takes to kill the bot.

The update rate for the game engine is fixed at 50 frames per second. Each frame, the game logs the running score for the player, the position of the enemy, the 2D distance of the reticle to the avatar, and 3D distance between the player's position and the bot's position. The game logs every shot as a hit or miss with

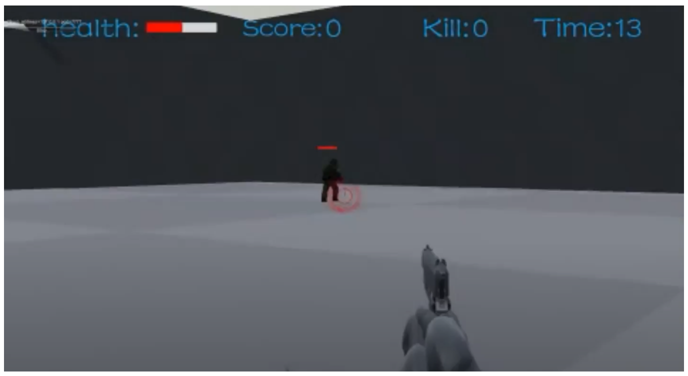


Figure 6.1: 3D target selection game screenshot.

corresponding timestamps.

A game round is over after the bot is killed (2 hits) or after 40 seconds, whichever comes first.

The game has one map – a single, square room, 36 meters in length and width, without any cover or obstacles. The player is always at a fixed location on the map in a corner. Before the round starts, a player is given a countdown timer whereupon a bot spawns at one of three different locations in the field of view. Figure 6.1 shows a screenshot of the game where the player is aiming at the target. The player health, score, number of kills and timer are shown in blue at the top of the screen.

The game has a client-server architecture typical of most network games where an authoritative server keeps the master world state and communicates state updates to the clients. In the default state, without latency compensation, all player input is sent to the server, the server applies the input to the game world and sends the new world state to the client which renders the state for the player.

6.1.2 Testbed Setup of Selection

We setup the game for our user study in a dedicated, on-campus computer lab. The testbed setup is depicted in Figure 6.2. The server hosts the game and is connected

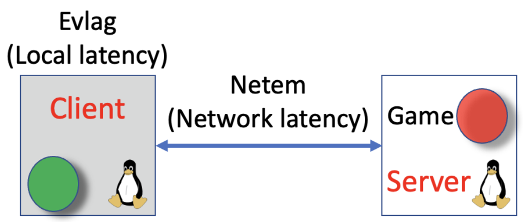


Figure 6.2: Client and server configuration of the selection study.

via high-speed LAN to the client. The client and server are Alienware PCs with Intel i7-4790K CPUs @4 GHz with 16 GB RAM and an Intel HD 4600 graphics card. The client is equipped with a gaming mouse and monitor so as to minimize local system latency and maintain consistency. The client has a 25" Lenovo Legion monitor running at 1920x1080 pixels displayed at 16:9 and 240 Hz, with AMD FreeSync and a 1 ms response time. The mouse is a Logitech G502 12k DPI with a 1000 Hz polling rate. The clients and the server run Ubuntu 20.04 LTS, with Linux kernel version 5.4.

The local latency was measured using the same technique as indicated in Section 5.1.2. This measurement method was done 10 times on our client, yielding an average base latency of 22 milliseconds, with a standard deviation of 5 milliseconds.

Local latency delays all input until resulting rendered output, whereas network latency delays receipt of the player's action at the server and subsequent server response to the client. Since the game server is authoritative, the client cannot update the position of an avatar until the server response has arrived. Thus, for the selection game without latency compensation techniques (as for all client-server games without latency compensation), local latency manifests similarly to network latency. Player orientation input until resulting avatar orientation change seen on the screen is delayed by at least the sum of the local latency and the network latency.

Our intent is to assess local latencies over ranges that might typically be found in personal computers, which range from about 25 milliseconds for a fast gaming system, are around 100 milliseconds for a typical computer system [ISGS15]. We added latency to all mouse and keyboard input using EvLag [LC21a] – an open-source tool for Linux that adds a constant amount of latency to any input device. Given our client has an average local latency of 22 milliseconds, EvLag adds either 3, 28, 53 or 78 milliseconds of latency for resulting total local latencies of 25, 50, 75, 100 milliseconds, respectively.

Similarly, our intent is to assess network latencies over ranges typically experienced by PC network game players, which can be near 0 milliseconds for a local area network (LAN) game, 100 milliseconds for a reasonable Internet connection, and 200 milliseconds for a slower Internet connection [opt20]. We added network latency to the server uplink and downlink equally using Linux tc with Netem[1] – a network control tool. The total network latency added to the client was either of 0, 50, 100 or 150 milliseconds. The total latency is the round trip time between the client and the server - it equals to the summation of the uplink latency and downlink latency. For example, if the total latency is 100 ms, the uplink latency and downlink latency are equal at 50 ms in our user studies.

Latency compensation techniques can mitigate the effects of network latency on game players. While there are many different types of latency compensation techniques, time warp and self-prediction are among the most commonly used in first-person shooter games [LXC22]. To better understand and quantify how much each helps users in 3D selection tasks with network latency, we investigated four different latency compensation conditions: none, time warp only, self-prediction only, and both time warp and self-prediction. We implemented the different latency compensation techniques in our custom selection game. Time warp is implemented

[1]`https://wiki.linuxfoundation.org/networking/netem`

the same way as in our user study in Section 5.1.

Target movement can change the game difficulty and affect player experience [CCG20]. To better understand the effects of target movement, we studied three motion modes of the enemy bot - normal that includes the movement described above (direction changes and jumping), normal but without jumping, and stationary without movement or jumping.

Table 6.1 summarizes the user study parameters.

Table 6.1: Parameters for the user study.

Parameters	Values
Local latency	25, 50, 75, 100 (ms)
Network latency	0, 50, 100, 150 (ms)
Latency compensation	none, time warp, self-prediction, both
Opponent motion	stationary, normal without jump, normal

Before the launch of the formal user study, a pilot study with 3 volunteers was conducted in order to test the viability of the procedure and tune the study settings. The pilot study results helped adjust round length, map size and layout, number of rounds, latency values and user instructions.

6.1.3 User Study Procedure of Selection

The study was approved by our Institute Review Board (IRB). Interested participants first filled out a screener questionnaire with questions on first-person shooter game-related experience to help select participants with some prior familiarity with FPS games. Selected users were invited to the lab at a pre-set time. Users then signed a consent form and positioned themselves at the test computer.

Users first did a custom reaction-time test written in Javascript and launched via a Chrome Web browser, the same reaction test as in our navigation study in

Section 5.1.3. In the test, users waited for a screen color change then clicked the mouse as quickly as possible, doing this 10 times. The average of the 10 values provides a measure of reaction time.

Users started by playing a practice round without any added latency to get familiar with the game. This data was not analyzed. Users next played additional rounds, each with options for local latency, network latency, latency compensation techniques, and target motion, randomly shuffled. The conditions tested include:

A *Local latency*: There are 3 conditions investigating local latency only (network latency of 0 ms, no latency compensation and normal bot motion): local latencies of 50, 75 or 100 ms.

B *Network latency and latency compensation:* There are 12 conditions investigating network latency with and without latency compensation (with local latency of 25 ms and normal bot motion): all combinations of network latencies of 50, 100 or 150 ms and four latency compensation conditions: none, time warp, self-prediction or both.

C *Local latency and network latency:* There are 3 conditions assessing latency compensation with local latency and network latency (network latency of 100 ms, local latency of 100 ms and normal bot motion): time warp, self-prediction or both.

D *Target motion:* There are 3 conditions investigating target motion (network latency and local latency of 100 ms, no latency compensation): normal bot motion, normal bot motion without jumping, or stationary bots.

Each condition above was repeated 3 times, for a total $(3 + 12 + 3 + 3) \times 3 = 63$ rounds plus the practice round. Other than the 64 rounds, there are 5 rounds with no latency, no latency compensation techniques and normal motion only. The 5 rounds are uniformly distributed between all rounds to ensure the consistency of player performance and test if there is fatigue giving the large number of game rounds. In total, each player played 64 + 5 = 69 rounds.

After each round, users provided a subjective Mean Opinion Score (MOS) rating on a discrete 5-point Likert scale about their experience: "Rate the quality of the previous game round". Players chose from 5 options: Excellent, Good, Fair, Poor or Bad. After completing the survey, the next round would commence when the user was ready, but users could take as much time as needed before starting the subsequent round.

It took each user about one hour to complete all the tasks in the study. A user study proctor was available for questions and trouble-shooting for the duration.

After completing all the game rounds, users were given a questionnaire with additional demographics questions about gamer experience – average time spent playing games and self-rated expertise with computer games.

In summary, the procedure each user followed was:

1. Fill out the screener questionnaire to ensure interest in participation and help understand player game familiarity.

2. Come to the dedicated lab at a pre-set time.

3. Adjust the computer chair height and monitor angle and height so as to be comfortably looking at the center of the screen.

4. Read the instructions regarding setup and game controls on the desktop.

5. Complete the reaction-time test. (Takes about 30 seconds.)

Table 6.2: Demographic information

Users	Age (yrs)	Gender	Gaming per week (hours)	Gamer Self-rating	FPS Self-rating	Reaction-time (ms)
39	20.0 (3.0)	31♂ 7♀ 1 Other	12.5	3.4 (1.3)	3.0 (1.2)	198.4 (16.9)

6. Complete the 3D target selection game rounds (1 practice round and 68 rounds with shuffled testing conditions), including the QoE surveys after each round. Take breaks between rounds if needed. (Takes a bit less than one hour, total.)

7. Complete the final demographics questionnaire.

Study participants were solicited via university email lists. All users were eligible for a raffle to win a $25 USD Amazon gift card upon completion of the study, and many users received playtesting credit for relevant classes in which they were enrolled.

6.2 Selection Analysis

This section first summarizes the demographics of our participants, then analyzes player performance and QoE for latency conditions without latency compensation, followed by analysis of latency compensation, and lastly, the effects of target motion on player performance.

6.2.1 Selection Results

Thirty-nine (39) users were recruited and participated in total. This section provides summary demographics for the participants.

Table 6.2 summarizes the demographic information for the user study participants. Gamer and first-person shooter (FPS) self-rating are on a five-point scale,

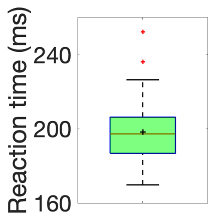


Figure 6.3: Reaction time (ms)

1-low to 5-high. For age, gamer self-rating, FPS self-rating, and reaction times, the mean values are given with standard deviations in parentheses. Ages ranged from 18-32 years old but with the large majority of typical college age. Gender breakdown is predominantly male (31 males), but does reflect the gender breakdown of first-person shooter game players (about 7% of first-person shooter gamers are women [Lee17]) and our sample pool of students at our university. Half of the participants played 10 or more hours of computer games per week. User self-ratings in general computer games slightly skews towards above the mid-point (mean 3.4), with self-rating in FPS games slightly lower (mean 3.0). Most participants majored in Robotics Engineering, Computer Science, or Game Development.

Figure 6.3 depicts the distribution of users' reaction times as a boxplot. The base local latency (22 ms) was subtracted from all reaction time trials and the resulting reaction times averaged for each user. The box depicts quartiles and median for the distribution. The whiskers span from the minimum non-outlier to the maximum non-outlier. The black plus shows the mean value. From the graph, reaction times are mostly fast (with an average about 200 ms), typical of computer game players [DGB09].

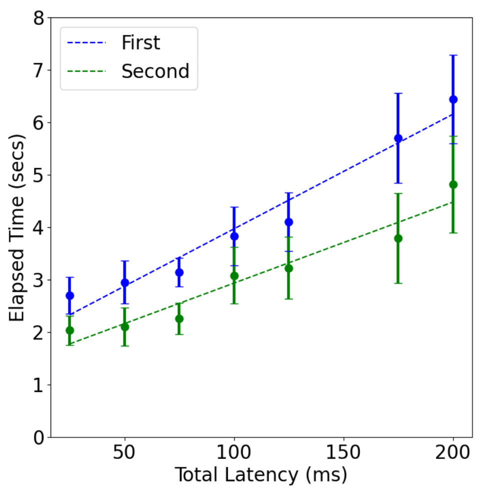


Figure 6.4: Elapsed time versus total latency

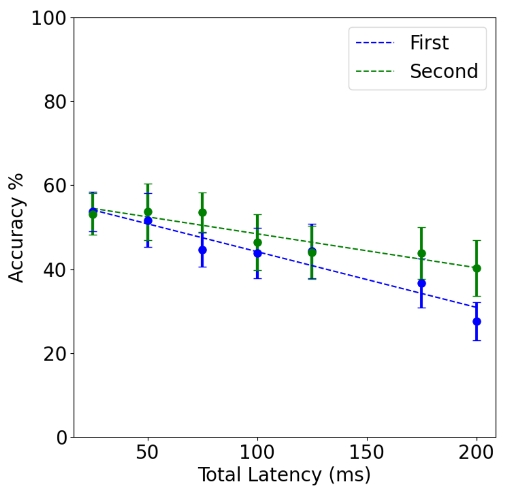


Figure 6.5: Accuracy versus total latency

6.2.2 Without Latency Compensation

We first analyze player performance without latency compensation. Analysis is for conditions where the opponent bot moves normally (test conditions A-C, but not D in Section 6.1). Since previous work [LKS$^+$21a] shows that local latency and network latency have an equivalent impact on players in the absence of latency compensation, we combine the conditions of local latency and network latency into seven different total latencies: 25, 50, 75, 100, 125, 175 and 200 ms. Thus, the analysis in this section pertains to selecting a moving, 3D target where all input – whether from the local computer system or from the network and remote servers – is delayed.

Figure 6.4 depicts the elapsed time required to select (hit) the target versus the total latency. The x-axis is the total latency (network plus local) in milliseconds and the y-axis is elapsed time in seconds. The circles are the mean elapsed times bounded by 95% confidence intervals and the dashed lines are linear regressions through the mean values. The blue points and line are for the first hit and the green points and line are for the second hit (the target takes 2 hits before the round ends). From the graph, the elapsed times increase with latency, indicating that latency makes it harder for players to shoot an opponent. The linear regressions fit the means well, with R^2 0.96 and $p < 0.001$ for the first hit (blue) and R^2 0.94 and $p < 0.001$ for the second hit (green). The blue line is above the green line meaning the first hit takes longer, on average, than does the first hit. The slopes differ slightly and suggest that the first hit is impacted slightly more by latency than the second hit. As a take-away, an increase in total latency by 100 ms increases elapsed time by about 2 seconds for the first hit, and 1.5 seconds for the second hit.

Figure 6.5 depicts accuracy versus total latency. The x-axis and trendlines are as for Figure 6.4, but here the y-axis is the accuracy (shots fired divided by shots taken) as a percent. From the graph, player accuracies for first and second hits degrade with latency, and the steeper slope of the blue line indicates that latency has higher impact for the first hit. The linear regressions fit the means well, with R^2 0.92 and $p = 0.001$ for the first hit (blue). R^2 0.87 and $p = 0.002$ for the second hit (green). As a take-away, an increase in total latency by 100 ms degrades accuracy percent by about 13% for the first hit, and about 8% for the second hit.

Note, the second hit times are affected by the weapon firing rate (i.e., the minimum time between successive shots – 250 ms in our study). Moreover, many first-person shooter games (although not ours) have weapon recoil that reduces accuracy and increases elapsed time for successive shots. While both factors – firing rate and recoil – are important for first-person shooter game performance, they are not the focus of our current study. Hence, for all subsequent analysis, we analyze the first hit only.

Figure 6.6 shows an analysis of quality of experience (QoE) versus total latency. The QoE is from the question "Rate the quality of the previous game round" from 1

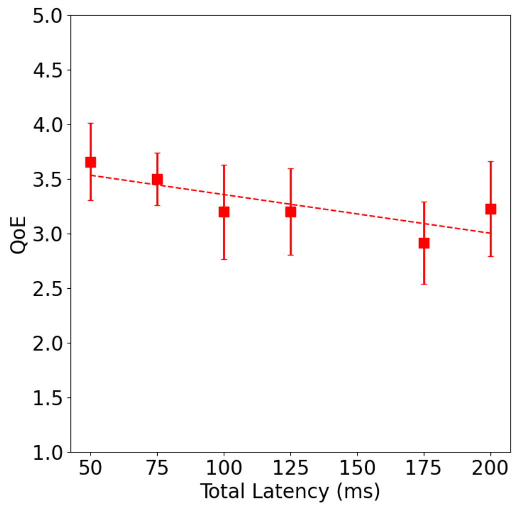


Figure 6.6: QoE versus total latency without latency compensation

(low) to 5 (high). The y-axis is the QoE and the x-axis is total latency. Each point is the QoE averaged over all users, bounded by 95% confidence intervals. The line is a regression through the mean values - with an R^2 of 0.62, $p = 0.064$. From the graph, latency degrades player experience, dropping about 0.4 points every 100 ms.

6.2.3 With Latency Compensation

Latency compensation techniques can mitigate the impact of network latency on players and are widely used in computer games, and self-prediction and time warp are commonly used latency compensation techniques in first-person shooter games [LXC22].

We compare four different latency compensation condition: none, only self-prediction, only time warp, and both self-prediction and time warp. Local latency is kept at a minimum (25 milliseconds), and network latency varies: 0 ms, 50 ms, 100 ms and 150 ms.

Self-prediction [LSGH17] predicts game state based on player input, but before getting confirmation from the server. In first-person shooter games, self-prediction

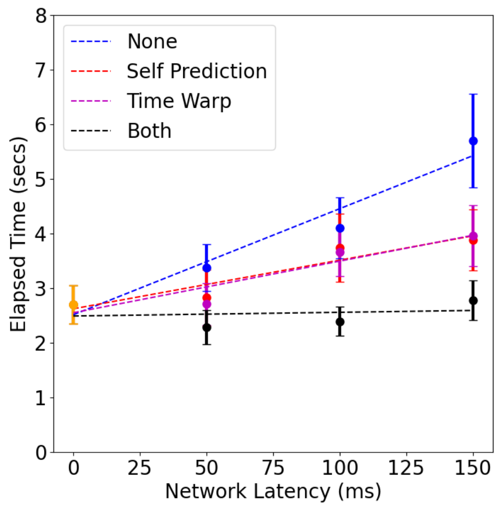


Figure 6.7: Elapsed time versus latency.

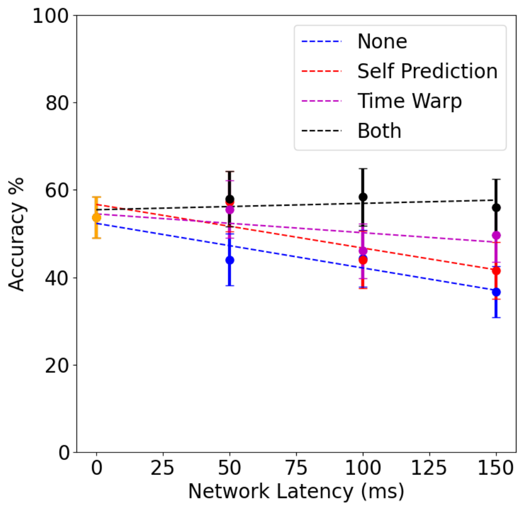


Figure 6.8: Accuracy versus latency.

primarily helps a player's movement in the presence of network latency where an avatar moves and change direction as if there is no network latency. In our first-person shooter, player prediction provides immediate feedback to the player for changing orientation (aiming) without having to get confirmation from the server.

Time warp [SG13] rolls back game state on the server to when the player action occurred on the client, applies the action, then rolls the game state forward to the current time. In first-person shooter games, the server decides whether a player hits a target based on the previous game state when the player fired the shot. With time warp, players can aim directly at the target.

Figure 6.7 depicts elapsed time versus network latency where the axes and data are as for Figure 6.4. The data is separated out (means and trendlines) by the four latency conditions: blue is without latency compensation, red is for self-prediction, purple is for time warp, and black is for both self-prediction and time warp. In the graph, the blue line has the steepest slope, showing that latency has the most impact on elapsed time when there is no latency compensation. The red line and the

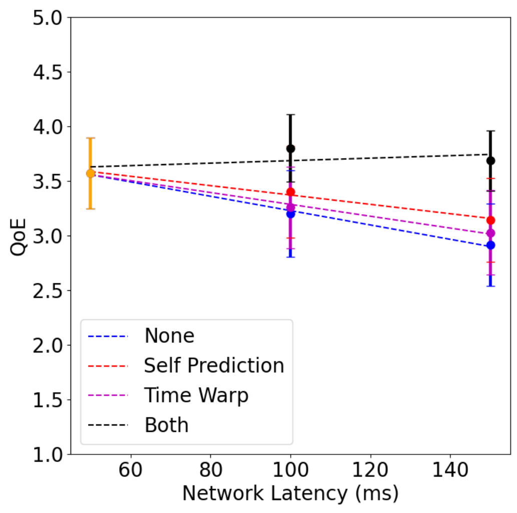


Figure 6.9: QoE versus network latency with latency compensation

purple line have shallower slopes and are comparable, indicating that each technique individually has about the same ability to mitigate network latency. The black line is almost flat, indicating that both techniques together can nearly completely overcome the effects of network latency. The blue, red and purple lines fit their respective mean values well, with R^2 0.95, 0.89 and 0.88 and $p = 0.025, 0.057, 0.061$ respectively. The black line is almost flat with R^2 0.03 and $p = 0.817$, indicating that with both self-prediction and time warp, latency does not have statistically significant impact on player elapsed time.

Figure 6.8 depicts the same data, but for accuracy. As for the elapsed time analysis, the uncompensated line (blue) is steeper than self-prediction (red) and time warp (purple), and with both self-prediction and time warp (black) the trend line is flat. The blue, red and purple lines fits have R^2 0.88, 0.32 and 0.49 and $p = 0.060, 0.433, 0.297$ respectively, while the black line has R^2 0.57 and $p = 0.244$ – with both self-prediction and time warp, network latency does not have significant impact on accuracy.

Figure 6.9 depicts QoE versus network latency with compensation techniques. The graph is the same as Figure 6.6 but the x-axis is only network latency and the data is separated by latency compensation condition. From the graph, as for player performance, QoE degrades the most with latency without compensation (blue), while self-prediction (red) and time warp (purple) both ameliorate the effects of network latency on QoE. The slightly steeper slope of the purple line compared to the red line indicates self-prediction helps QoE more than does time warp. The blue, red and purple lines have R^2 0.99, 0.99 and 0.99 and $p = 0.048, 0.073, 0.058$, respectively. The black line for QoE is almost flat with R^2 0.25 and $p = 0.667$ – with both time warp and self-prediction, latency has little impact on QoE.

As a take away, with self-prediction and time warp both on, latency does not appreciably affect player performance and QoE on 3D target selection tasks.

6.2.4 Local Latency and Network Latency

The latency compensation benefits from Section 6.2.3 can only mitigate the network latencies experienced in Section 6.2.2. We analyze how effective latency compensation is when there are also high local latencies (test condition C from our methodology).

So, for Figure 6.10, Figure 6.11 and Figure 6.12 the network latency is fixed at 100 ms, while the local latency – either 25 ms or 100 ms – is on the y-axis. The Cohen's d effect size quantifies the differences in means in relation to the standard deviation. Generally small effect sizes are anything under 0.2, medium is 0.2 to 0.5, large 0.5 to 0.8, and very large is above 0.8. The t test and effect size results are shown in Table 6.3, 6.4, 6.5 and 6.6. Statistically significance is highlighted in bold.

From these graphs, performance is worse without compensation and using both techniques helps more than each individually. However, even with both latency

Table 6.3: Pairwise T-test (p-value) for data at local latency 25 ms, in order of elapsed time, accuracy and QoE

	None			Self-prediction			Time warp			Both		
	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE
None	-	-	-	0.400	0.955	0.484	0.220	0.702	0.833	**<0.001**	**0.003**	**0.018**
Self-prediction	0.400	0.955	0.484	-	-	-	0.826	0.661	0.607	**<0.001**	**0.002**	0.123
Time warp	0.220	0.702	0.833	0.826	0.661	0.607	-	-	-	**<0.001**	**0.007**	**0.027**
Both	**<0.001**	**0.003**	**0.018**	**<0.001**	**0.002**	0.123	**<0.001**	**0.007**	**0.027**	-	-	-

Table 6.4: Pairwise T-test (p-value) for data at local latency 100 ms, in order of elapsed time, accuracy and QoE

	None			Self-prediction			Time warp			Both		
	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE
None	-	-	-	**<0.001**	**0.007**	0.758	**<0.001**	**<0.001**	0.917	**<0.001**	**<0.001**	0.758
Self-prediction	**<0.001**	**0.007**	0.758	-	-	-	0.778	0.254	0.817	**0.003**	**0.002**	1.0
Time warp	**<0.001**	**<0.001**	0.917	0.778	0.254	0.817	-	-	-	**<0.001**	0.060	0.817
Both	**<0.001**	**<0.001**	0.758	**0.003**	**0.002**	1.0	**<0.001**	0.060	0.817	-	-	-

Table 6.5: Effect size (Cohen's d) for data at local latency 25 ms, in order of elapsed time, accuracy and QoE. (We compute effect size only if T-test reports significance in Table 6.3)

	None			Self-prediction			Time warp			Both		
	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE
None	-	-	-	-	-	-	-	-	-	0.72	0.40	0.57
Self-prediction	-	-	-	-	-	-	-	-	-	0.53	0.41	-
Time warp	-	-	-	-	-	-	-	-	-	0.65	0.36	0.54
Both	0.72	0.40	0.57	0.53	0.41	-	0.65	0.36	0.54	-	-	-

Table 6.6: Effect size (Cohen's d) for data at local latency 100 ms, in order of elapsed time, accuracy and QoE. (We compute effect size only if T-test reports significance in Table 6.4)

	None			Self-prediction			Time warp			Both		
	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE	Elapsed time	Accuracy	QoE
None	-	-	-	0.56	0.36	-	0.64	0.50	-	0.95	0.76	-
Self-prediction	0.56	0.36	-	-	-	-	-	-	-	0.40	0.41	-
Time warp	0.64	0.50	-	-	-	-	-	-	-	0.45	-	-
Both	0.95	0.76	-	0.40	0.41 -	-	0.45	-	-	-	-	-

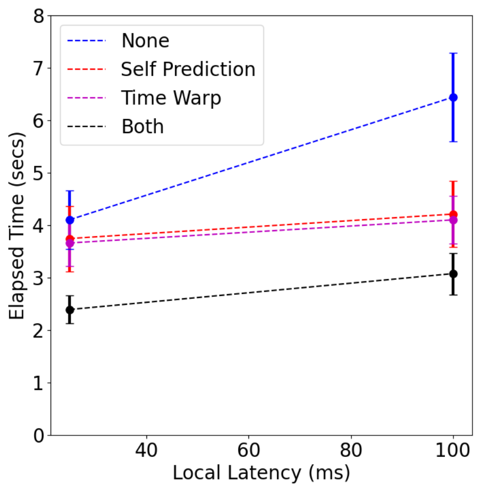


Figure 6.10: Elapsed time versus local latency with 100 ms network latency

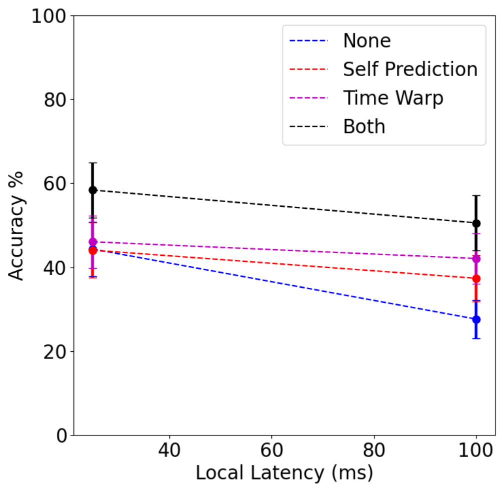


Figure 6.11: Accuracy versus local latency with 100 ms network latency

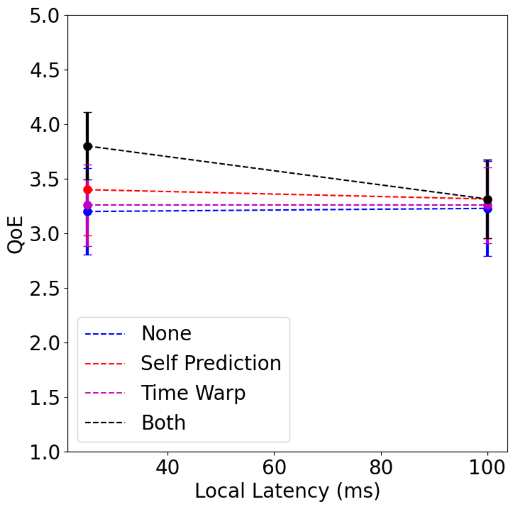


Figure 6.12: QoE versus local latency with 100 ms network latency

compensation techniques in effect, local latency of 100 ms still has worse performance than at 25 ms since the latency compensation techniques only mitigate the network latency and not the additional 75 ms of local latency.

In Figure 6.10, without latency compensation, there is significant difference between player elapsed time for 25 ms and 100 ms with $p < 0.001$. With only self-prediction or time warp, there is no significant different between player elapsed time at 25 ms and 100 ms with $p = 0.169$ and $p = 0.005$, respectively. With both compensation techniques on, there is significant difference between player elapsed time at 25 ms and 100 ms with $p < 0.001$. In Figure 6.11, without latency compensation, there is significant difference between 25 ms and 100 ms with $p < 0.001$. With only self-prediction or time warp, there is no significant different between 25 ms and 100 ms with $p = 0.121$ and $p = 0.364$, respectively. With both compensation techniques, there is no significant difference between 25 ms and 100 ms with $p = 0.094$.

For QoE, the individual techniques cannot fully overcome the 100 ms of network latency so player QoE is about the same, although both techniques together improve QoE from about 3.25 to 3.75 when there is only 25 ms of local latency.

In Figure 6.12, without latency compensation, there is no significant difference between 25 ms and 100 ms with $p = 0.922$. With only self-prediction or time warp, there is no significant different between 25 ms and 100 ms with $p = 0.753$ and $p = 1.0$, respectively. With both compensation techniques, there is significant difference between 25 ms and 100 ms with $p = 0.041$.

6.2.5 Target Motion

While players in first-person shooter games usually shoot at moving opponents, some opponents deliberately jump to avoid being shot while others deliberately stand still, either unaware they are being shot at or to better aim their own weapons.

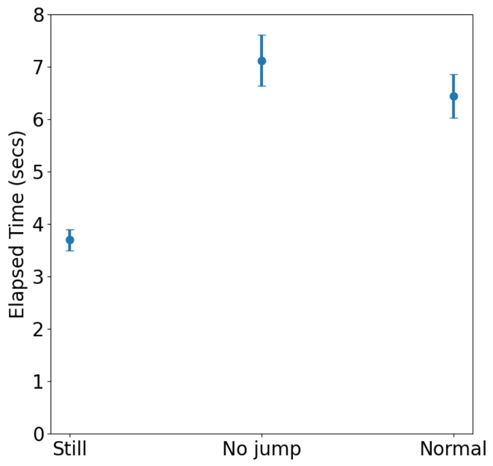


Figure 6.13: Elapsed time versus target motion

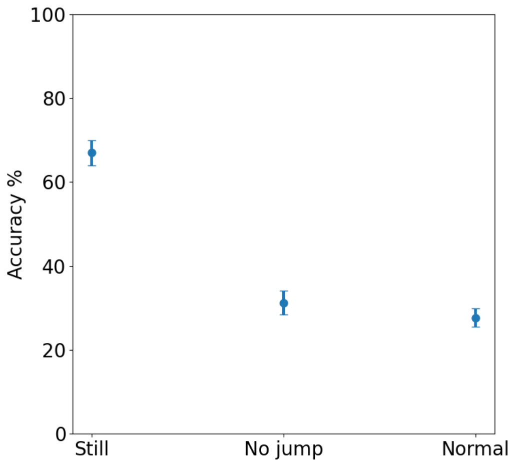


Figure 6.14: Accuracy versus target motion

The data from test condition D in our methodology lets us assess how much these motion variants impact target selection. For this condition, the motion varies – normal, normal without jumping, and stationary – with local latency and network latency are fixed at 100 ms, each.

Figure 6.13 depicts elapsed time versus the three motion conditions. The x-axis is the motion condition and the y-axis is the elapsed time. The circles are mean values and the bars are 95% confidence intervals. Figure 6.14 is as Figure 6.13 but the y-axis is accuracy instead of elapsed time. From the graphs, player performance is significantly better – about 1/2 the elapsed time and twice the accuracy – when the target is still. Moreover, jumping – a commonly used tactic by first-person shooter opponents – does not significantly degrade player performance. While we believe these results likely hold in other first-person shooter games, the degree to which they hold will depend upon the avatar speeds, and frequency of jumping and direction changes.

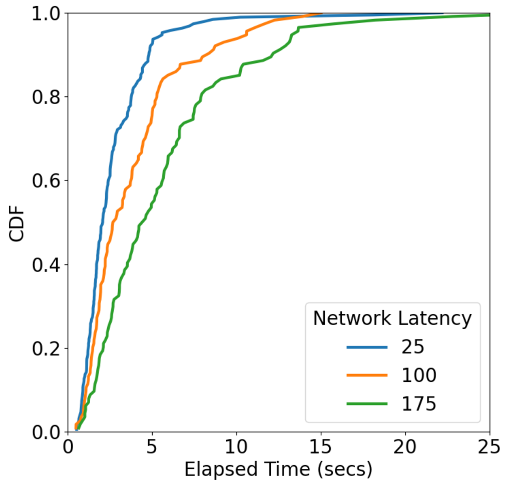


Figure 6.15: Distributions of first hits without latency compensation

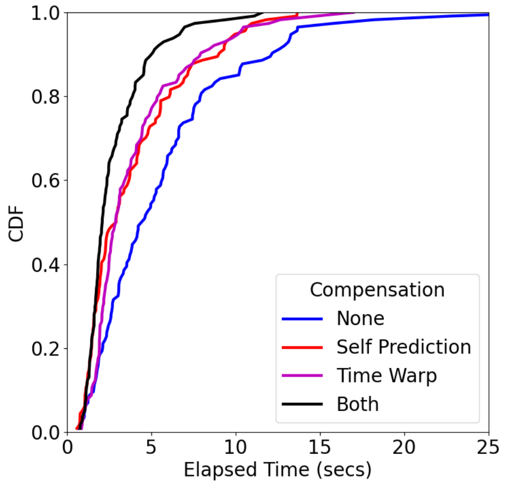


Figure 6.16: Distributions of first hits with latency compensation and 175 ms of latency

6.2.6 Selection Models

Analytic models can help generalize results beyond the necessarily narrow range of conditions tested in a user study. In our case, a model of 3D target selection with latency is be combined with models of navigation with latency from Chapter 5 in order to simulate moving and shooting in a first-person shooter game in Chapter 7. This enables predictions of player performance over a broad range of latency conditions, as well as other in-game conditions such as specific game parameters such as weapon attributes in Chapter 10.

Since the intent of 3D target selection is to click on the target as fast as possible, we analyze and then model the elapsed time with latency. For now, we only consider the rounds with normal target movement and without latency compensation.

Figure 6.15 depicts the cumulative distribution functions (CDF) of elapsed time. The x-axis is length of elapsed time in seconds and the y-axis is the cumulative distribution. The data is grouped for three[2] different total latency conditions. From the graph, there is some visual separation of the lines based on latency, with lower latencies having shorter elapsed times (the lines are shifted up and to the left).

Table 6.7: Models for selection.

Name	Model	Adjusted R^2
Log gamma	$\frac{\gamma((a \cdot L+b),(c \cdot L+d) \cdot T)}{\Gamma(a \cdot L+b)}$	0.62
Pareto	$1-\left(\frac{T_m}{T}\right)^{a \cdot L+b}$	0.92
Weibull	$1-e^{-(T/(a \cdot L+b))^{c \cdot L+d}}$	0.95
Stretched exponential	$1-a*e^{-(b \cdot L+c)*T^{f \cdot L+g}}$	0.97
Exponential	$1-a*e^{-(b \cdot L+c)*T}$	**0.98**

Since the CDF distributions appear have an exponential shape, we fit different exponential-like models to the data as indicated in Table 6.7 and find exponential fits the data best. The CDF (p) of the exponential distribution is described by:

$$p = 1 - a * e^{-(b \cdot L+c)*T} \tag{6.1}$$

where L is the total latency in milliseconds, T is the elapsed time in seconds and a, b and c are constants.

Figure 6.16 depicts the CDFs of elapsed time with latency compensation at 175 ms, as an example. In the graph, blue is for no compensation, red is for self-prediction, purple is for time warp, and black is for both techniques. Since there is no latency compensation in Figure 6.15, the blue line in Figure 6.16 is the same as the green line in Figure 6.15. From the graph, there is some visual separation of the lines based on latency compensation techniques, with the "both" condition having the shortest elapsed times, self-prediction only and time warp only having slightly longer elapsed times and "none" having the longest elapsed times. An exponential

[2]The other latency conditions are not shown on the graph to make it readable.

distribution fits the data of the four latency compensation conditions well, with an average R^2 0.98 and average RMSE of 0.041. For reference, the model parameters are included in Table 6.8.

Table 6.8: Parameters of models for elapsed time distributions (Equation 6.1)

Latency compensation	a	b	c	R^2	RMSE
None	1.34	-1.57	0.51	0.99	0.033
Self-prediction	1.33	-0.97	0.50	0.98	0.039
Time warp	1.39	-1.16	0.53	0.98	0.042
Both	1.54	-0.52	0.60	0.97	0.049

6.3 Selection Limitations

Our methodology intentionally had users play against a bot. The movement of the target can alter the difficulty of the game and hence affect player performance and experience. Although the movement of the bot is simulated from real player data from the first-person shooter game CS:GO, the results may not generalize to all FPS games which may differ in their target motion parameters.

In our custom game, the player only has a pistol as the weapon. However, the play style and strategy can vary with different types weapons, which, in turn, may result in different elapsed times and accuracy. Similarly, weapon accuracies and firing rates different than the ones in our study may have alternate performance data.

As noted in Section 6.2.1, our sample is skewed towards young males. While this may reflect the gender and age breakdown present in some first-person shooter games today, the results reported may not be indicative of players outside of this demographic.

There might be learning effects where players become more familiar with the game and perform better at later rounds, regardless of the latency. While we did not explicitly look for learning effects, our shuffled test conditions across all rounds should minimize any learning effects on specific conditions.

Serious game players often customize the software settings on their computers and games to suit their personal play preferences. For example, players may alter the mouse sensitivity or change the graphics resolution from the system defaults. These custom changes presumably improve the specific player's experience and may improve the player's performance. However, since customizations that deviated from our settings create a difference in test conditions between users, we did not allow any changes to the computer settings. This holds for other game configurations, too, such as other mice, keyboards or monitors.

6.4 Selection Summary

This chapter presents results from a user study on first-person selection under controlled latency conditions. We isolated selection in first-person games via a custom 3D target selection game where players took a fixed position, selected avatars in a matter akin to first-person shooter games – aiming and shooting the opponent. We setup our game in a private, local area network where we could control the local latency and network latency. We also investigated common latency compensation techniques applied in first-person shooter games and studied different types of target movement. Thirty-nine (39) participants played our custom game for 69 rounds across 22 different latency, latency compensation techniques and target motion conditions (a total of 60 minutes of gameplay), providing objective player performance data (elapsed time, hit/miss ratio) via log files and subjective opinion data (Quality of Experience) via surveys.

Analysis of the results shows that across the range of total latencies studied, player performance and quality of experience both degrade linearly as latencies increase from 25 ms to 200 ms. Specifically, elapsed time to select at 25 ms of latency average about 60% shorter than elapsed time at 200 ms. Over this same range, Quality of Experience (QoE) decreases about 0.4 with every 100 ms of latency. Time warp and self-prediction both mitigate the effects of latency, and, when applied together, can eliminate the effects of latency on both player performance and QoE – i.e., players can feel and perform as if there is no network latency when time warp and self-prediction are both used. Moreover, we derive models for elapsed time with different latency compensation techniques. The selection model is combined with the navigation model from Chapter 5 to simulate various first-person shooter scenarios in Chapter 7.

Chapter 7

Simulations

Analytic models and simulations have the potential to enable exploration of player performance with latency and game parameters as an alternative to time-intensive user studies. We develop simulations of first-person shooter scenarios using the navigation models from Chapter 5 and the selection models from Chapter 6. Then we validate the simulations with a study using a custom first-person shooter game in Chapter 9 and two studies using Counter Strike: Global Offensive in Chapter 8. Finally, we use our simulations to explore player performance with different game configurations and delays in Chapter 10. This section describes how we use our models of navigation and selection from Chapter 5 and Chapter 6 in our simulations of first-person shooter games.

7.1 Simulation Game

The default game we simulate is a two player first-person shooter. We simulate this game because killing the opponent is the core part of first-person shooter games. First-person shooter games may have different game mode and goals. However, players all need to shoot and kill the opponent. The goal for each player is to kill the opponent - the other player - as fast as possible. A game is over after one of the two players is killed or a 40 seconds time limit is reached, whichever comes first. The game map is a single, square room, default size of 36 meters in length and width, with multiple obstacles, the same as in our Hide and Seek game in Chapter 5. The player avatars spawn at at random location on the map but not currently in view of the opposing player. Each player spawns with a pistol-type weapon with unlimited ammo. The simulation also assumes a client-server architecture typical of most network games where an authoritative server keeps the master world state and communicates state updates to the clients. We simulate the win rates of players while varying local latency, network latency, latency compensation techniques, firing rate, number of hits required for a kill, map size, target size and player skill.

7.2 Models with Player Skill

To integrate player skill into our model, we first assess skill indicators, considering self-rated score, reaction time and in-game performance.

Our previous work [LCD+20] concluded that self-rated skill can be a good indicator of actual player skill on average. However, in our custom FPS game user study (Chapter 9), player performance also has a weak correlation with self-rated score in FPS games (R 0.21). Computer gamers tend to have faster reaction than average [Ric14]. We correlate reaction time from players in our user studies with their performance. However, in our custom FPS game user study (Chapter 9), player performance has a weak correlation with reaction time (R -0.13).

Instead, we use in-game performance as an indicator of player skill and separate player skill into two groups - players with top third performance are higher skill players and players with bottom third performance are lower skill players. We have

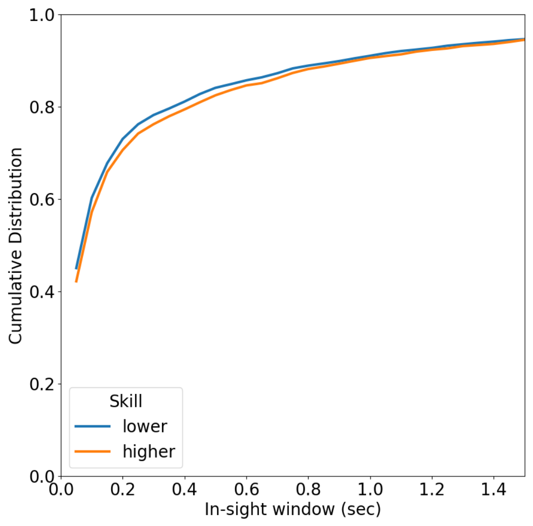


Figure 7.1: Distributions of in-sight windows

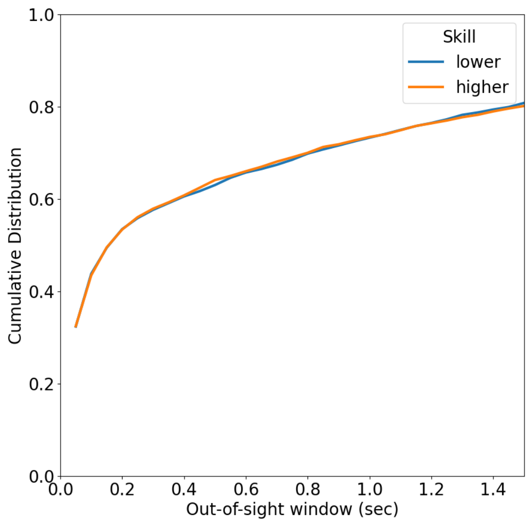


Figure 7.2: Distributions of out-of-sight windows

the separate models for 3 different skill groups of player - higher, using only top third players data; lower, using only bottom third players data; and all, using all players data.

Figure 7.1 depicts the cumulative distribution of in-sight window length from our navigation study in Chapter 5. The x axis is in-sight window length in seconds and the y axis is the cumulative distribution. The blue line is for lower skill players and the orange line is for higher skill players. From this graph, higher skill players have longer in-sight windows in general. Figure 7.2 depicts the cumulative distribution of out-of-sight window length. This graph is as Figure 7.1 but the data is for out-of-sight window length. From this graph, out-of-sight window length does not differ between higher skill players and lower skill players.

Figure 7.3 depicts the cumulative distribution of selection time. This graph is as Figure 7.1 but the data is for selection time. From this graph, higher skill players has shorter elapsed time.

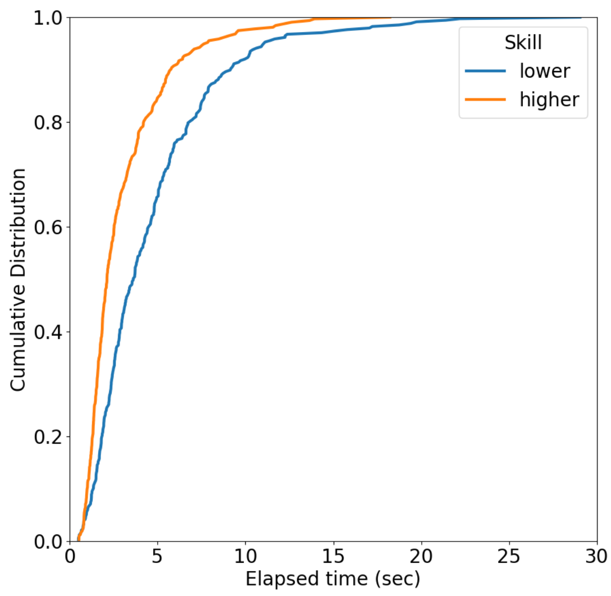


Figure 7.3: Distributions of elapsed time

The final models for in-sight windows CDF is:

$$p = 1 - e^{(-\frac{in\_sight\_window}{a\times(loc\_lat+net\_lat)+b})^{c\times(loc\_lat+net\_lat)+d}+e} \tag{7.1}$$

where p is probability, in_insight_window is the length of in_insight_window, loc_lat is local latency, net_lat is the network latency, and a,b,c,d and e are constants. We have separate constants for each latency compensation and player skill. The values of a, b, c, d and e of all the in-sight window models are included in Table 7.1.

Since player skill does not alter out-of-sight distribution as indicated in Figure 7.2, out-of sight window does not have skill as a parameter. The final models for out-of-sight windows are:

$$p = 1 - e^{-(\frac{out\_of\_sight\_window}{0.5})^{0.72}} \tag{7.2}$$

where p is probability, and out_of_sight_window is the length of out of sight windows.

The final models with player skill for selection times are:

$$p = 1 - a \times e^{(-(b\times(loc\_lat+net\_lat)+c\times firing\_rate+d\times 3D\_distance+e\times \frac{target\_size}{3D\_distance^2}+f)\times elapsed\_time)} \tag{7.3}$$

where p is probability, elapsed_time is the selection time, loc_lat is the local latency, net_lat is the network latency, firing_rate is the weapon firing rate, target_size is the height of the target, and 3D_distance is the 3D distance between two players, and a, b, c, d, e and f are constants. We have separate constants for each latency compensation and player skill. The values of a, b, c, d, e and f of all the selection models are included in Table 7.2.

Table 7.1: Simulation coefficient values - in sight window

Player skill	Latency compensation	a	b	c	d	e
All	none	- 3.64e-4	0.22	- 4.06e-4	0.69	- 0.27
All	self-prediction	- 9.44e-5	0.22	- 6.24e-16	0.69	- 0.23
All	time warp	- 5.60e-4	0.25	- 8.38e-4	0.77	- 0.43
All	self-prediction and time warp	- 5.65e-29	0.34	- 9.91e-5	0.82	- 0.29
Low	none	- 2.90e-4	0.19	- 3.05e-4	0.66	- 0.27
Low	self-prediction	- 6.53e-5	0.16	- 2.18e-23	0.67	- 0.21
Low	time warp	- 3.92e-4	0.19	- 4.66e-4	0.67	- 0.35
Low	self-prediction and time warp	- 6.28e-34	0.31	- 1.28e-4	0.81	- 0.30
High	none	- 4.05e-4	0.24	- 4.53e-4	0.72	- 0.24
High	self-prediction	- 1.23e-4	0.24	- 3.85e-5	0.71	- 0.21
High	time warp	- 6.37e-4	0.28	- 1.00e-3	0.83	- 0.43
High	self-prediction and time warp	- 1.21e-24	0.34	- 9.54e-5	0.82	- 0.24

Table 7.2: Simulation coefficient values - selection

Player skill	Latency compensation	a	b	c	d	e	f
All	none	1.18e-3	-1.43	-0.75	-3.03e-5	-0.53	0.82
All	self-prediction	1.17e-3	-0.93	-0.48	9.73e-4	16.07	0.64
All	time warp	1.22e-3	-0.97	-0.55	3.60e-4	12.89	0.75
All	self-prediction and time warp	1.27e-3	-0.02	-0.69	3.04e-3	57.63	0.69
Low	none	1.14e-3	-1.20	-0.50	-2.37e-5	2.54	0.61
Low	self-prediction	1.18e-3	-0.39	-0.63	9.92e-4	7.97	0.64
Low	time warp	1.20e-3	-0.40	-0.50	-8.29e-4	-4.05	0.67
Low	self-prediction and time warp	1.19e-3	0.17	-0.52	-2.14e-3	-18.40	0.76
High	none	1.23e-3	-1.72	-0.87	1.68e-3	21.13	0.96
High	self-prediction	1.17e-3	-1.21	-0.25	1.67e-3	29.38	0.60
High	time warp	1.22e-3	-1.56	-0.52	-1.40e-4	-0.61	0.89
High	self-prediction and time warp	1.32e-3	-0.09	-0.75	7.74e-3	118.67	0.66

7.3 Simulation Overview

For each game, we represent moving to shoot and avoid being shot with successive, alternative in-sight window and out-of-sight windows. Each in-sight window is followed by an out-of-sight window and each out-of-sight window is followed by an in-sight window. The game starts with an out-of-sight window since both players cannot see each other upon spawning. The time windows are generated until the time line reaches the game round time limit of 40 seconds , as depicted in Figure 7.4.

In each of the in-sight windows, we generate elapsed times from the selection model. If the elapsed time is within the in-sight window, the player hits the shot. Otherwise, if the elapsed time is longer than the end of the in-sight window, the shot is a miss as shown in Figure 7.5.

We keep generating hit times until the number of hits for a kill is required. Each player then has a series of hit timestamps indicating when they hit their shots in the game. Whoever gets the required number of hits soonest wins the game. Figure 7.6

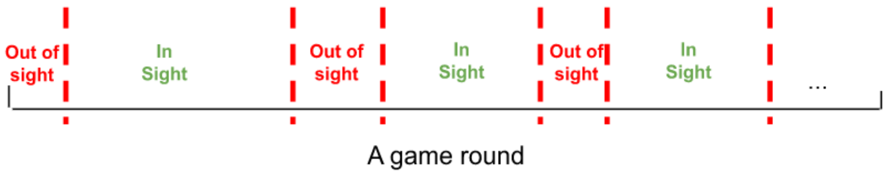


Figure 7.4: Opponent visibility intervals in simulation game

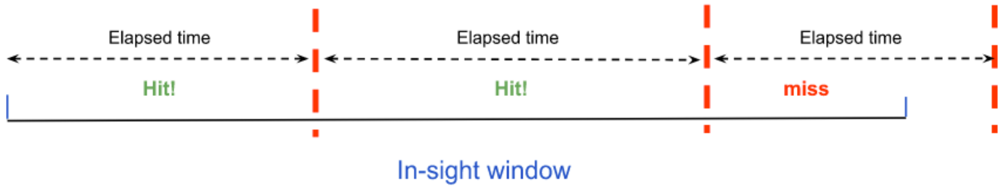


Figure 7.5: Opponent visibility intervals in simulation game

depicts the hit timestamps for each player. If we assume the number of hits required to kill is 5, in this example, player 2 gets 5 hits on player 1 before player 1 gets 5 hits on player 2, so player 2 wins the game.

7.4 Simulation Algorithm

This section shows the pseudocode for a first-person shooter game simulation. In the code blocks, variables have a style of "snake_case", functions and structures have a style of "camelCase", comments are in green and constants are CAPS.

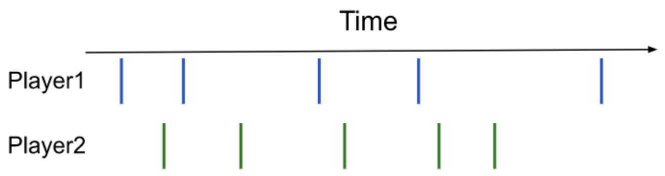


Figure 7.6: Hit timeline. The player gets the required number of hits first win the game.

7.4.1 Structures

The structure of gameInfo is included in Listing 7.1. The gameInfo includes game specific information. This structure is used in the simulation functions to pass game related information.

```
# Game parameters are the same for both players
struct gameInfo {
  latency_compensation = 0
  firing_rate = 250
  number_of_hits_required = 1
  map_size = 36
  target_size = 2
  time_limit = 40
}
```

Listing 7.1: The gameInfo structure with default values shown

The structure of playerInfo is included in Listing 7.2. The playerInfo structure holds the latency conditions, game parameters and the skill of the specific player. The skill values can be low, high and all. This structure is used in the simulation functions to pass game related information specific for a player.

```
# Parameters which can vary between players
struct playerInfo {
  local_latency = 0
  network_latency = 0
  skill = All
}
```

Listing 7.2: The playerInfo structure with default values shown

7.4.2 Functions

```
# Return a sample from model
function deriveSampleFromModel (gameInfo, playerInfo, model_type) {
  # Pick model based on model type, compensation and player skill
  compensation = gameInfo.latency_compensation
  skill = playerInfo.skill
  local_latency = playerInfo.local_latency
  network_latency = playerInfo.network_latency
  tot_lat = local_latency + network_latency
  # Sample a random  probability  from 0 to 1
  p = random (0, 1)
  if (model_type == in_sight_window)
  {
    # Find values of coefficient from lookup table
    # See Table 7.1
    a = COEFFICIENTS_IN_SIGHT[a][skill][compensation]
    b = COEFFICIENTS_IN_SIGHT[b][skill][compensation]
    c = COEFFICIENTS_IN_SIGHT[c][skill][compensation]
    d = COEFFICIENTS_IN_SIGHT[d][skill][compensation]
    e = COEFFICIENTS_IN_SIGHT[e][skill][compensation]
    in_sight_window_length =
      (e - ln(1 - p)) ^ (1 / (c * tot_lat + d)) * (a * tot_lat + b)
    return in_sight_window_length
  }
  elif (model_type == out_of_sight_window)
  {
    # Find values of coefficient from lookup table
    # See equation 7.2
    a = COEFFICIENTS_OUT_OF_SIGHT[a][skill][compensation]
    b = COEFFICIENTS_OUT_OF_SIGHT[b][skill][compensation]
    out_of_sight_window_length = a*(-ln(1-p))^(1/b)
    return out_of_sight_window_length
  }
  elif (model_type == selection)
  {
    # Find values of coefficient from lookup table
    # See Table 7.2
    a = COEFFICIENTS_SELECTION[a][skill][compensation]
    b = COEFFICIENTS_SELECTION[b][skill][compensation]
    c = COEFFICIENTS_SELECTION[c][skill][compensation]
    d = COEFFICIENTS_SELECTION[d][skill][compensation]
    e = COEFFICIENTS_SELECTION[e][skill][compensation]
    f = COEFFICIENTS_SELECTION[f][skill][compensation]
    distance = random (0, gameInfo.map_size)
    firing_rate = gameInfo.firing_rate
    target_size = gameInfo.target_size
    elapsed_time = - ln((1 - p) / a) / (b * tot_lat +
      c * firing_rate + d * distance +
      target_size * e / distance^2 + f)
    return elapsed_time
  }
  else
    return "Model type does not exist"
}
```

Listing 7.3: Pick model equation and return random sample

```
# Return in-sight sample window (in seconds)
function getInSightWindow (gameInfo, playerInfo) {
  model_type = in_sight_window
  # Get in-sight window with model
  window = deriveSampleFromModel(gameInfo, playerInfo, model_type)
  return window
}
```

Listing 7.4: Sample in-sight window from navigation model

```
# Return out-of-sight window (in seconds)
function getOutOfSightWindow (gameInfo, playerInfo) {
  model_type = out_of_sight_window
  # Get out-of-sight window with model
  window = deriveSampleFromModel(gameInfo, playerInfo, model_type)
  return window
}
```

Listing 7.5: Sample out-of-sight window from navigation model

In Chapter 5, we modeled the CDF of the seeker intervals - length of the time window with the opponent in sight; and the CDF of hider intervals - length of the time window that the player is out of the opponent's sight. An in-sight window is a time window when the player can see the opponent. We are able to simulate in-sight windows with the navigation seeker intervals model, using Equation 7.1. Function 7.4 depicts how an in-sight window is sampled. We first specify the model type and pass it in to Function 7.3 along with gameInfo and playerInfo. In Function 7.3, we first randomize a probability value between 0 and 1. Then we find model coefficients values corresponding to compensation technique and player skill. With the probability value and the model of the in-sight window CDF, we are able to derive a sample of the length of an in-sight window. An out-of-sight window is a time window when the player cannot see the opponent. We simulate an out-of-sight window the same as an in-sight window with the navigation hider intervals model, using Equation 7.2.

```
# Return elapsed time (in seconds)
function getSelectionTime (gameInfo, playerInfo) {
  model_type = selection
  # Get elapsed time with model
  time = deriveSampleFromModel(gameInfo, playerInfo, model_type)
  return time
}
```

Listing 7.6: Sample elapsed time from selection model

In Chapter 6, we modeled elapsed time for selection - time taken to hit the target. We are able to simulate elapsed time with the selection model, using Equation 6.1 in Section 6.2.6 on page 84. Function 7.6 depicts how an elapsed time is sampled. We first randomize a probability value between 0 and 1. With probability value and the elapsed time CDF model, we are able to derive an elapsed time.

```
# Return the length of a time window (in seconds)
function getWindow (is_hider, gameInfo, playerInfo) {
  current_window = 0
  if (is_hider)
    current_window = getOutOfSightWindow(gameInfo, playerInfo)
  else
    current_window = getInSightWindow(gameInfo, playerInfo)
  return current_window
}
```

Listing 7.7: Get a time window

Function 7.7 depicts how we simulate a time window. If the opponent is hiding, the current window is an out-of-sight window. Function 7.5 is called to sample an out-of-sight window. Otherwise, the opponent is in sight and Function 7.4 is called to sample an in-sight window.

```
# Return all hit timestamps in an in-sight window and total hits
function getHitTimestampsInAWindow (window_start_time,
   window_end_time, hit_count, gameInfo, playerInfo){
```

```
3    hit_timestamps_window = []
4    total_time = window_start_time #Timeline
5    while (total_time < window_end_time
6      and hit_count < gameInfo.number_of_hits_required)
7    {
8      elapsed_time = getSelectionTime(gameInfo, playerInfo)
9      hit_timestamp = total_time + elapsed_time
10     # Player missed shot, or already enough hits to kill
11     if (hit_timestamp > time_limit or
12       hit_count >= gameInfo.number_of_hits_required)
13       break
14     hit_timestamps.append(hit_timestamp)
15     hit_count += 1
16   }
17   return hit_timestamps_window, hit_count
18 }
```

Listing 7.8: Get hit timestamps in a window

Function 7.8 shows the pseudocode of getting hit timestamps for a player in an in-sight window. In each of the in-sight windows, if the hit count is less than the number of hits required, we keep generating elapsed times from the selection model, as indicated by line 8 in Function 7.8. If the elapsed time is within the in-sight window, the player hits the shot. Otherwise, if the elapsed time is longer than the end of the in-sight window, the shot is a miss as shown in Figure 7.5 (line 11 in Function 7.8). When complete, getHitTimestampsInaWindow() returns all timestamps where a player hits the target in an in-sight window.

```
# Return all hit timestamps for a player in a game
function getHitTimestamps (gameInfo, playerInfo) {
  hit_timestamps = []
  isHiderWindow = true
5   total_time = 0
6   hit_count = 0
7   while(total_time < gameInfo.time_limit)
8   {
9     window_length = getWindow(isHiderWindow, gameInfo)
10    window_start_time = total_time
11    window_end_time = min(window_start_time + window_length,
12      GAME_TIME_LIMIT)
13    # In-sight window
14    if (!isHiderWindow)
15    {
16      hit_timestamps_this_window, hit_count =
17        getHitTimestampsInAWindow(
18        window_start_time, window_end_time, hit_count, gameInfo,
19        playerInfo)
20      hit_timestamps.append(hit_timestamps_this_window)
21    }
22    isHiderWindow = !isHiderWindow
23    total_time += window_length
24  }
25  return hit_timestamps
26 }
```

Listing 7.9: Get hit timestamps in a game for a player

Function 7.9 shows the pseudocode for getting all the hit timestamps for a player in a game. Each while loop iteration (line 7), we first generate an either in-sight window or an out-of-sight window in line 9 with Function 7.7, depending on the state decided by the Boolean value isHiderWindow in line 4. For each in-sight window, the functions calls Function 7.8 to get all hit timestamps in the time window and add them to the results. After dealing with each window, the simulation accumulates time and resets current state of window, as indicated by lines 22 - 23. The function returns a series of all the hit timestamps in a game when a player hits an opponent.

Whichever player kills their opponent first wins the game. With the hits timestamps from both players, 7.10 detects the winner. Function 7.10 first calls Function 7.9 for both players passing in struct playerInfo which contains their gameInfo to get their hit timestamps, respectively (line 4 and line 5). The two players are simulated independently - their actions such as hiding and seeking and selection times are independent of what the other player is doing.

```
 1 # Return the winner of a game
 2 function findWinnerForaGame (gameInfo, player0Info, player1Info) {
 3     # Get player hit timestamps
 4     player_0_hits = getHitTimestamps(gameInfo, player0Info)
 5     player_1_hits = getHitTimestamps(gameInfo, player1Info)
 6 
 7     nhits = gameInfo.number_of_hits_required
 8     # Draw, did not kill each other in time limit
 9     if (len(player_0_hits) < nhits)
10       and (len(player_1_hits) < nhits)
11        return "Draw"
12     # Player 1 did not have enough hits
13     elif (len(player_1_hits) < nhits)
14        return player0
15     # Player 1 kills player 0
16     elif (len(player_0_hits) < nhits)
17        return player1
18     # Both have enough hits, so whoever gets enough hits first wins
19     elif (player_1_hits[nhits-1] < player_0_hits[nhits-1])
20         return player1
21     elif (player_1_hits[nhits-1] > player_0_hits[nhits-1])
22         return player0
23     else # They kill each other at the same time
24         return "Draw"
25 }
```

Listing 7.10: Find winner for a game

In Function 7.10, there are 4 outcomes for a game: both of players did not get enough hits and the game is a draw (line 9); only one player gets enough hits and the player with enough hits wins (line 13 and line 16); both players get enough hits, so the player with enough hits first wins (line 19); both players get enough hits and they get enough hits at the same time, so the game is a draw (line 23). findWinnerForaGame() returns the winner of a game.

```
# Return win rates of both player across a large number of rounds
function fpsSimulation (gameInfo, player0Info, player1Info) {
  win_count_player_0 = 0
  win_count_player_1 = 0
  for i in range(ITERATION)
  {
    winner = findWinnerForAGame(gameInfo, player0Info, player1Info)
    if (winner == player_0)
      win_count_player_0 += 1
    if (winner == player_1)
      win_count_player_1 += 1
  }
  player_0_win_rate = win_count_player_0/ITERATION
  player_1_win_rate = win_count_player_1/ITERATION
  return player_0_win_rate, player_1_win_rate
}
```

Listing 7.11: Entry function to the simulation

Function 7.10 simulates a single game and has one winner. Function 7.11 shows how we run a large number of games. Constant ITERATIONS denotes number of games to simulate (line 5). Each loop (line 5) is a game. For each game, we call Function 7.10 (line 7) with struct gameInfo to get the winner of the game. Function 7.10, we count how many times the players win across all the games (line 9). Finally, win count divided by number of iterations gets the win rate of both players (line 15).

```
# Print win rate of both players with 100,000 games.
function main(){
  # Set up gameInfo, player0Info and player1Info.
  gameInfo.latency_compensation = 0
  gameInfo.firing_rate = 0
  gameInfo.number_of_hits_required = 4
  gameInfo.map_size = 50
  gameInfo.target_size = 50
  gameInfo.time_limit = 60
  player0Info.local_latency = 25
  player0Info.network_latency = 100
  player0Info.skill = high
  player1Info.local_latency = 25
  player1Info.network_latency = 0
  player1Info.skill = high

  # Set number of games
  ITERATION = 100,000

  # Print both player win rate out of the games
  print(fpsSimulation(gameInfo, player0Info, player1Info))
}
```

Listing 7.12: How fpsSimulation() is called

Listing 7.12 shows how we setup and invoke function 7.11. As an example, assume we wanted to simulate a game no compensation, firing rate at 250 ms, number of hits required to kill at 4, map size at 50 x 50 m, target size at 2 m and time limit at 60 secs; player 0 has 100 ms of network latency, player 1 has no network latency and both players 25 ms of local latency; and both players are from high skill group. Listing 7.12 first apply values to gameInfo, playerInfo for both players and set number of games. Finally, it prints both player win rate out of the games.

7.5 Number of Iterations

In order to determine the number of iterations required for consistent results (constant ITERATION in Function 7.1), we run simulations with our base conditions - network latency at 150 ms and other parameters at the defaults. We increase number of iterations by 1000 each time (1000, 2000, 3000...) and observe the point where win rate mean and standard deviation are stable. Figure 7.7 depicts average win rate for the test player versus number of iterations. The x axis is the number of iterations and the y axis is average win rate. Figure 7.8 is as Figure 7.7 but the y axis is the standard deviation of the win rate. From the graphs, the mean and standard deviation stabilize after about 10,000 iterations. We use 100,000 iterations in all following simulations and explorations.

7.6 Simulation Parameters

The simulation allows specifications of the following game parameters: local latency and network latency, latency compensation techniques, weapon firing rate, number of hits, map size, target size and player skill. This section describes the bottom-up procedure (Section 7.6.1) and top-down procedure (Section 7.6.2) for how the

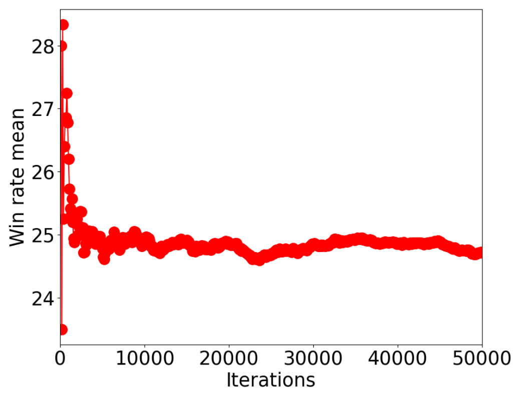


Figure 7.7: Average win rate versus iterations

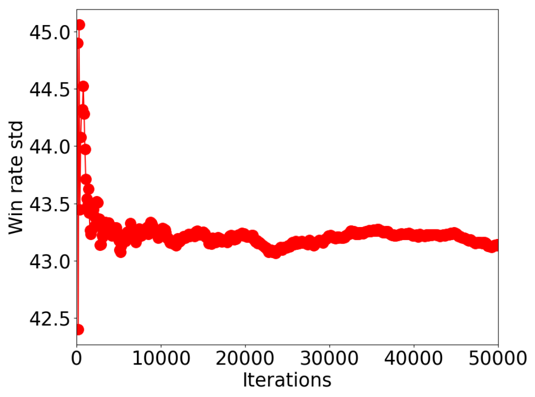


Figure 7.8: Average win rate standard deviation versus iterations

parameters are represented in the simulation.

7.6.1 Change Parameters - Bottom Up

The local latency and network latency impact the distribution of the in-sight window as indicated by line 7 in Function 7.4. The out-of-sight window is not impacted by latency. The distribution of selection (elapsed time) is also be changed with latency, so is the value generated from the selection model as indicated by line 8 in Function 7.6. The changes in distribution of selection and in-sight window can change elapsed time and window length generated from the models, followed by player's hit timestamp in a time window (Function 7.7) and in a game (Function 7.9). As a result, the winner from Function 7.10 can be different which can affect simulation results (win rates) from Function 7.11.

There are four conditions for latency compensation - no compensation, self-prediction only, time warp only and both. In our studies, player skills are decided by player performance. Players in the top third performance are higher skill players. Players in the bottom third performance are lower skill players. Our simulations consider two conditions for player skill - lower and higher. For each of the latency compensation and player skill conditions, we model selection and in-sight windows. There is a model for selection and a model for in-sight window under every latency compensation/player skill conditions. For each latency compensation / player skill conditions, Function 7.4 and 7.6 first pick the corresponding model at line 5, and then generate in-sight window lengths and selection time. The changes in distribution of selection and in-sight window can potentially change player's hit timestamp in an in-sight window from Function 7.8 and in a game from Function 7.9.

To change number of hits required to kill an opponent, we change number of total hits allowed in gameInfo as indicated by Listing 7.1. Once the number of hits reach preset value, the function returns hits timestamps. A change to number of hits required can potentially change players hit timestamps return from Function 7.9.

Firing rate is a parameter in selection model. In the 3D selection study (Chapter 6), it takes the player two shots to kill the enemy bot. The weapon firing rate is 4 rounds per second with unlimited ammo (no reload) as indicated in Chapter 6. We set 250 ms to be the firing rate of all shots after the first shot, and the minimum first hit elapsed time as the firing rate for the first shot. We then integrate the firing rate values into the selection model. By change weapon firing rate, we change the distribution of selection as indicated by line 5 in Function 7.6. Then the value generated from the selection model may be changed, changing the players hit timestamp in an in-sight window from Function 7.8 and in a game from Function 7.9.

Map size can be represented by the furthest 3D distance between the two players. 3D target size, 3D distance and 2D target size are mathematically related, any one of the three metrics can be derived from the other two. In our target selection game, we logged 2D target size and 3D distance between the two players every frame. To simulate map size, we integrated 3D distance into our selection model.

With simulation for each in-sight time window, we randomize a 3D distance value within the furthest 3D distance based on map size. The map size decides the limit of the randomized 3D distance. Then the value generated from the selection model as indicated by line 8 in Function 7.6 is changed. The changes in distribution of selection can potentially change players hit timestamp in an in-sight window from Function 7.8 and in a game from Function 7.9.

3D target size can be derived from 3D distance and 2D target size. If we change 3D target size, 2D target size changes proportionally. To simulate target size, we integrated 2D target size into our selection model. We use the 3D target size ratio (the ratio to the 3D target size used in selection study) as an input for the selection model. A change in the 3D target size changes the 3D target size ratio, as well as the distribution of the selection. Then the value generated from the selection model as indicated by 8 in Function 7.6 is changed. The changes in distribution of selection can potentially change players hit timestamp in an in-sight window from Function 7.8 and in a game from Function 7.9.

The bottom up procedure of how the parameters are presented are shown in Table 7.3.

7.6.2 Change Parameters - Top Down

To change local latency and network latency, we change latency values included in gameInfo as indicated in Listing 7.1. In Function 7.11, the gameInfo is passed into the function at line 7. Then Function 7.10 pass different latency values to Function 7.9 at line 4 and 5. Finally Function 7.9 pass different latency values to Function 7.4 and 7.6, changing the distribution of the in-sight window and selection which can potentially change simulation results.

To change latency compensation techniques or player skill, we change the pa-

Table 7.3: Change parameters - bottom up

Parameters	How represented
Local/network latency	In-sight window distribution, selection distribution → values of in-sight window length, selection time from Function 7.7 and 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Latency compensation	In-sight window model, selection model at line 5 in Function 7.7 and 7.6 → values of in-sight window length, selection time from Function 7.7 and 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Firing rate	Selection distribution → values of selection time from Function 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Number of hits	Variable nhits_required in Listing 7.1 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Map size	Selection distribution → values of selection time from Function 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Target size	Selection distribution → values of selection time from Function 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11
Player skill	In-sight window model, selection model at line 5 in Function 7.7 and 7.6 → values of in-sight window length, selection time from Function 7.7 and 7.6 → hit timestamps from Function 7.8 and 7.9 → winner from Function 7.10 simulation results from Function 7.11

rameter condition condition included in gameInfo as indicated in Listing 7.1. In Function 7.11, the gameInfo is passed into Function 7.10 at line 7. Function 7.10 pass different latency compensation techniques or player skill conditions to Function 7.9 at line 4 and 5. Then Function 7.9 asks Function 7.4 and Function 7.6 to pick the corresponding models at line 5. The difference in models can changes the distribution of selection and in-sight window which can potentially change simulation results.

To change weapon firing rate, map size, or 3D target size, we change the parameters included in gameInfo as indicated in Listing 7.1. In Function 7.11, the gameInfo is passed into the function at line 7. Then Function 7.10 pass playerInfo with different parameter values to Function 7.9 at lines 4 and 5. Finally Function 7.9 pass different parameter values to Function 7.6, changing the distribution of selection which can potentially change simulation results.

To change number of hits required, we will change number of hits required included in gameInfo as indicated in Listing 7.1. In Function 7.11, the gameInfo is passed into Function 7.10 at line 7. Then Function 7.10 pass gameInfo to Function 7.9 at line 4 and 5. Finally Function 7.9 changes the variable nhits_required in Function 7.8.

The top down procedure of how parameter are represented are shown in Table 7.4.

Table 7.4: Change parameters - top down

Parameters	How represented
Local/network latency	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 9, 16 in Function 7.9 →
	playerInfo at line 7 in Function 7.7 and line 8 in Function 7.8 →
	playerInfo at line 5 in Function 7.4 and line 5 in Function 7.6
Latency compensation	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 9, 16 in Function 7.9 →
	playerInfo at line 7 in Function 7.7 and line 8 in Function 7.8 →
	latency_compensation at line 5 in Function 7.4, line 5 in Function 7.6,
	playerInfo at line 5 in Function 7.4 and line 5 in Function 7.6
Firing rate	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 16 in Function 7.9 →
	playerInfo at line 8 in Function 7.8 →
	playerInfo at line 5 in Function 7.6
Number of hits	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	Variable nhits_required in Function 7.9
Map size	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 16 in Function 7.9 →
	playerInfo at line 8 in Function 7.8 →
	playerInfo at line 5 in Function 7.6
Target size	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 16 in Function 7.9 →
	playerInfo at line 8 in Function 7.8 →
	playerInfo at line 5 in Function 7.6
Player skill	gameInfo at line 7 in Function 7.11 →
	player0Info and/or player1Info at line 4, 5 in Function 7.10 →
	playerInfo at line 9, 16 in Function 7.9 →
	playerInfo at line 7 in Function 7.7 and line 8 in Function 7.8 →
	player_skill at line 5 in Function 7.4, line 5 in Function 7.6,
	playerInfo at line 5 in Function 7.4 and line 5 in Function 7.6

Chapter 8

Validation: Commercial First-person Shooter Game

We select *Counter-strike Global Offensive (CS:GO)* (Valve, 2012) for validation of the simulation in Chapter 7 as it is: a) used for esports, and b) has a large player base, but c) allows options for single-player play in stand-alone mode without an Internet connection so as to remove the confounding effect of uncontrollable network latencies from the study.

8.1 The Effects of Local Latency on CS:GO Game Players

This section presents results from a 43-person user study that evaluates the impact of local latency on Counter-strike: Global Offensive (CS:GO) players. Analysis of the results show pronounced benefits to CS:GO player performance (accuracy and score) for even small reductions in latency, with subjective opinions on Quality of Experience following suit. We compare the user study results to our simulation results and find that the trends of player performance versus latency hold.

8.1.1 CSGO (Local Latency) Study Methodology

To investigate how low amounts of system latency affect first-person shooter (FPS) gamers, we deploy the following methodology: 1) Find an FPS game that provides gameplay data in the form of log files and allows for customization of game settings; 2) Measure base system latency for our game system to get the minimum latency values; 3) Design and conduct a user study with the customized game and added delays to evaluate the impact of system latency on FPS gamers; and 4) Analyze the user study results in terms of player performance and quality of experience with latency.

A high-end gaming laptop and low latency gaming mouse were selected and benchmarked together in order to assess "best case" latency conditions while allowing room for emulating slightly higher latencies by adding delay to user input. The selected laptop was the Gigabyte Aero 15, accompanied by a Logitech G502 mouse. The Aero has an 8-core i7 9750H / 2.6 GHz processor, 16 GB RAM, an NVidia GF RTX 2070 graphics card, and a screen resolution of 1920x1080 pixels at 240 Hz. The G502 is a laser mouse with 12k DPI, 300 IPS, and a polling rate of 1 kHz. The laptop was configured with Ubuntu 20.04 LTS, with Linux kernel version 5.4. For this study, we assembled a set of 11 identical laptops and mice, all configured exactly the same way.

The local latency was measured with a 1000 frame/s camera (a Casio EX-ZR100) setup to capture the moment a user presses the mouse button and the resulting screen output, as indicated in Section 5.1.2. The average base latency is measured to be 25.2 milliseconds, with a standard deviation of 2.8 milliseconds.

In order to test the effects of latencies above the baseline 25 ms, additional

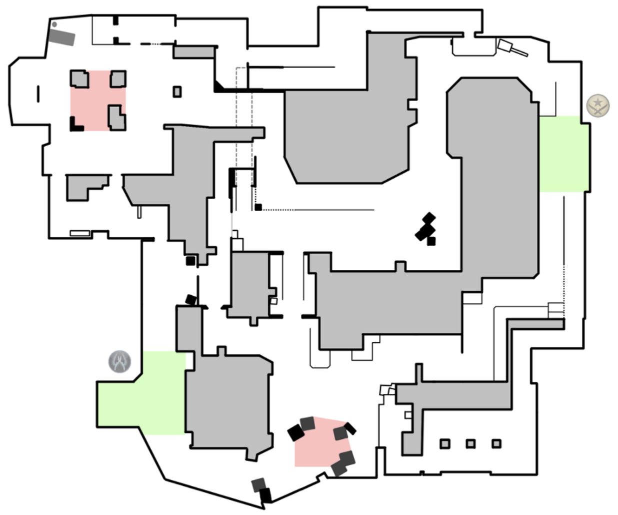


Figure 8.1: User study CS:GO map – Mirage

latency was added to all keyboard and mouse user input with *EvLag*, as described in Section 5.1.2. The added latencies for EvLag in our user study were 0, 25, 50, 75, 100 ms. So, with the 25 ms base latency, the resulting total system latencies experienced by our users are shown in Table 8.1.

Table 8.1: Total latencies for the user study.

Latency values (ms)				
25	50	75	100	125

Table 8.2: Weapon attributes

Weapon	Firing mode	Firing rate	Clip size	Reload time	Damage	Accuracy range
AK-47	Automatic	600 per min	30	2.43 s	36	21.74 m
AWP	Sniper	41 per min	1	3.7 s	115	69 m

While CS:GO matches often include team strategy, the focus of this study is on the effects of latency on individual player tactics. As such, a death match free-for-all game mode (no teams) was chosen. Thus, each round had open combat for the user

Table 8.3: Subjective questions per round

	Question	Source
Q1	I was frustrated by the round	GEQ [IdKP13]
Q2	The delayed reactions of the round annoyed me	TLX [GE88]
Q3	How well I did was completely due to me	Attribution [DM17]
Q4	Rate the responsiveness of the round	Survey [ERC18]

and 20 AI-controlled bots, where everyone fights everyone and the goal was to kill as many opponents as possible. CSGO has 4 bot options: 0 - easy, 1 - normal, 2 - hard, and 3 - expert. The bots in our study are at the third (of four) difficulty level, 2 - hard. While it is likely that the absolute scores would indeed differ for users pitted against human players, the relative effects should be similar since the latency affects the ability to aim and shoot (thus, score and accuracy). The AI-controlled avatars move with the same game physics as do human-controlled avatars, with the primary difference aiming accuracy and firing speed, impacting player deaths only, not player accuracy nor score (kills, assists).

There was no upper limit on player score – the game terminated after a fixed 4 minutes of time.

The size of CS:GO game maps range from the smallest map “Train” (5.4 kHu^2) to the largest map “Subzero” (9.7 kHu^2), where 1 kHu^2 equals about 0.02 km^2. A small map was desired to limit the need for the player to explore and wander and to maximize combat. The map chosen, “Mirage”, depicted in Figure 8.1, is the most popular [HLT20a] and third smallest map (5.9 kHu^2) [u/k19]. The user and the bots spawned at random locations on the map that were not currently in view of anyone else.

The selected CS:GO options were invoked via configuration files launched from the command line – this meant when starting CS:GO in the study, users immediately launched into the game, bypassing normal game lobbies and weapon selection phases.

The laptops were configured to start up and launch the user study sessions with minimal instructions so as to reduce user error. The laptops automatically booted to the test user account upon powering on and, effectively, double-clicking a custom test harness script labeled "Play" on the desktop launched each session. Once launched, the test harness: 1) started EvLag with the appropriate latency, 2) started CS:GO with the pre-determined map, weapon, bots and game mode, 3) allowed the game to run for 4 minutes, 4) stopped the game, 5) launched a quality of experience survey, 6) gathered and archived all game logs, input logs and survey results, and 7) repeated the process for each weapon and latency condition. The script provided for robust error checking in the event of user error (e.g., closing the game early) or system malfunction (e.g., software crashing).

The software configuration was replicated across 11 identical laptops. These laptops were hand-delivered to each user in a contact-less fashion, enabling users to participate in the study from their homes. A pickup time was arranged a few days after drop-off. Upon pickup, each laptop and mouse was carefully sanitized, the study data extracted, and the laptop reset for the next participant.

A user study proctor was available (by messaging and phone) for questions and trouble-shooting at all hours of the day during the study.

Before any sessions started, users first completed a reaction-time test as described in Section 5.1.3.

Users then played 5 sessions of CS:GO, each session being 5 rounds, each round having a different total latency selected from Table 8.1, randomly shuffled.

In order to assess the effects of latency for weapons with different skill requirements (e.g., high precision versus low precision), users were equipped with only one of two weapons at a time: A) the AK-47 (the most popular automatic rifle), or B) the AWP (the most popular sniper rifle) [HLT20b]. Detailed specifications on the two weapons are shown in Table 8.2. Both weapons had unlimited ammunition, but still needed to be reloaded when their magazine clips were empty (after 30 bullets for the automatic rifle and after each bullet for the sniper rifle). Since the firing rate for the sniper rifle is lower than that of the automatic rifle, we had 3 sniper sessions as opposed to the 2 automatic rifle sessions to get more action data (shots fired) for the sniper. The first 2 sessions were with the automatic rifle as the only weapon and the final 3 sessions were with the sniper rifle as the only weapon.

After each round, users were presented with a subjective survey consisting of four questions on a 5-point, discrete Likert scale about the game experience in the preceding round. The questions are shown in Table 8.3.

After completing the survey, the next round would commence. However, users could take as much time as they desired before starting the following round, and users were encouraged to take at least an hour break between sessions to avoid fatigue.

To allow users to get familiar with the map and the weapon, the first session with each weapon started with an additional practice round without any induced latency. Data from the practice round was discarded.

Before the launch of the formal user study, two pilot studies were conducted with volunteers (first 3 and then 4 friends/family, with 4 of them experienced CS:GO players) in order to test the viability of the procedure and tune the study settings. The pilot study results helped adjust weapon choices, latency values, number of rounds, number of sessions, system settings and user instructions.

Study participants were solicited via University email lists. Interested participants first filled out a screener questionnaire to provide basic demographic information (e.g., age and gender), but also for screening for appropriate CS:GO experience.

Qualification criteria included: 1) extensive experience playing games on a PC, 2) prior experience playing CS:GO, 3) at least 100 hours playing CS:GO and/or a high self-rating in FPS games, and 4) residence within 40 miles (65 km) of our university (to allow us to hand-deliver a laptop). Users were rewarded with a $75 USD Visa gift card upon completion of the study.

Forty-three (43) users were selected to participate in the user study out of 248 initial responses.

After returning the laptops, users were given a demographics questionnaire with additional questions about average time spent playing games and self-rated expertise with different CS:GO weapons.

In summary, the procedure each user followed was:

1. Fill out the screener questionnaire to ensure sufficient CS:GO experience.
2. Receive a pre-configured laptop and instructions regarding setup and game controls.
3. When ready to start, setup the laptop on a desk, plug in the power supply, connect the external mouse and place it on the included mouse pad for use by whichever hand is preferred. Plug in external headphones, if those are preferred over the laptop speakers.
4. Adjust the computer chair height and laptop angle/tilt so as to be comfortably looking at the center of the screen.
5. After powering on the computer, start the study by double-clicking on the "Play" icon on the desktop.
6. Complete the reaction-time test (takes about 30 seconds).

Table 8.4: Demographic information

Users	Age (yrs)	Gender	FPS Self-rating	CS:GO Self-rating	CS:GO Hours	Reaction-time (ms)
43	21.1 (5.0)	42 ♂1 ♀	4.5 (0.7)	4.2 (0.8)	664 (827)	217.2 (59.5)

7. Complete the first session (1 practice round and 5 rounds with shuffled latencies), including the QoE surveys after each round (each session takes about 25 minutes).

8. After a break of at least an hour, repeat the previous step for the remaining 4 sessions.

9. Return the laptop and receive remuneration.

10. Complete final demographics questionnaire online.

8.1.2 CSGO (Local Latency) Study Analysis

This section first summarizes participant demographics (Section 8.1.2) then presents the core results – user performance (Section 8.1.2) and Quality of Experience (Section 8.2.2) in the presence of local latency. Additional analysis examines user actions by weapon type in the presence of local latency (Section 8.1.2) and comparative performance by weapon type (Section 8.1.2).

Demographics

Table 8.4 summarizes the demographic information for the user study participants. FPS self-rating and CS:GO self-rating are on a five-point scale, 1-low to 5-high. For age, FPS self-rating, CS:GO self-rating, CS:GO hours played, and reaction times, the mean values are given with standard deviations in parentheses. Our user study

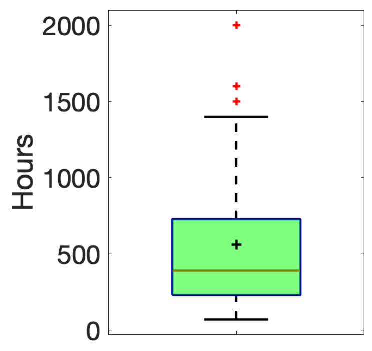


Figure 8.2: CS:GO hours played

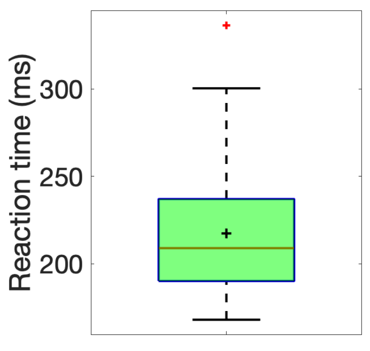


Figure 8.3: Reaction times

had 43 participants, ranging from 12-45 years old but with the large majority of typical college-age. Gender breakdown is predominantly male (42 males versus 1 female). We were slightly disappointed by the low number of female participants, but it should be noted that this is not atypical of esports players (about 5% are women) and reflects the gender breakdown of FPS game players, specifically [Sta20]. User self-rating as FPS and CS:GO gamers both skew towards "high" (mean 4.5 and 4.2 out of 5, respectively). Half of the users played 10 or more hours of computer games per week and most users majored in Robotics Engineering, Computer Science, or Game Development (not shown in the table).

Figure 8.2 depicts the distribution of users' hours playing CS:GO, and Figure 8.3 depicts the distribution of users' reaction times as boxplots. Each box depicts quartiles and median for the distribution. Points higher or lower than 1.4 $\times$ the interquartile range are outliers, shown by red pluses. The whiskers span from the minimum non-outlier to the maximum non-outlier. The black pluses shows the mean values. Users played from 70-5000 hours of CS:GO, with a mean of 664 hours and a large standard deviation of 827. Reaction times are mostly fast (just over 200 ms), typical of experienced computer game players [DGB09].

Data Cleaning

For the reaction times, out of the 430 reaction time trials, 3 were extremely long – over 700 milliseconds – perhaps because the user's attention wandered. These three trials are removed from the reaction time analysis (and from Figure 8.3).

For the game data, out of 1075 rounds, 17 had mouse or keyboard log files that were considerably shorter than the 4 minute round time, possibly because the user stopped playing or a program crashed. The game data and the QoE data from these 17 rounds are removed for analysis.

Player Performance

We measure user performance by effectiveness with each weapon: accuracy (shots hit divided by shots fired) and score (CS:GO computes score as $score = 2 \times kills + assists$). The CS:GO log files are mined to determine number of hits, kills and assists by each user for each round, and the EvLag log files are used to determine the number of shots fired based on the number of left mouse-button clicks.

Table 8.5 shows performance results averaged over all users and all game rounds, broken down by rifle type: automatic and sniper. The table has mean values, with standard deviations in parentheses.

Table 8.5: Performance summary

Weapon	Shots fired		Accuracy (%)		Score	
Automatic (AK-47)	385.0	(86.4)	17.8	(4.8)	45.8	(12.6)
Sniper (AWP)	50.8	(13.0)	56.9	(11.8)	55.9	(14.9)

Below we analyze weapon accuracy versus latency and player score versus latency.

Figure 8.4 depicts weapon accuracy versus latency for the automatic rifle (the

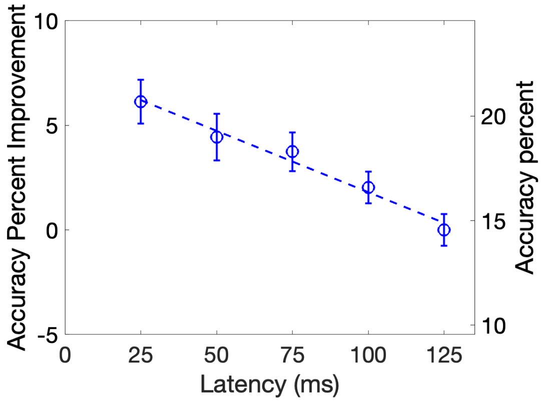


Figure 8.4: Accuracy – Automatic

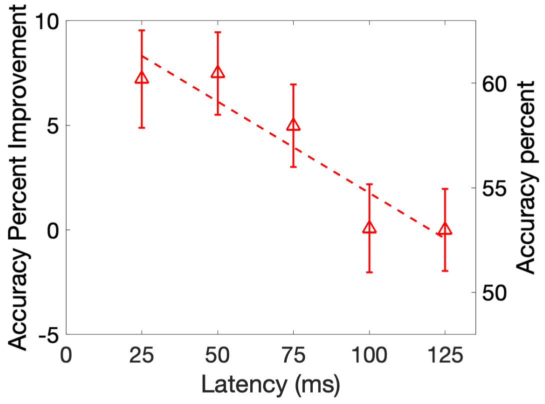


Figure 8.5: Accuracy – Sniper

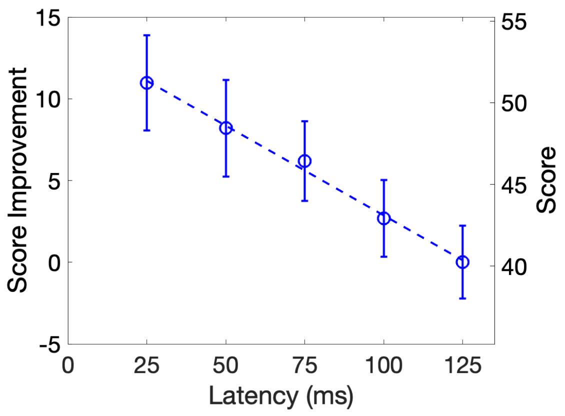


Figure 8.6: Score – Automatic

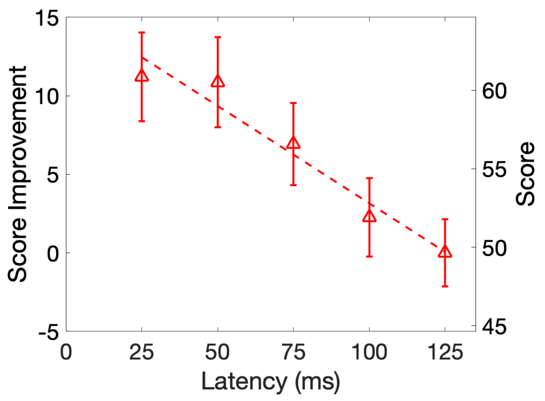


Figure 8.7: Score – Sniper

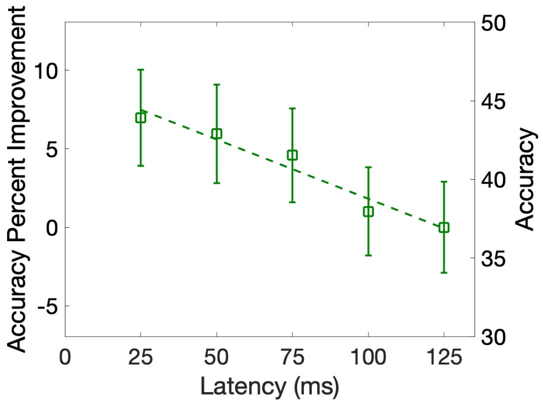


Figure 8.8: Accuracy – Combined

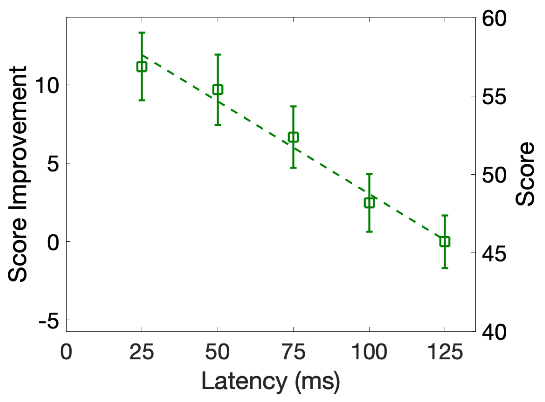


Figure 8.9: Score – Combined

AK-47). The x axis is total system latency in milliseconds. The right y axis is the weapon accuracy (percent) and the left y axis is the percent increase from the 125 ms latency condition. For example, an accuracy of 20 percent at 25 ms of latency compared to an accuracy of 15 percent at 125 ms of latency would be a 5 percent improvement on the left y axis. The circles are the means for all users for that weapon and latency condition, bounded by 95% confidence intervals. The dashed line shows a linear regression for the mean values. The regression fits the mean values well, with an R^2 of 0.98 and $p = 0.001$. As a take-away, for an automatic rifle, a decrease in latency by 10 ms improves accuracy by 0.6 percent on average.

Figure 8.5 depicts the same data as in Figure 8.4, but for the sniper rifle. The linear regression also fits the mean values well with an R^2 of 0.87 and $p = 0.02$. As a take-away, for a sniper rifle, a decrease in latency by 10 ms improves accuracy by 0.9 percent. However, from the figure, the sniper rifle accuracy trend may not follow this same linear trend from 50 ms to 25 ms latency and from 125 ms to 100 ms latency.

Considering the slopes of the regression lines in both Figure 8.4 and Figure 8.5, latency has a slightly larger effect on accuracy for sniper rifles than for automatic rifles.

Figure 8.6 depicts player score versus latency for the automatic rifle. The axes and points are as in Figure 8.4, but the data is the CS:GO score ($2 \times kills + assists$) instead of accuracy. The liner regression fits the mean values well, with an R^2 of 0.99 and $p < .001$. As a take-away, a decrease in latency by 10 ms improves player score by 1.1 points per 4 minutes of gameplay. For reference, often less than a single point separates the scores of top CS:GO players.

Figure 8.7 depicts the same data as Figure 8.6, but for the sniper rifle. The linear regression fits the mean values well, here, too, with an R^2 of 0.95 and $p = 0.005$. As

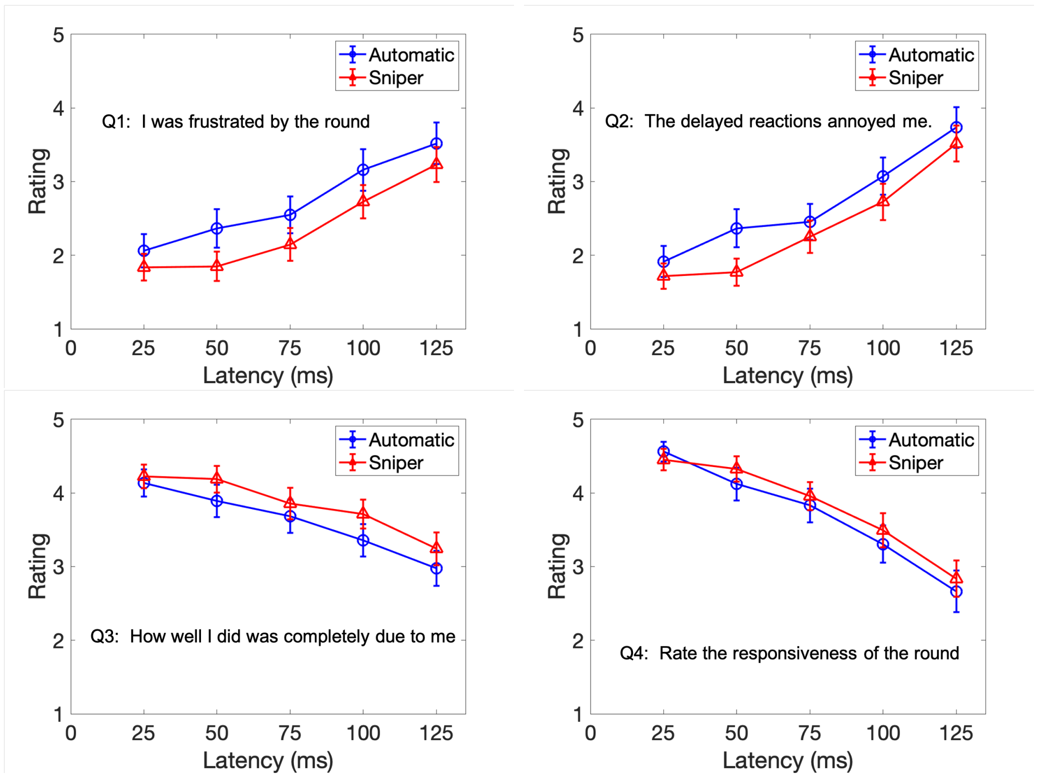


Figure 8.10: QoE – separate questions

a take-away, a decrease in latency by 10 ms improves player score by 1.2 points per 4 minutes of game play.

Considering the regression lines in both Figure 8.6 and Figure 8.7, latency has a similar impact on score for both sniper rifles and automatic rifles, with slightly more impact on the former.

To study how latency affects performance overall, the combined automatic rifle and sniper rifle data was analyzed. Figure 8.8 depicts the results for accuracy, with axes and data as for Figures 8.4 and 8.5. The linear regression fits the mean values for the combined data well with an R^2 of 0.95 and $p = 0.005$. As a take-away, a decrease in latency by 10 ms improves overall accuracy by 0.8 percent. Figure 8.9 depicts the results for score, with axes and data as for Figures 8.6 and 8.7. The linear regression fits the mean combined score values well with an R^2 of 0.98 and $p = 0.001$. As a take-away, a decrease in latency by 10 ms improves score by 1.2 points per 4 minutes of gameplay.

Tables 8.6 and 8.7 summarize the results in tabular form, providing the slope, y-intercept, adjusted coefficient of determination (R^2) and statistical significance ($pvalue$).

Table 8.6: Analysis Summary – Accuracy

Weapon	Slope	y-intercept	R^2	P value
Automatic	-0.06	22.23	0.98	0.001
Sniper	-0.09	63.49	0.87	0.02
Combined	-0.08	46.32	0.95	0.005

Quality of Experience

Quality of Experience (QoE) was assessed from user responses to 4 survey questions filled out at the end of each round. Responses are on a 5 point scale, from 1-low to

Table 8.7: Analysis Summary – Score

Weapon	Slope	y-intercept	R^2	P value
Automatic	-0.11	54.09	0.99	< 0.001
Sniper	-0.12	65.20	0.95	0.005
Combined	-0.12	60.57	0.98	0.001

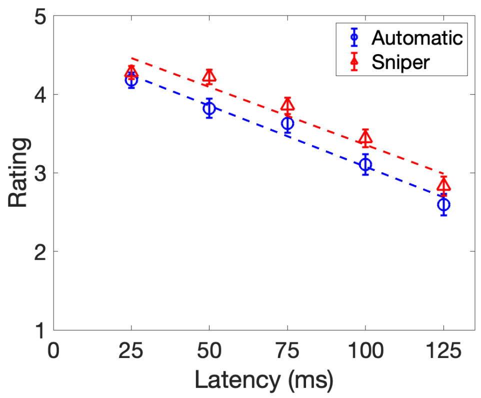


Figure 8.11: QoE – combined questions

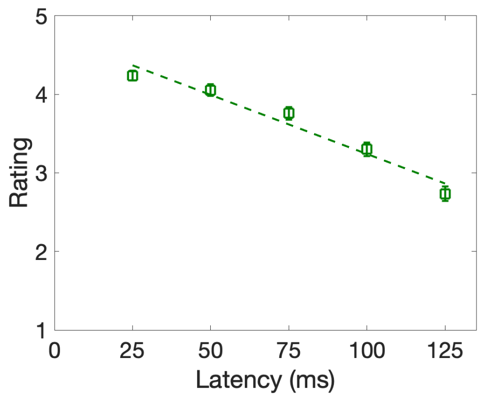


Figure 8.12: QoE – combined questions and weapons

5-high.

Figure 8.10 depicts ratings for each question versus latency. The x axis is total system latency in milliseconds and the y axis is the rating. The circles are the means for all users for that weapon and latency condition, bounded by 95% confidence intervals. The blue circles, bars and lines are for the automatic rifle and the red triangles, bars and lines are for the sniper rifle. The top left graph is for question 1: "I was frustrated by the round", the top right graph is for question 2: "The delayed reactions of the round annoyed me", the bottom left graph is for question 3: "How well I did was completely due to me", and the bottom right graph is for question 4: "Please rate the responsiveness of the round." For questions 1 and 2, lower is better and for questions 3 and 4, higher is better. In general, mean user perceptions get worse with latency, roughly the same for both weapons. Mean values for user perceptions when using the automatic rifle are slightly worse than those for the sniper rifle, but most confidence intervals overlap for the same latency values.

For an overall measure of QoE, we flip the ratings of question 1 and 2 (for example, a rating at 5 would be converted to 1, a 4 would be a 2, etc.), and compute an overall average (i.e., all questions are weighted equally) – here, higher is better.

Figure 8.11 depicts the results, with axes as for the graphs in Figure 8.9. The circles are mean values for all users across all latency conditions, shown with 95% confidence intervals. The dashed lines are linear regression fits through the automatic rifle (blue) and sniper rifle (red), separately. Figure 8.12 shows the same data, but combines the automatic and rifle data. The linear regressions fit the means well in all cases, with R^2 values of 0.97, 0.93 and 0.96 for automatic, sniper and combined, respectively. All values are statistically significant ($p = 0.002$, $p = 0.008$, and $p = 0.004$). However, the sniper rifle QoE values may not follow the same linear trend from 25 to 50 ms latency, similar to sniper accuracy values (Figure 8.5). As a take-away, a decrease in latency by 10 ms improves QoE by 0.15 points out of 5.

Play Style

In addition to performance and perception, latency may impact how a player interacts with the game. For CS:GO, this may manifest in a different firing rate or different ratio of avatar movement to shooting.

Figure 8.13 and Figure 8.14 depict shots fired per minute versus latency for the automatic rifle and the sniper rifle, respectively. The x axes are the total latency and the y axes are shots per minute – the right y axis is the number of shots per minute and the left y axis is the shot per minute increase over the 125 ms condition. Points are mean values across all users for that latency condition and weapon, shown with 95% confidence intervals. From Figure 8.13, users generally fire automatic rifles

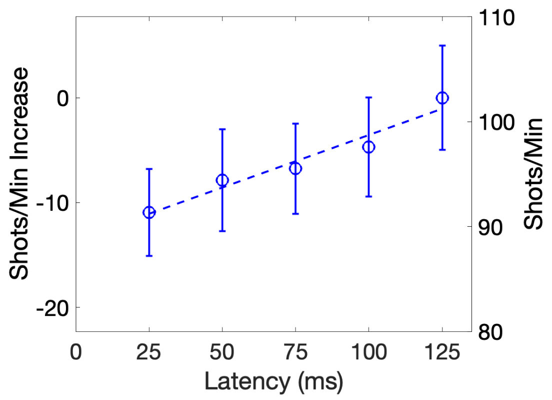


Figure 8.13: Shots fired – Automatic

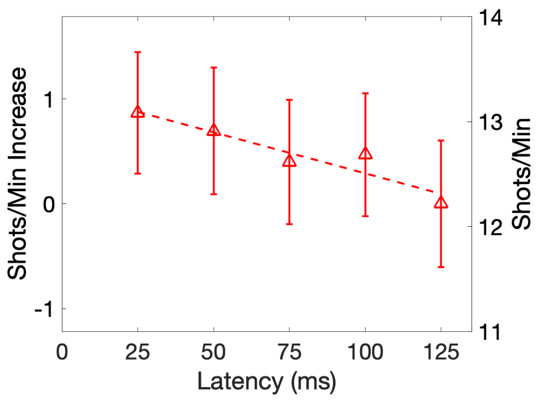


Figure 8.14: Shots fired – Sniper

more often for higher latencies, possibly trying to compensate for the decreased responsiveness. However, from Figure 8.14, the reverse is true for sniper rifles as users fire less often for higher latencies, possible because it is more difficult to align the gun reticle with the target before pulling the trigger.

Figure 8.15 and Figure 8.16 depict avatar movement per minute versus latency for the automatic rifle and sniper rifle, respectively. Movement is computed from the number of times the 'w', 'a', 's', and 'd' keys on the keyboard are pressed, divided by the length of the round (4 minutes). The axes are as for the graph in Figure 8.13. In the case of movement, users with both types of rifles move less often with higher latencies, possibly because the lower responsiveness requires more deliberate movement actions by the players.

Comparative Rifle Performance

Lastly, this section provides a brief analysis of user performance with each weapon type to asses whether good players are skilled with both weapons or if, instead, players tend to specialize and be better at one weapon versus the other. In order to compare performance across latency conditions and weapons, we normalize the data by computing the overall mean for each weapon across all conditions and dividing a

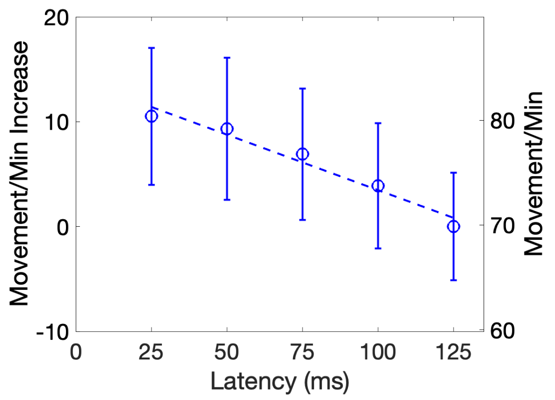


Figure 8.15: Movement – Automatic

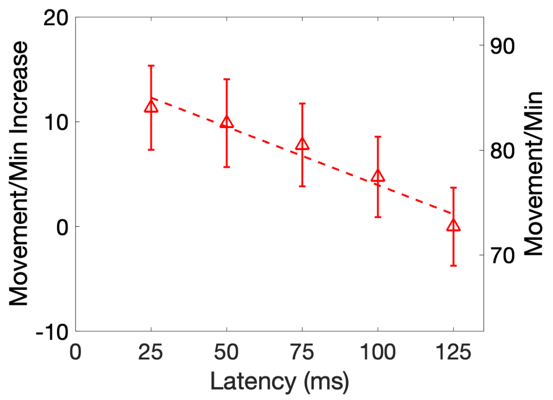


Figure 8.16: Movement – Sniper

user's mean by this total. So, a user with a value of 1.2 is 20% better than average with that weapon and a user with a 0.5 is 50% worse than average.

Figure 8.17 and Figure 8.18 depict scatter plots of the results – sniper rifle versus automatic rifle – for accuracy and score, respectively. Each point is the normalized mean value for one user across all latency conditions. From Figure 8.17, there are few visual patterns for accuracy with the sniper rifle and accuracy with the automatic rifle ($R = 0.21$). However, from Figure 8.18, there is a visual correlation in score for sniper rifles versus score for automatic rifles ($R = 0.86$), with a lack of points in the second and fourth quadrants. These graphs suggest specialization may show up in differences in accuracy, but when it comes to score, players that are good with one weapon are good with another, and vice-versa, regardless of specialization.

8.1.3 CSGO (Local Latency) Study Limitations

While our methodology described in Section 8.1.1 is designed to minimize the differences in the test conditions across participants (e.g., identical laptops, no network connection), the home environment where each user played was not controlled. Users were asked to choose a place with a desk where they could play each session undistributed for the time required for one session (30 minutes), but whether those

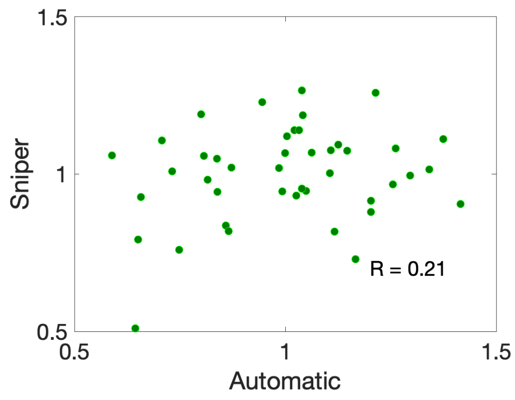


Figure 8.17: Accuracy – Sniper versus Automatic

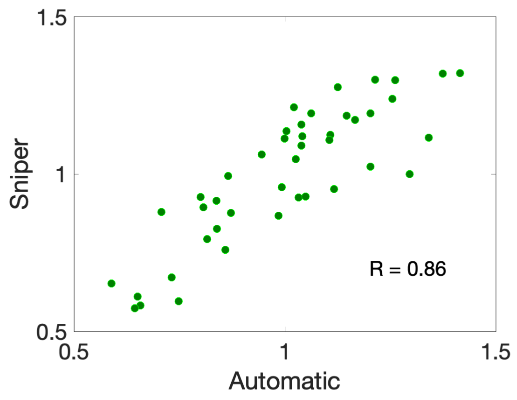


Figure 8.18: Score – Sniper versus Automatic

guidelines were adhered to could not be determined. As such, differences in seating, lighting and noise levels across test locations may have added unknown confounding effects.

Our user study intentionally focused on the effects of latency on individual player actions. However, as noted in Section 8.1.1, CS:GO is often a team game, where groups of players (typically 5 per team in esports) must work together to defeat the opposing team. The impact of latency on CS:GO team efforts, perhaps even team strategies, was not assessed.

As noted in Section 8.1.2, our study is considerably skewed towards males (only 1 female participated). While this may reflect the gender breakdown present in First-Person Shooter games and esports today, the results reported may not be indicative of female performance.

Serious game players often customize the software settings on their computers and games to suit their personal play preferences. For example, players may alter the mouse sensitivity or change the graphics resolution from the system defaults. These custom changes presumably improve the specific player's experience and may improve the player's performance. However, since customizations that deviated from our settings create a difference in test conditions between users, we did not allow any changes to the computer or game settings.

8.1.4 CSGO (Local Latency) Study Summary

This section (Section 8.1) presents results from a user study designed to provide for controlled latency conditions with limited confounding effects. Identical game systems were configured and distributed to qualified users in order for them to provide comparable game data for five levels of local system latency (25, 50, 75, 100 and 125 milliseconds), representing high-end through mid-range game systems. Forty-three people participated in the study, all highly experienced with the game under test: *Counter-strike: Global Offensive (CS:GO)* (Valve, 2012), a popular First-Person Shooter game used in esports. These users each played CS:GO for 25 rounds across 5 different latency conditions with two different weapons (100 total minutes of gameplay), providing objective player performance data (accuracy and score) via logs and subjective opinion data (Quality of Experience, QoE) via surveys.

Analysis of the results shows that across the range of local latencies studied, player performance and quality of experience both improve linearly as latencies decrease from 125 ms to 25 ms. Specifically, player scores at 25 ms average 20% higher than player scores at 125 ms, an equivalent of 5 additional kills or 10 additional assists in a 4 minute game. Over this same range, Quality of Experience (QoE) increases by about the same amount (20%), with the QoE at 125 ms being about 3 (out of 5) and the QoE at 25 ms being about a point better at 4. These same latency reductions impact play with sniper rifles more than automatic rifles, the former weapon requiring more precision than the latter. Latency differences result in different impacts on play characteristics, too, with sniper rifle players shooting less at higher latencies and automatic rifle players shooting more. The results apply to local latencies, typical of high-end to mid-range personal computers. They also pertain to cloud-based game streaming systems with low latencies (e.g., due to edge-clustering), an increasingly important area for commercial game systems and game development.

8.1.5 Simulation Validation (CSGO (Local Latency) Study)

We validate our simulations by comparing the results from our simulations.

We simulate a two player FPS game with same size of map as Mirage and a weapon AWP with the same settings as an AWP: 41 shots per minute and 1 hit kill. There are 9 parameters in our simulation - local latency, network latency, latency compensation techniques, firing rate, number of hits required, target size, map size and player skill. For the simulation, some parameters have exactly the same values as in the user study; Some are not exactly the same but approximately the same; and some factors are different or we not accounted for in the simulation. The three categories of simulation parameters is as below:

A *Parameters that have the same values as in our CS:GO local latency study*:

- Local latency values: 25, 50, 75, 100, 125 ms.
- Network latency: 0 ms.
- Latency compensation: No compensation
- Target size: 180 cm.
- Firing rate: 41 shots per minute
- Map size: 34.4 x 34.4 m.

B *Parameters that have approximately the same values:*

- Number of hits: 1. AWP can kill the enemy with one shot but may needs two shots depends on hit groups (where the hits land on the opponent) and armors.
- Player skill: high. The skill level in our simulation may not exactly match the CS:GO user study participants.

C *Factors that are different or we do not account for:*

- The shape of the map is different. Our map is a square while Mirage has multiple rooms connected by hallways.
- The distribution of obstacle/terrain is different. Mirage has multiple slopes and different shapes of obstacles while our simulation has a room with flat ground and uniform obstacles.
- There are 20 bots in CS:GO user study game which fight with the player avatar and kill each other.

The simulation parameters are included in Table 8.8.

Table 8.8: Parameters values for the CS:GO local latency simulation.

Parameter	Values
Network latency	0 (ms)
Latency compensation	none
Firing rate	41 (shots/min)
Number of hits required	1
Target size	180 cm
Map size	34.4 x 34.4 m (1180 m^2)
Player skill	High
Test player local latency	25, 50, 75, 100, 125 (ms)
Control player local latency	25 (ms)

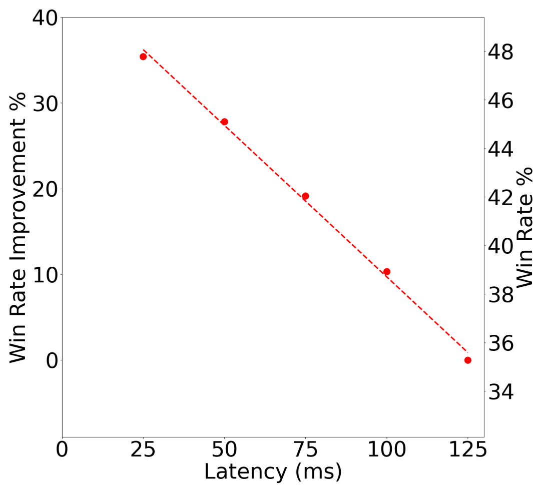


Figure 8.19: CS:GO simulation

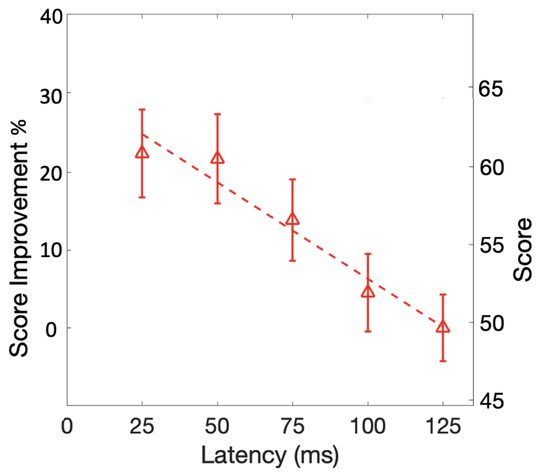


Figure 8.20: CS:GO user study

Figure 8.20 depicts how a reduction in latency improves player performance. The graph is as Figure 8.7 but the y axis on the left is score improvement percent. From this graph, when local latency reduces from 125 ms to 25 ms, player performance improves about 25%.

Figure 8.19 depicts the result of the simulation. The x axis is total system latency in milliseconds. The y axis on the left is the win rate improvement percent and the y axis on the right is the win rate. The circles are the means for the latency condition on the x axis. The dashed line shows a linear regression for the mean values. The regression fits the mean values well, with an R^2 of 0.99 and $p < 0.001$. The slope of the red line is -0.12 and the intercept on y axis is 50.7. From this graph, when local latency reduces from 125 ms to 25 ms, player performance improves about 34%.

Similar to Figure 8.20 from our CS:GO study on local latency, player performance degrades linearly in the range of 25 - 125 ms. With local latency decreases from 125 ms to 25 ms, player performance improves about 34% in our simulation and 25% in our CS:GO study. Local latency helps player performance in the simulation more, perhaps because CS:GO game has 20 bots which makes the game more difficult and leads to more deaths by the player. With each death, players have to respawn and walk to find the next fight. As a result, players spend less time engaged in fights. In our simulation, players never take time to respawn and reset.

8.2 The Effects of Network Latency on CS:GO Game Players

This section presents results from a 25-person user study that evaluates the impact of network latency on experienced *Counter-strike: Global Offensive* players. Analysis of the results shows pronounced benefits to player performance (accuracy and score) for even small reductions in network latency, with subjective opinions on Quality of Experience (QoE) following suit. Latency compensation significantly improves player performance and QoE. We compare the results to our simulation and find that the trends of player performance versus latency hold.

8.2.1 CSGO (Network Latency) Study Methodology

To investigate how network latency affects CS:GO players, we configured a client-server system with CS:GO, added controlled amounts of network latency, recruited players for a user study, and measured player performance and quality of experience.

Our user study was conducted in a dedicated, on-campus computer lab using a client-server architecture shown in Figure 8.21. The server hosts the game and is connected via high-speed LAN to the client. The client and server are Alienware PCs with Intel i7-4790K CPUs @4 GHz with 16 GB RAM and an Intel(R) HD 4600 graphics card. The client is equipped with a gaming mouse and high-refresh rate monitor so as to minimize local system latency. The client has a 24.5” Lenovo LCD monitor with 1920x1080 pixels at 240 Hz and a G502 laser mouse with 12k

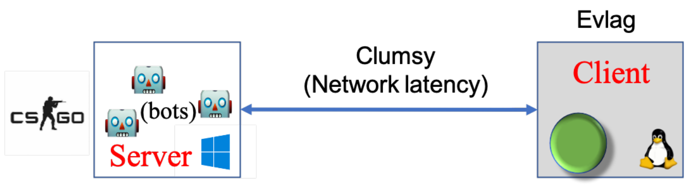


Figure 8.21: CS:GO computer configuration

DPI, 300 IPS, and a 1 KHz polling rate. The client runs Ubuntu 20.04 LTS, with Linux kernel version 5.4 and the server runs Windows 10. Both server and client run *Counter-strike Global Offensive (CS:GO)* (version 10.15.2020). Users were given wired Apple airpods for audio. The base system latency was measured the same way as indicated in Section 5.1.2. This measurement method was done 10 times on our client, yielding an average base latency of 24.6 milliseconds, with a standard deviation of 3.4 milliseconds.

We added network latency to the server uplink and downlink equally using Linux tc with Netem[1] – a network control tool The network latency added to the client was one of 25, 50, 100, or 150 milliseconds. The added network latency is in addition to the base system latency. Thus, the user always experiences 24.6 milliseconds of base system latency from the client computer and actions sent to the server have the additional network latency added to them. For example, the minimum network condition we test is 25 milliseconds. With this condition, the player has a base latency of 24.6 milliseconds at the client and an additional 25 milliseconds of network latency for messages sent to the server. We do not have results with 0 ms network latency – such a condition is only for LAN games, not a typical network game over the Internet.

During our experiments, we gathered ping times from the client collected 5 times every second for every player for every round of game play. The network latency

[1]`https://wiki.linuxfoundation.org/networking/netem`

Table 8.9: Subjective questions per round

	Rate:	Source
Q1	The quality of the round	Stadia [Goo20]
Q2	The responsiveness of the round	Long [LG19]
Q3	Your annoyance with the unresponsiveness	GEQ [PdI07]
Q4	The inconsistency of the round	Custom
Q5	Your annoyance with the inconsistency	GEQ [PdI07]
Q6	How capable and effective you felt	PENS [RRP06]
Q7	How fun the round was	GEQ [PdI07]
Q8	Your frustration in the round	iGEQ [PdI07]
Q9	How much your performance was due to you	Attribution [DM17]

observed by these ping values closely matches the intended added latency (over 99% of values are within 1 millisecond of what is intended). Variations to this are within normal system variations observed by ping with no added network latency and are indistinguishable from the latency variation caused by the LAN itself. The LAN latency was less than 1 ms.

The game has the same map, game and game settings as in Section 8.1.1.

CS:GO includes a server configuration option for the time warp latency compensation technique [LXC21]. With time warp, the server resolves a shot based on the timestamp when the player fires instead of when the server receives the event. Time warp is enabled by default, but can be disabled. CS:GO has another latency compensation technique called interpolation – where the player position is smoothed out based on past positions – that cannot be disabled.

The CS:GO settings were pre-configured at the server with the experiment controlled by scripts on the client – this meant when starting the study, users immediately joined and launched into the game, bypassing normal game lobbies and weapon selection phases.

The IRB-approved user study was conducted during the COVID pandemic, so everyone wore masks and respected social distancing requirements. Upon completion of each user's study, we carefully sanitized the keyboard, mouse and earphones.

A user study proctor was available for questions and trouble-shooting during the experiment.

Users first did a custom reaction-time test 10 times as described in Section 5.1.2 on page 37.

Users played a practice round without any added network latency to get familiar with the map and game mode. This data was not analyzed. Users then played additional 3.5 minute rounds of CS:GO, each round with a different network latency (25, 50, 100, or 150 milliseconds) and with latency compensation either on or off, randomly shuffled.

After each round, users filled out a subjective survey consisting of nine questions on a discrete 5-point Likert scale about the game experience in the preceding round. The abbreviated questions are shown in Table 8.9. The complete questions and answers are available on our website.[2]

After completing the survey, the next round commenced when the user was ready.

After completing all the game rounds, users were given a questionnaire with additional demographics questions.

Study participants were solicited via University email lists. Interested participants first filled out a screener questionnaire to ensure appropriate CS:GO experience (at least 100 hours). Users were rewarded with a $10 USD Amazon gift card upon completion of the study.

[2]`https://web.cs.wpi.edu/~claypool/papers/csgo-net-21/`

Table 8.10: Demographics

Users	Age (yrs)	Gender
25	20.8 (3.0)	25 male, 0 female

FPS Self-rating	CS:GO Self-rating	FPS Hours	CS:GO Hours	Reaction-time (ms)
4.4 (0.7)	4.6 (0.5)	2436 (3866)	832 (703)	205 (24)

8.2.2 CSGO (Network Latency) Study Analysis

This section first summarizes participant demographics, then the effects of network latency on player performance, with and without latency compensation, and Quality of Experience.

Demographics

Twenty-Five (25) users were screened to participate in the user study out of 128 initial responses. Table 8.10 summarizes the participant demographics. FPS self-rating and CS:GO self-rating are on a five-point scale, 1 (low) to 5 (high). For age, FPS self-rating, CS:GO self-rating, CS:GO hours played, and reaction times, the mean values are given with standard deviations in parentheses. Ages ranged from 17-29 years old, typical of a University subject pool. All participants were male – while disappointed there were no female participants, we note esports players are mostly males, especially for FPS games [Sta20]. User self-ratings as FPS and CS:GO gamers both skewed towards "high" (mean 4.4 and 4.6 out of 5, respectively). Half of the users played 10 or more hours of computer games per week.

Figures 8.22, 8.23 and 8.24 depict boxplot distributions for FPS hours played, CS:GO hours played and reaction times, respectively. Each box depicts quartiles and median for the distribution. The whiskers span from the minimum to the

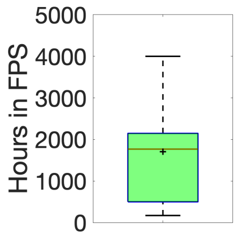


Figure 8.22: FPS hours

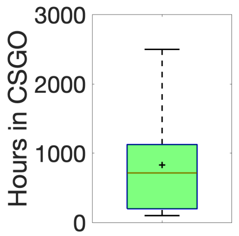


Figure 8.23: CS:GO hours

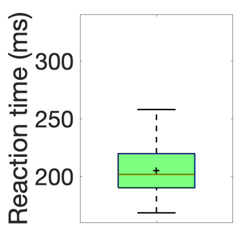


Figure 8.24: Reaction time

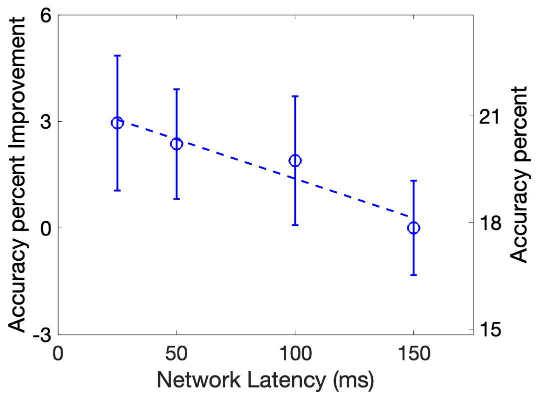


Figure 8.25: Accuracy (means with 95% confidence intervals)

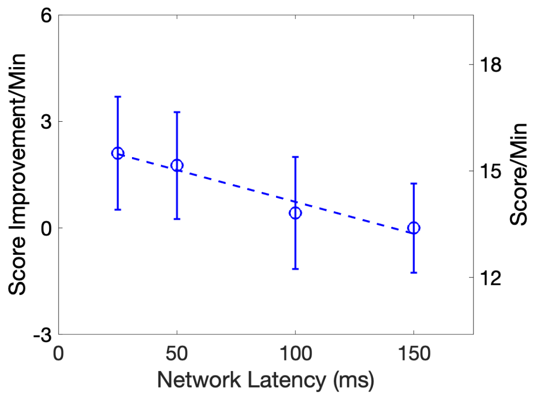


Figure 8.26: Score (means with 95% confidence intervals)

maximum. The black pluses shows the mean values. Most users played from 500-2250 hours of FPS games and from 100-1100 hours of CS:GO. Reaction times were mostly fast – most between 195 and 220 ms – typical of experienced computer game players [DGB09] and about 80 ms faster than the average reaction time collected by the human benchmark site [Hum].

Player Performance

Figure 8.25 depicts weapon accuracy versus network latency on the x axis (the 25 ms system latency is not included). The right y axis is the weapon accuracy (percent) and the left y axis is the percent increase from the 150 ms latency condition. For example, an accuracy of 15 percent at 150 ms of latency compared to an accuracy of 20 percent at 25 ms of latency would be a 5 percent improvement on the left y axis. The circles are the means for all users for that latency condition, bounded by 95% confidence intervals. The dashed line shows a linear regression for the mean values. The regression fits the mean values well, with an R^2 of 0.93 and $p = .04$. As a take-away, a decrease in network latency by 100 ms improves accuracy by an average of about 2 percent.

Figure 8.26 depicts player score versus latency. The axes and points are as in Figure 8.25, but the data is the score ($2 \times kills + assists$) per minute instead of accuracy. The linear regression fits the mean values well, with an R^2 of 0.96 and $p = .002$. As a take-away, a decrease in latency by 100 ms improves player score by 2 points per minute of gameplay. For reference, often less than a single point in a game separates the scores of top CS:GO players.

An effect size provides a measure of the magnitude of difference – in our case, the difference when reducing network latency to the 25 ms base condition. We compare performance with latency to this base condition by independent, 2-tailed t tests ($\alpha = 0.05$) with a Bonferroni correction and compute the Cohen's d effect sizes. The Cohen's d effect size assesses the differences in means in relation to the pooled standard deviation. Generally small effect sizes are anything under 0.2, medium is 0.2 to 0.5, large 0.5 to 0.8, and very large above 0.8. The results are shown in Table 8.11. From the table, while only the 150 ms condition is significant, this is likely due to the sample size and player variation. For both accuracy and score, there is a small effect when reducing latency from 50 ms to 25 ms, a medium effect for 100 ms to 25 ms and a large effect for 150 ms to 25 ms.

Latency Compensation

CS:GO by default has latency compensation on (time warp [LXC21]), but it can be explicitly turned off. Figure 8.27 and Figure 8.28 depict accuracy and score,

Table 8.11: Significance and Cohen's D Effect Size (compared to 25 ms)

	Accuracy			Score		
Latency (ms)	t(22)	p	Effect	t(22)	p	Effect
50	0.750	.460	0.15	0.515	.611	0.10
100	1.196	.244	0.24	2.353	.027	0.47
150	4.623	<**.001**	0.92	3.142	**.004**	0.63

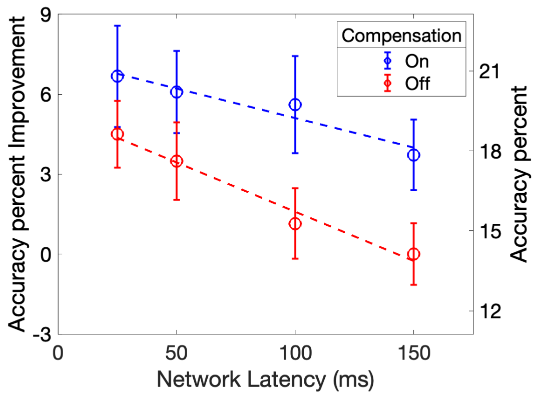


Figure 8.27: Accuracy – latency compensation (means with 95% ci)

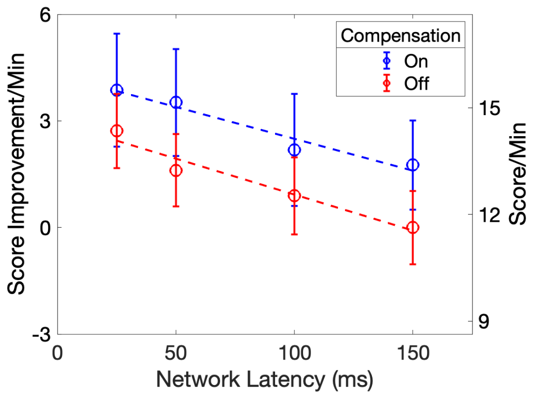


Figure 8.28: Score – latency compensation (means with 95% ci)

Table 8.12: Linear regression for performance

Metric	Compensation	y-intercept	Slope	R^2	p
Accuracy	On	21.45	-0.022	0.93	**.011**
Accuracy	Off	19.42	-0.037	0.98	**.037**
Score	On	15.92	-0.018	0.96	**.023**
Score	Off	14.59	-0.020	0.95	**.024**

Table 8.13: Linear regression for QoE questions

Question	y-intercept	Slope	R^2	p
Q1	4.66	-0.008	0.997	**.001**
Q2	4.71	-0.009	0.997	**.001**
Q3	4.27	-0.007	0.987	**.006**
Q4	4.36	-0.006	0.996	**.002**
Q5	4.40	-0.009	0.999	**.004**
Q6	4.29	-0.005	0.901	.051
Q7	4.31	-0.006	0.993	**.004**
Q8	4.00	-0.006	0.955	**.023**
Q9	4.21	-0.005	0.905	**.049**
Combined	4.36	-0.007	0.994	**.003**

respectively, comparing latency compensation on and off. The axes and points are as in Figures 8.25 and 8.26, with the blue lines denoting latency compensation on and the red off. The results of the linear regressions are provided in Table 8.12, with slope units of percent per millisecond for accuracy and point per millisecond for score. The p values all indicate statistical significance. From the table and figures, there is an observable benefit to using latency compensation for both score and accuracy. As take-aways, 1) accuracy degrades slightly faster with network latency for compensation off than for compensation on, 2) latency compensation improves accuracy by about 19 percent, and 3) latency compensation improves score by about 1.5 points per minute.

Quality of Experience

Quality of Experience (QoE) was assessed from user responses to 9 survey questions filled out at the end of each round. Responses are on a discrete 5-point scale. For the analysis, we rearranged the answers for question 3, 5 and 8 so for all questions, 1 is

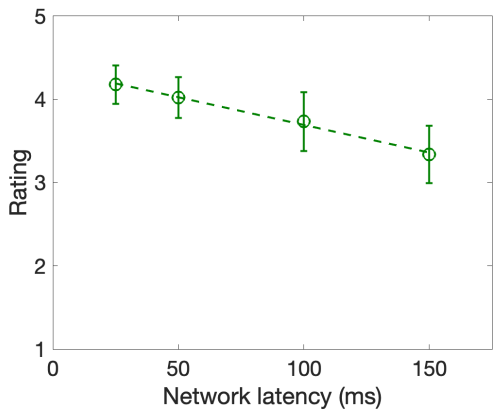


Figure 8.29: QoE – combined questions (means with 95% ci)

low (worse) and a 5 is high (better). Table 8.13 shows linear regression parameters fitting the means for each question. From the table, QoE degrades with latency for all questions – the linear regressions fit the mean ratings well for all questions, with R^2 values from 0.901 to 0.999. All values are statistically significant except for question 6 (capable and effective).

For an overall measure of QoE, we compute the overall mean rating (i.e., weighting all questions equally). Figure 8.29 depicts the results. The x axis is the network latency in milliseconds and the y axis is the rating. The circles are the means for all users for that latency condition, bounded by 95% confidence intervals. The dashed line is a linear regression fit through the mean values. The linear regression fits the means well, with R^2 0.99 and $p = .003$. A one-way between subjects ANOVA shows a significant effect of latency on combined QoE rating at the 0.05 significance level for the four conditions, F(3, 96) = 5.85, $p < .001$. As a take-away, a decrease in latency by 100 ms improves QoE by 0.7 points on a 5-point scale.

8.2.3 CSGO (Network Latency) Study Limitations

Our user study intentionally focuses on the effects of latency on individual player performance. However, as noted in Section 8.1.1, CS:GO is often a team game, where groups of players (typically 5 per team in esports) work together to defeat the opposing team.

As noted in Section 8.2.2, our study is skewed towards males (no females participated). While this may reflect the gender breakdown of FPS games today, the results may not be indicative of female performance in competitive FPS games.

Our study intentionally isolated CS:GO play to a single weapon type only – the most popular [HLT20b] AK-47 rifle – whereas players typically can choose from a variety of weapons with different firing rates, magazine capacities and damages inflicted.

Most CS:GO games use only human players and not AI-controlled bots, as in our study. However, the relative effects should be similar since latency affects aiming and shooting.

8.2.4 CSGO (Network Latency) Study Summary

We study the effects of latency on competitive First-Person Shooter (FPS) game players. We setup a testbed for CS:GO with four levels of network latency (25, 50, 100 and 150 milliseconds). Twenty-five (25) experienced CS:GO players participated in a user study, each playing 8 rounds of CS:GO with 4 different latency conditions both with and without latency compensation. In total, the study provides over 10 hours of gameplay with objective player performance data (accuracy and score) via logs and subjective opinion data (Quality of Experience – QoE) via surveys.

Analysis of the results shows that across the range of network latencies studied, player performance and quality of experience both improve linearly as latencies decrease from 150 ms to 25 ms. Specifically, player accuracy at 25 ms is about 3% higher than player accuracy at 150 ms, and scores are 17% higher over the same range, an equivalent of about 1 additional kill or 2 additional assists per minute of gameplay. Over this same range, latency compensation improves player accuracy by about 3-4% and score per minute by about 1.5 points. From 150 ms to 25 ms, Quality of Experience (QoE) increases by about 25%, with the QoE at 150 ms being about 3.3 (on a 5 point scale) and the QoE at 25 ms being about a point better at 4.2.

8.2.5 Simulation Comparison (CSGO (Network Latency) Study)

With simulations, we are able to simulate first-person shooter scenarios. For example, a CS:GO game with the AK-47 automatic rifle - the same setting as our CS:GO study with network latency. This section presents comparison of the results from the simulation and the CS:GO network latency study.

We simulate a two-player FPS game with same size of map as Mirage and a weapon AK-47 with the same settings as an AK: 600 shots per minute. There are 9 parameters in our simulation - local latency, network latency, latency compensation techniques, firing rate, number of hits required, target size, map size and player skill. For the simulation, some parameters have exactly the same values as in the user study; some are not exactly the same but approximately the same; and some factors are different or we not accounted for. The three categories of simulation parameters is as below:

A *Parameters that have the same values as in our CS:GO local latency study*:

- Local latency values: 0 ms.

  - Network latency: 25, 50, 75, 100, 125 ms.
  - Firing rate: 600 shots/minute
  - Target size: 180 cm.
  - Map size: 34.4 x 34.4 m.

B *Parameters that have approximately the same values:*

  - Latency compensation: since self-prediction in CS:GO cannot be turned off, there are two latency compensation techniques condition in our simulation - 1) self-prediction only, 2) both self-prediction and time warp. However, CS:GO also has interpolation [LXC21] as one of the built-in compensation techniques which cannot be turned off.
  - Number of hits: We set number of hits require to be 4 - it can take 1 - 4 hits to kill the opponent with weapon AK-47 depends on whether the target is armored and the hit groups (e.g., headshots).
  - Player skill: high. The skill level in our simulation may not exactly match the CS:GO user study participants.

C *Factors that are different or we do not account for:*

  - The shape of the map is different. Our map is a square while Mirage has multiple rooms connected by hallways.
  - The distribution of obstacle/terrain is different. Mirage has multiple slopes and different shapes of obstacles while our simulation has a room with flat ground and uniform obstacles.
  - There are 20 bots in CS:GO user study game which fight with the player avatar and kill each other.

Table 8.14: Parameters values for the CS:GO network latency simulation.

Parameters	Values
Latency compensation	self-prediction only, both self-prediction and time warp
Firing rate	600 (shots/min)
Number of hits required	4
Target size	180 cm
Map size	34.4 x 34.4 m (1180 m^2)
Player skill	high
Local latency	25 (ms)
Test player network latency	25, 50, 100, 150 (ms)
Control player network latency	0 (ms)

The simulation parameters are included in Table 8.14.

Figure 8.31 depicts how a reduction in latency improves player performance with different latency compensation conditions in the CS:GO study. The graph is as Figure 8.28 but the y axis on the left is score improvement percent. From this graph, when latency reduces from 150 ms to 25 ms, player performance improves about 20% with compensation off and 15% with compensation on. Since CS:GO has built in self-prediction which cannot be turned off, compensation off is "self-prediction only" and compensation on is "both self-prediction and time warp on" from our simulation.

Figure 8.30 depicts the result of the simulation. The x axis is total network latency in milliseconds. The y axis on the left is the win rate improvement percent and the y axis on the right is the win rate. The circles are the means for the latency condition on the x axis. The dashed line shows a linear regression for the mean values. Red is for time warp off and self-prediction on, and blue is for time warp on and self-prediction on. The latency compensation conditions are as Figure 8.31. The regression fits the mean values well for both of the compensation condition, with an R^2 of 1 and $p < 0.001$ for the red line, and an R^2 of 0.99 and $p < 0.001$ for

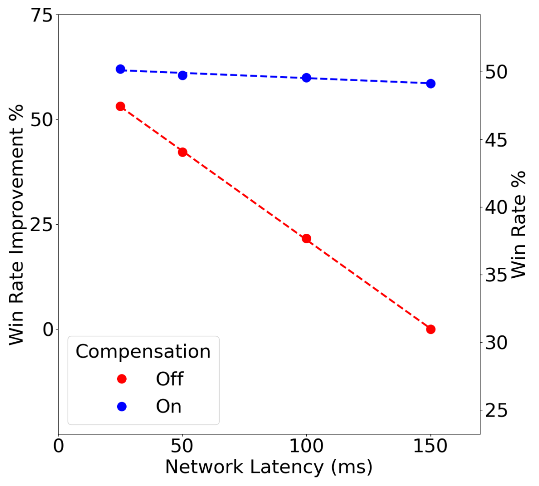


Figure 8.30: CS:GO simulation

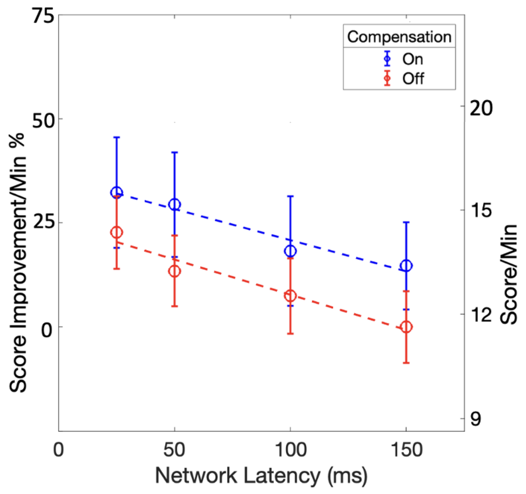


Figure 8.31: CS:GO user study

the blue line. The slope of the red line is -0.13 and the intercept on the y axis is 50.7. The slope of the blue line is -0.005 and the intercept on the y axis is 49.9. From this graph, when latency reduces from 150 ms to 25 ms, player performance improves about 50% with compensation off. Moreover, with both compensation techniques on, player win rate almost does not change with latency.

Similar to Figure 8.31 from our CS:GO study on network latency, with time warp off, player performance degrades linearly in the range of 25 - 125 ms. With compensation off and latency decreases from 150 ms to 25 ms, player performance improves about 50% in our simulation and 20% in our CS:GO study. The reason of the lower impact for CS:GO might due to the shorter proportion of time actively engaging in fights as mentioned in Section 8.1.5 and that CS:GO has built-in interpolation and our simulation does not. Although there are game parameters in CS:GO deathmatch that we cannot simulate (e.g., number of bots, shape of map, distribution of obstacles), the trend of performance with latency holds with self-prediction on and time warp off. Moreover, from both of the study and the simulation, players perform better with self-prediction and time warp on. With time warp and self-prediction both on, player performance still degrades with latency in our CS:GO study but does not change in our simulation. The reason might be that even with compensation on, players can possibly be killed before their shot information reaches to the server due to the high latency. However, in our simulations, the game ends when one of the players is killed.

8.3 Validation with CS:GO Summary

Table 8.15 summarizes the results from the simulation of the CS:GO game, providing the slope, y-intercept, adjusted coefficient of determination (R^2) and statistical significance (*p value*). Statistical significance is indicated in bold. Overall, as in our CS:GO user studies, player performance degrades linearly with latency. Reductions in latency help player performance in CS:GO studies less than in our simulation possibly because players spend less proportion of their time actively engaging in fights, or because CS:GO has built in interpolation which adds delay to all player actions. In both the CS:GO study and the simulation, time warp helps mitigate the impact of latency on players.

Table 8.15: Validation with CS:GO Summary

metric	Compensation	Data	slope	y-intercept	R^2	p value
local latency	NA	Simulation	-0.12	50.7	0.99	**<.001**
		User study	-0.12	65.2	0.95	**0.005**
network latency	on	Simulation	-0.005	49.9	0.99	**<.001**
		User study	-0.018	15.92	0.96	**0.023**
network latency	off	Simulation	-0.13	50.7	1	**<.001**
		User study	-0.020	14.59	0.95	**0.024**

Chapter 9

Validation: Custom First-person Shooter Game

This chapter presents validation of the simulations with a custom first-person shooter game. We design and conduct a user study with a custom game, analyze the results, then compare the measured results to the same game with simulation using the code from Chapter 7.

9.1 Methodology for Validation with the Custom Game

In order to assess the effects of latency on player performance in a first-person game and validate the simulations of first-person shooter game scenarios, we built a custom first-person shooter game based on the navigation hide-and-seek game and the selection game, implemented two latency compensation techniques commonly used in FPS games (self-prediction and time warp), added controlled amounts of local and network latency, varied game parameters (firing rate, number of hits required to kill the opponent, target size, target speed and map size), recruited participants for a user study, and measured player performance and quality of experience.

9.1.1 A Custom First-Person Shooter Game

We designed and implemented a custom game in Unity with first-person shooter specific settings. Our game is a two-player first-person shooter game, shown via screen shot in Figure 9.2. The goal for each player is to kill the opponent - the other player - as fast as possible. A game round is over after one of the two players is killed or after 40 seconds, whichever comes first. The game has one map, depicted in Figure 9.1. The map is a single, square room, default size of 36 meters in length and width, with multiple obstacles to mimic maps in typical first-person shooter games where terrain can play a role in the combat. The player avatars spawn randomly at one of several fixed spawn locations on the map but not in view of the opposing player. Upon spawning, the game provides a countdown for each player until the round starts. Each player has a pistol type of weapon with unlimited ammo with a default maximum firing rate of one shot every 250 milliseconds. Figure 9.2 shows a screenshot of the game where the player is aiming at the target. The player health, score, and timer are shown at the top of the screen.

The update rate for the game engine is fixed at 50 frames per second. Each frame, the game logs the running score for the players, the 3D position of the players, the 2D position of the opponent (the position of opponent on the screen), whether the opponent is in sight, the fraction of the opponent avatar that is in sight and the keyboard and mouse actions. Moreover, the game logs every shot as a hit or miss with corresponding timestamps.

The game has a client-server architecture typical of most network games where an authoritative server keeps the master world state and communicates state updates

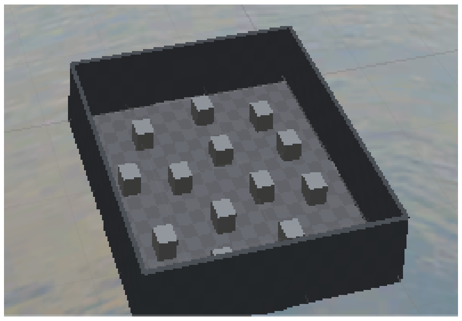


Figure 9.1: The custom FPS game map.

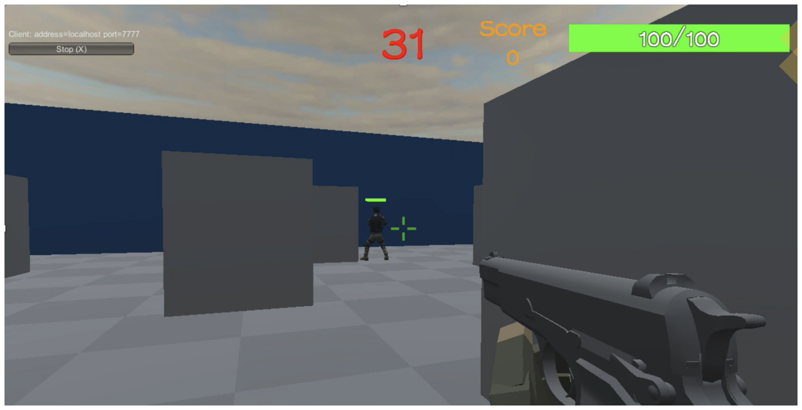


Figure 9.2: The custom FPS game screenshot.

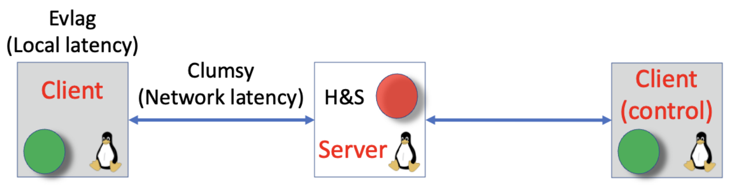


Figure 9.3: First-person shooter game computer configuration.

to the clients. In the default state, without latency compensation, all player input is sent to the server, the server applies the input to the game world and sends the new world state to the client which renders the state for the player.

9.1.2 Testbed Setup for Validation with the Custom Game

We setup the game for our user study in a dedicated, on-campus computer lab. The testbed setup is depicted in Figure 9.3. The server hosts the game and is connected via high-speed LAN to the clients. The clients and server are Dell PCs with Intel i7-8700K CPUs @3.70 GHz 6 cores with 64 GB RAM and an NVIDIA GeForce GTX 1080 graphics card. The clients are equipped with a gaming mouse and monitor so as to minimize local system latency and maintain consistency. The clients have a 25" Lenovo Legion monitor running at 1920x1080 pixels displayed at 16:9 and 240 Hz, with AMD FreeSync and a 1 ms response time. The mouse is a Logitech G502 12k DPI with a 1000 Hz polling rate. The clients and the server run Ubuntu 20.04 LTS, with Linux kernel version 5.4.

The local latency was measured as described in Section 5.1.3 on page 41. This measurement method was done 10 times on our client, yielding an average base latency of 25 milliseconds, with a standard deviation of 5 milliseconds.

Local latency delays all input until resulting rendered output, whereas network latency delays receipt of the player's action at the server and subsequent server response to the client. Since the game server is authoritative, the client cannot update the position of an avatar until the server response has arrived. Thus, for the games without latency compensation techniques (as for all client-server games without latency compensation), local latency manifests similarly to network latency. Player orientation input until resulting avatar orientation change seen on the screen is delayed by at least the sum of the local latency and the network latency.

Our intent is to assess local latencies over ranges that might typically be found in personal computers, which range from about 25 milliseconds for a fast gaming system, are around 100 milliseconds for a typical computer system [ISGS15]. We added latency to all mouse and keyboard input using EvLag [LC21a] – an open-source tool for Linux described in Section 5.1.2 on page 39.

Similarly, our intent is to assess network latencies over ranges typically experienced by PC network game players, which can be near 0 milliseconds for a local area network (LAN) game, 100 milliseconds for a reasonable Internet connection, and 200 milliseconds for a slower Internet connection [opt20]. We added network latency to the server uplink and downlink equally using Linux tc with Netem[1] – a network control tool. The total network latency added to the client was either 0 or 150 milliseconds.

Latency compensation techniques can mitigate the effects of network latency on game players. While there are many different types of latency compensation techniques, time warp and self-prediction are among the most commonly used in first-person shooter games [LXC22]. To better understand and quantify how much the compensation techniques help users in first-person shooter games with network latency, we investigated two different latency compensation conditions: none and both time warp and self-prediction together. We implemented the different latency

[1]`https://wiki.linuxfoundation.org/networking/netem`

compensation conditions in our custom selection game.

Game parameters, such as target size and speed, can change the game difficulty and affect player performance and experience. To better understand the effects of game parameters, we studied six game parameters common to first-person shooter - firing rate, how fast the gun can be fired to shoot at the opponent; number of hits, how many shots are required to eliminate the opponent; target size, the size of the player avatar; movement speed, how fast the player avatar can move in the game; and map size, how large the game map is.

Table 9.1 summarizes the latency values in the user study for both of the players. For the test player, we vary both local latency and network latency. For the control player, there is no extra latency added - the local latency is fixed at 25 ms and the network latency is fixed at 0 ms across all games. Table 9.2 summarizes the game parameters for the user study. The game parameters values are the same for both players at all times. The default values of the game parameters are highlighted in bold.

Table 9.1: Latency values for the user study.

Parameters	Test player values	Control player values
Local latency	25, 100 (ms)	25 (ms)
Network latency	0, 150 (ms)	0 (ms)

Before the launch of the formal user study, a pilot study with 3 volunteers was conducted in order to test the viability of the procedure and tune the study settings. The pilot study results helped adjust round length, map size and layout, number of rounds, latency values, game parameter values and user instructions.

Table 9.2: Game parameter values for the user study.

Parameters	Values (same for both players)
Latency compensation	**none**, both time warp and self-prediction
Firing rate	**250**, 1000 (ms)
Number of hits required	**1**, 4
Target size	50, **200** (cm)
Movement speed	5, **10**
Map size	18 x 18 m, **36 x 36 m**

9.1.3 User Study Procedure for Validation with the Custom Game

The study was approved by our Institute Review Board (IRB). Interested participants first filled out a screener questionnaire with questions on first-person shooter game-related experience to help select participants with some prior familiarity with FPS games. Selected users were invited to the lab at a pre-set time, one time slot for two users to play against each other. Users then signed a consent form and positioned themselves at the test computer.

Users first did a custom reaction-time test 10 times as described in Section 5.1.2 on page 37. The average of the 10 values provides a measure of reaction time.

Users started by playing a practice round without any added latency to get familiar with the game. This data was not analyzed. Users next played additional rounds, each with options for local latency, network latency, latency compensation techniques, and the game parameters, randomly shuffled. The control player always had local latency and network latency as in Table 9.1. The conditions tested include:

A *Best condition*: There is 1 condition with both players having local latency of 25 ms, network latency of 0 ms, no latency compensation and default game parameters values.

B *Varying latency conditions*: There are 3 conditions investigating latency conditions - local latency, network latency and latency compensation. We change the values for the latency conditions one at a time. All the other game parameters - firing rate, number of hits required, target size, movement speed and map size - are at their default values. Note that with latency compensation on, network latency is at 150 ms.

C *0 ms network latency and no compensation:* There are 5 conditions assessing game parameters - firing rate, number of hits required, target size, movement speed and map size - without any network latency and latency compensation techniques. The 5 conditions are with fixed local latency of 25 ms, network latency of 0 ms and no compensation techniques. We then vary one of the 5 game parameters in each of the 5 conditions.

D *150 ms network latency and no compensation:* There are 5 conditions assessing the game parameters - firing rate, number of hits required, target size, movement speed and map size - in presence of network latency without latency compensation techniques. The 5 conditions are with fixed local latency of 25 ms, network latency of 150 ms and no compensation techniques. We then vary one of the 5 game parameters in each of the 5 conditions.

E *150 ms network latency and compensation on:* There are 5 conditions assessing game parameters - firing rate, number of hits required, target size, movement speed and map size - in presence of network latency with latency compensation techniques on. The 5 conditions are with a fixed local latency of 25 ms, and network latency of 150 ms and both self-prediction and time warp. We then vary one of the 5 game parameters in each of the 5 conditions.

The best condition (A) is repeated 3 times and all other conditions (B - E) are repeated 2 times, for a total $3 + (3 + 5 + 5 + 5) \times 2 = 39$ rounds plus two practice rounds. The 3 rounds with the base condition are uniformly distributed between all rounds to ensure the consistency of player performance and test if there is fatigue giving the large number of game rounds. The analysis of the 3 rounds is included in Section 9.2. In total, each player played $39 + 2 = 41$ rounds.

After each round, both users provided a subjective Mean Opinion Score (MOS) rating on a discrete 5-point Likert scale about their experience: "Rate the quality of the previous game round". Players chose from 5 options: Excellent, Good, Fair, Poor or Bad. After completing the survey, the next round would commence when both user were ready, but users could take as much time as needed before starting the subsequent round.

It took each pair of users about half an hour to complete all the tasks in the study. A user study proctor was available for questions and trouble-shooting for the duration.

After completing all the game rounds, users were given a questionnaire with additional demographics questions about gamer experience – average time spent playing games and self-rated expertise with computer games.

In summary, the procedure each user followed was:

1. Fill out the screener questionnaire to ensure interest in participation and help understand player game familiarity.

2. Come to the dedicated lab at a pre-set time.

3. Adjust the computer chair height and monitor angle and height so as to be comfortably looking at the center of the screen.

4. Read the instructions regarding setup and game controls on the desktop.

Table 9.3: Demographic information for test players

Group	Users	Age (yrs)	Gender	Gaming per week (hours)	Gamer Self-rating	FPS Self-rating	Reaction-time (ms)
Test	46	19.3 (3.0)	37♂ 6♀ 3 Other	13.6	3.4 (1.1)	2.8 (1.2)	183.3 (29.5)
Control	38	20.7 (3.9)	29♂ 8♀ 1 Other	12.8	3.5 (1.2)	3.1 (1.2)	196.1 (32.4)

5. Complete the reaction-time test. (Takes about 30 seconds.)
6. Complete the first-person shooter game rounds (2 practice round and 39 rounds with shuffled testing conditions), including the QoE surveys after each round. Take breaks between rounds if needed. (Takes a bit less than one hour, total.)
7. Complete the final demographics questionnaire.

Study participants were solicited via university email lists. All users were eligible for a raffle to win a $25 USD Amazon gift card upon completion of the study, and many users received playtesting credit for relevant classes in which they were enrolled.

9.2 Results

9.2.1 Demographic Information

Eighty-four (84) users were recruited and participated in total, with 46 in the test group and 38 in the control group. For 8 sessions, Shengmei played as the control player. This section provides summary demographics for the participants.

Table 9.3 summarizes the demographic information for the user study participants. The first row is for the test group and the second row is for the control group. Gamer and first-person shooter (FPS) self-rating are on a five-point scale,

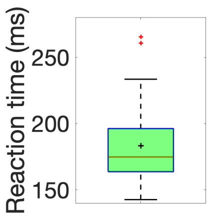


Figure 9.4: Reaction time (ms)

1-low to 5-high. For age, gamer self-rating, FPS self-rating, and reaction times, the mean values are given with standard deviations in parentheses. For the test group of 46 players, ages ranged from 18-27 years old but with the large majority of typical college age. Gender breakdown is predominantly male (37 males), but does reflect the gender breakdown of first-person shooter game players (about 7% of first-person shooter gamers are women [Lee17]) and our sample pool of students at our university. Half of the participants played 10 or more hours of computer games per week. User self-ratings in general computer games slightly skews towards above the mid-point (mean 3.4), with self-rating in FPS games slightly lower (mean 2.8). For the control group of 38 player, ages ranged from 18-34 years old but with the large majority of typical college age. Gender breakdown is also predominantly male (29 males). Half of the participants played 10 or more hours of computer games per week. Average gamer self-rating is slightly higher that the test group (mean 3.5), with self-rating in FPS games following suit (mean 3.1). For both of the test group and test group, most participants majored in Robotics Engineering, Computer Science, or Game Development.

Among the 46 players in the test group, 38 played against another player in the control group and 8 played against me. Statistical analysis shows that there is no significant difference between the performance of the group of test player who played against the control players and the group of test player who played against Shengmei with a p value at 0.69. In the following sections, the data from the 46 test players are analyzed together.

Figure 9.4 depicts the distribution of users' reaction times from the test group as a boxplot. The base local latency (25 ms) was subtracted from all reaction time trials and the resulting reaction times averaged for each user. The box depicts quartiles and median for the distribution. The whiskers span from the minimum non-outlier to the maximum non-outlier. The black plus shows the mean value. From the graph, reaction times are mostly fast (with an average about 183 ms), faster than average computer game players [DGB09].

9.2.2 Fatigue Analysis

As mentioned in Section 9.1.3, to test consistency of player performance across, there are 3 best conditions uniformly spread out during the study. Figure 9.5 depicts player win rate of the 3 rounds. The average win rate decreases with the game rounds but an ANOVA test suggests no significant difference between player performance across the 3 game rounds with a p value at 0.17 and f value at 1.84.

9.2.3 Inconsistency Analysis

As mentioned in Chapter 2, latency compensation technique often trade off consistency and responsiveness. Time warp improves the responsiveness of first-person shooter games, but can cause inconsistency between players. There are mainly two types of inconsistency - the client hits an opponent but the server disagrees; and client misses (e.g., players intentionally aim ahead) but the server registers the hit.

Out of the 92 rounds in total with latency compensation on, the first type of inconsistency (client hits an opponent but the server disagrees) happened 22 times to the test player and the second type of inconsistency (client misses but the server disagrees) happened 8 times to the test player.

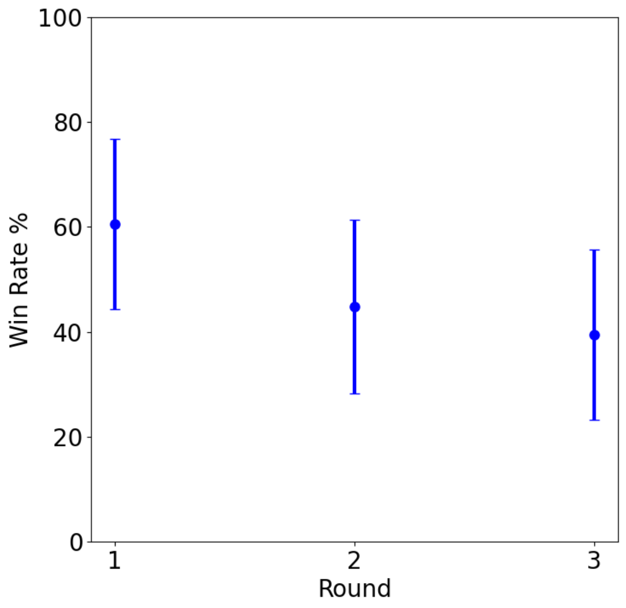


Figure 9.5: Win rate of “best” rounds

9.3 Comparison of the Seeker and Hider Intervals

This section compares the length of seeker and hider intervals from the custom FPS game study with the navigation study from Chapter 5. For each frame, we logged if the opponent is in-sight and the fraction of in-sight.

Figure 9.6 depicts the fraction visible while the opponent is in sight. From the graph, the visible factor does not vary with latency. When the opponent is in-sight, more than 70% of time the avatar is fully visible. For the seeker interval, if any part of the opponent was visible, the opponent is considered in sight.

Figure 9.7 depicts the cumulative distribution function (CDF) of seeker intervals from the navigation study (Chapter 5). From the graph, the vast majority of the seeker intervals are below 2.5 seconds and the medians are less than 200 ms regardless of the latency. The custom FPS game study confirms that there is some visual separation of the lines based on latency, with lower latencies having slightly longer

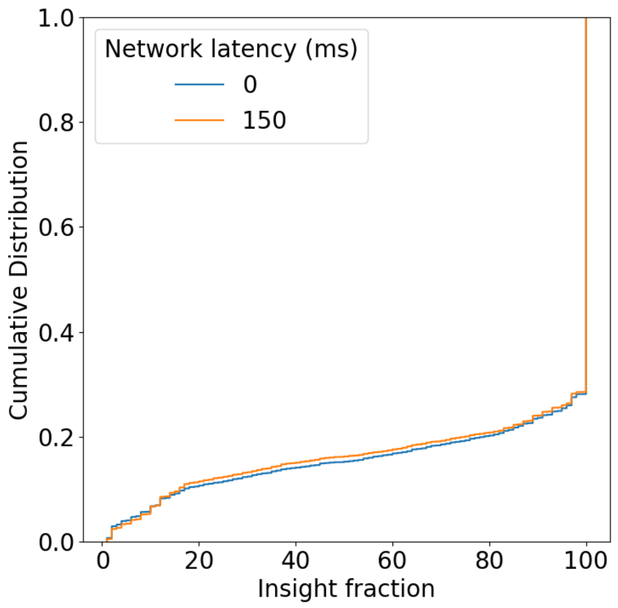


Figure 9.6: Fraction visible

intervals (the lines are shifted down and to the right).

Figure 9.8 depicts the cumulative distribution function (CDF) of the seeker intervals from the custom FPS game study. The graph is as Figure 9.7 but the data is from the custom FPS study and there are only two latency values - 25 (25 local latency and 0 network latency) and 175 ms (25 local latency and 150 network latency) without any latency compensation. Similar to Figure 9.7, the vast majority of the seeker intervals are below 2.5 seconds. There is some visual separation of the lines based on latency, with 0 ms latency having slightly longer intervals than 150 ms latency (the lines are shifted down and to the right). CDF lines in Figure 9.8 and Figure 9.7 have similar shape but seeker intervals from the hide and seek game are shorter in general.

Figure 9.9 depicts the same data as Figure 9.7 but for the hider intervals. From the graph, visually, the hider interval distributions do not separate based on latency.

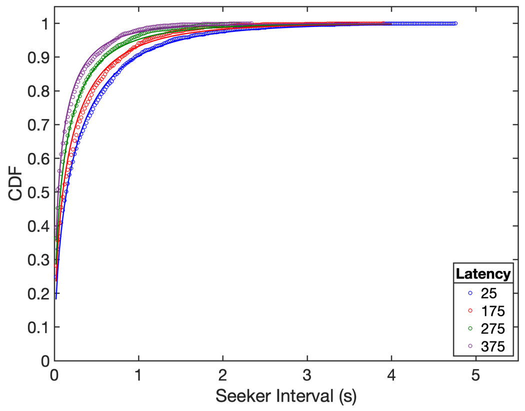


Figure 9.7: Seeker interval distribution from the navigation study

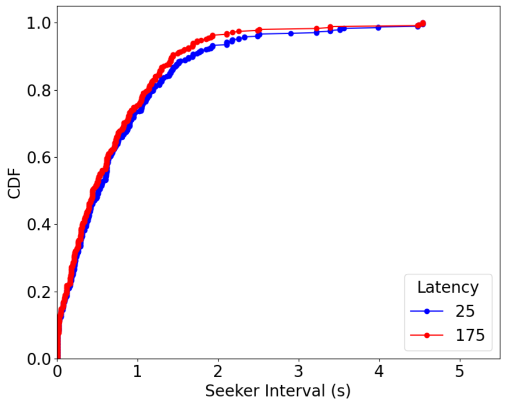


Figure 9.8: Seeker interval distribution from the custom FPS game study

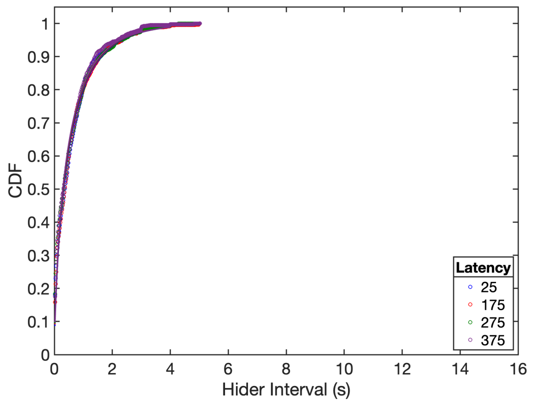


Figure 9.9: Hider interval distribution from the navigation study

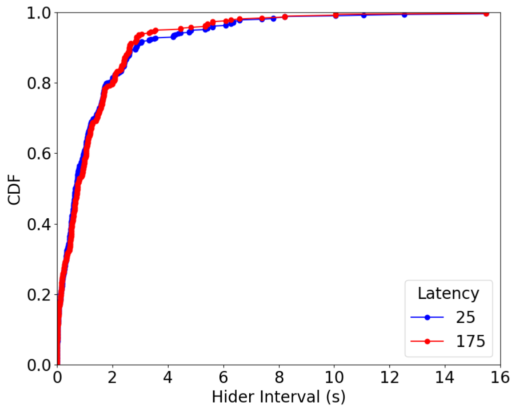


Figure 9.10: Hider interval distribution from the custom FPS game study

The FPS game study confirms the distribution of time durations where a hider cannot be seen is about the same regardless of the latency.

Figure 9.10 depicts the cumulative distribution function (CDF) of the hider intervals from the custom FPS study. The graph is as Figure 9.9 but the data is from the custom FPS study and there is only two latency values - 25 and 175 ms. The hider interval distributions do not separate based on latency. About 90% of hider intervals are below 4 seconds. However, both of the lines have heavy tails - there are some values that are longer than 8 seconds. The reason might be that players occasionally lose track of the opponent for large stretches of time. CDF lines in Figure 9.10 and Figure 9.9 have similar shape but hider intervals from the hide and seek game are shorter in general.

9.4 Comparison with Simulations

We simulate a two-player game with the same settings as in our custom first-person shooter games. There are 8 parameters varied in our simulation - local latency, network latency, latency compensation techniques, firing rate, number of hits required, target size, map size and player skill.

Figure 9.11 depicts the comparison of the study data and the simulations. In this graph, the x axis is the simulation parameters. The y axis on the left is the win rate percent improvement over the base conditions and the y axis on the right is the win rate percent. The circles are the mean values of the user study data and the bars are the 95% confidence intervals bars through the mean values. The order of the x axis parameters are sorted by the mean values. The green bar is the "base" condition - 25 ms of local latency, 150 ms of network latency, all other parameters are the default (compensation off, firing rate at 4 shots/s, number of hits at 1, map size at 36 x 36 m, target size at 2 m and player skill at "all") as indicated by the bold values in Table 9.1 and Table 9.2. With the "best" condition, there is no latency and latency compensation, and all other parameters are at default values. For all other conditions, we have 25 ms local latency and 150 network latency and vary one of the parameters to the other none bold value in Table 9.2 based on the "base" condition. The red plus is the simulated mean values for the condition. Varying the parameters on the right side of dashed green line improves player performance while varying the parameters on the left side of dashed green line degrades player performance.

In the graph, compared to the base condition, number of hits, latency compensation, no latency, and player skill make significant differences on player performance.

Latency has significant impact on player performance. By comparing the "base" condition and "best" condition, reducing network latency by 150 ms improves win rate by 98%. More hits make it harder for players with network latency to win - number of hits increases from 1 to 4, the win rate for the player with latency decreases by 60%. Latency compensation can significantly mitigate the effects of latency on player. With latency compensation on, player win rate improves by 96%. With latency, higher skill players are 37% more likely to win against "all" skill players and lower skill players are 39% less likely to win against "all" skill players on average, but both with a large confidence interval size probably due to lower sample size. Moreover, target size reductions from 200 cm to 50 cm improves the win rate of players with latency by 37%.

Room size and firing rate do not have significant impact on player win rates. Room sizes shrinking from 36 x 36 m to 18 x 18 m improves win rate for the player with network latency by 22%. Firing rate dropping from 250 ms to 1000 ms improves win rate for the player with network latency by 25%. The effects of these parameters

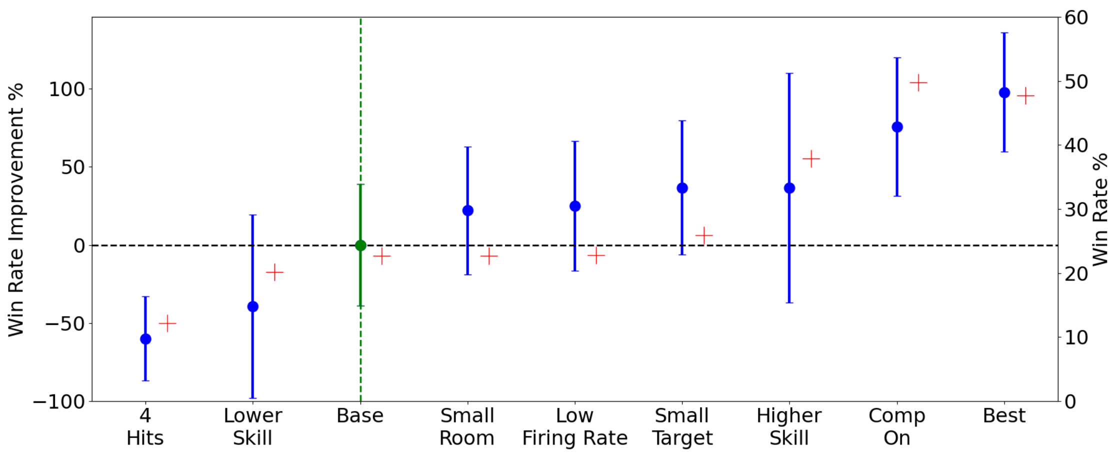


Figure 9.11: Win rate for test player

are low perhaps because the game parameters are the same across the two players.

9.5 Simulations Accuracy

From Figure 9.11, all the red pluses are within the bar range on its left - all the simulated values fall in the corresponding 95% confidence intervals. Our simulation predicts the scenarios from the custom first-person shooter game accurately and can be used for exploration.

Figure 9.12 depicts the accuracy of the simulation for the parameters. The x axis is the 9 parameters - same as in Figure 9.11. The y axis is the absolute difference between the simulated values and the experimental means. The simulation conditions for each parameter are the same as in Section 9.4. For each of the parameters, we vary the parameter value and run 10,000 iterations for 10 trials. For each of the trial, we have a mean win rate value from the iterations. The absolute value of the delta between the mean win rate and mean experimental value (the circles in Figure 9.11) is an absolute difference. We then have 10 absolute difference values for the 10 trials in total. Finally we calculate the mean and confidence

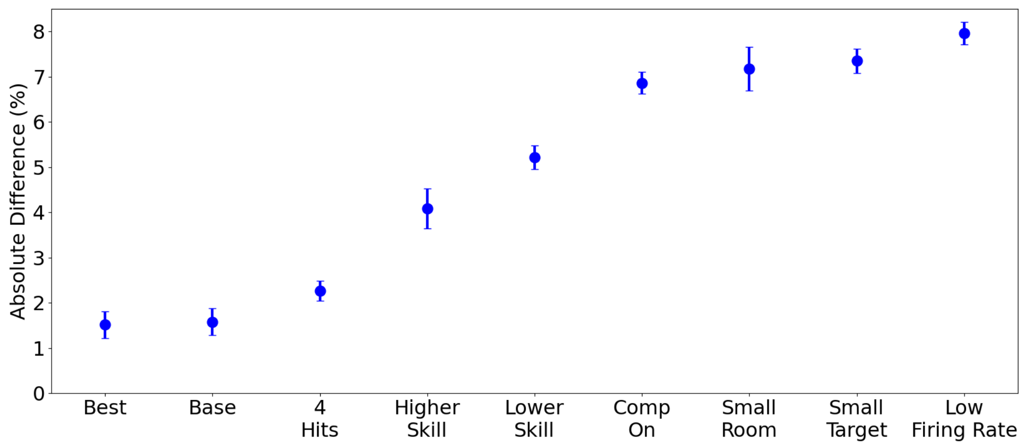


Figure 9.12: Simulation accuracy

intervals of the 10 resulting absolute difference values. The circles are the mean absolute difference and the bars are 95% confidence intervals across the mean values. From this graph, our simulation predicts the win rate percent for the best and the base condition well with mean absolute difference at only about 1.5 percent. The absolute difference between the mean simulated value and the experimental value is about 2.4 percent for number of hits required. The simulation predicts win rate for higher skill and lower skill less well with mean absolute different at about 4 and 5 percent. Latency compensation on has larger difference at about 7 percent probably because players are killed before the hits reach to the server. Small room, small target and low firing rate conditions are the least accurate in our simulation with an absolute different of 7 - 8 percent. In all cases, our simulation values are within 10% of the measured user study values.

Chapter 10

Exploration

With validated simulations from Section 7, we are able to explore the space of first-person shooter games by varying the game parameters - latency, latency compensation techniques, number of hits, firing rate, map size, target size and player skill. This chapter describes how different game parameters change the impact of latency on player performance in first-person shooter games.

We first simulate a two-player first-person shooter game with parameters as in Table 10.1. This is the "base" exploration. We then vary the parameters one at a time from Section 10.2 to 10.8 and observe the trends in player performance versus latency.

10.1 Performance and Latency

We vary the network latency for the first player using the base settings. Figure 10.1 depicts the win rate of the test player (the lagged player) versus total latency. The x axis is network latency and the y axis is the win rate. From this graph, player win rate degrades with latency. The trend appears mostly linear over the simulated range. At 275 ms of latency, the win rate is only about 4%. Win rate percent drops

Table 10.1: Two-player FPS Game parameters values for the "base" exploration.

Parameter	Test player	Control player
Network latency	0 - 150 (ms)	0 (ms)
Local latency	25 (ms)	25 (ms)
Player skill	all	all
Latency compensation	none	
Firing rate	250 (ms) (4 shots per sec)	
Number of hits required	1	
Target size	200 cm	
Map size	36 x 36 m	

about 17 for each 100 ms of latency.

10.2 Latency Compensation Techniques

We analyze latency compensation using the base setting with four conditions: no compensation, self-prediction only, time warp only, both self-prediction and time warp.

Figure 10.2 depicts win rate with latency under different latency compensation conditions. The trendlines are separated out by the four latency conditions: blue is without latency compensation (the same line as in Figure 10.1), orange is for self-prediction, green is for time warp, and red is for both self-prediction and time warp. In the graph, the blue line has the steepest slope, showing that latency has the most impact on player performance when there is no latency compensation. The green line and the orange lines have shallower slopes and are comparable, indicating that each technique individually has about the same ability to mitigate network latency. The red line is almost flat, indicating that both techniques together can nearly completely overcome the effects of network latency on player performance.

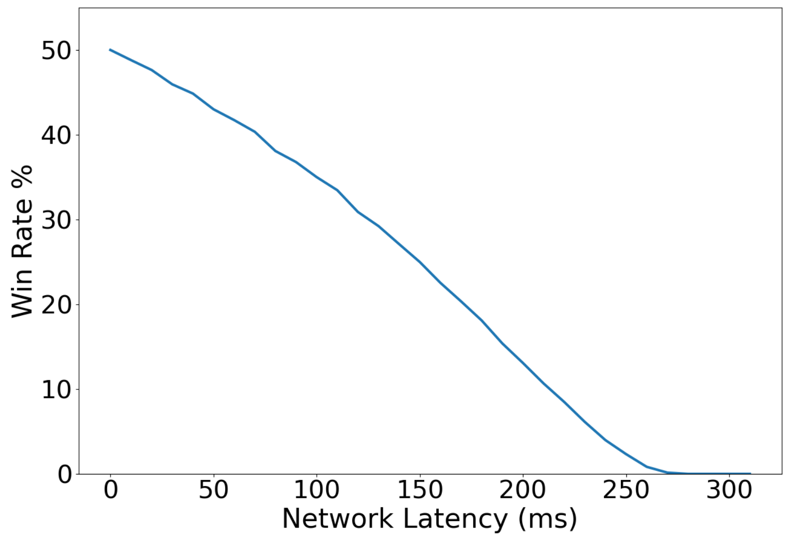


Figure 10.1: Simulated win rate for test player

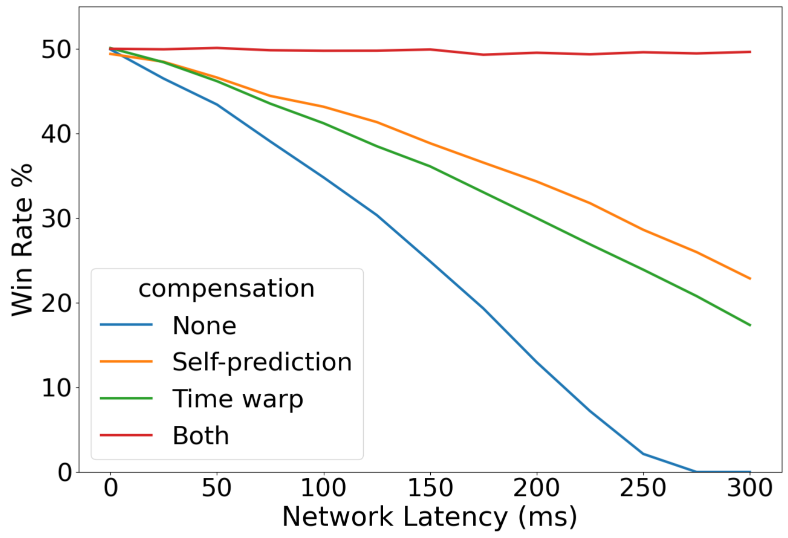


Figure 10.2: Simulated win rate for test player - compensation

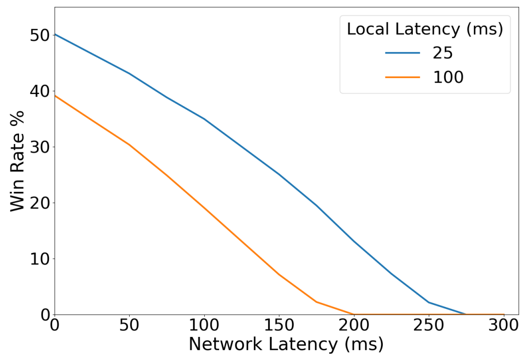


Figure 10.3: Simulated win rate for test player - local latency and no latency compensation

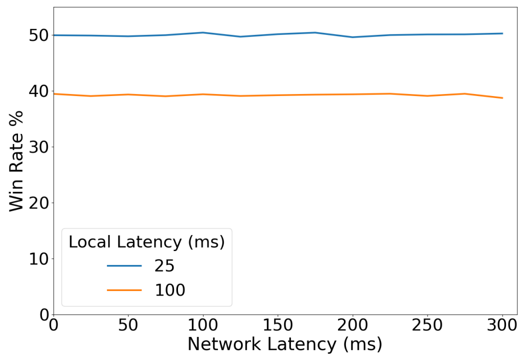


Figure 10.4: Simulated win rate for test player - with local latency both time warp and self-prediction

10.3 Local latency

We vary local latency for the test player from the base setting with two values: 25 and 100 ms.

Figure 10.3 depicts win rate versus network latency with the two different local latencies and no latency compensation. The trendlines are separated out by the two local latency values: blue is for local latency at 25 ms (the same line as in Figure 10.1) and orange is for local latency at 100 ms. In the graph, the orange line is above the blue line, showing that with the same amount of network latency, players perform worse with higher local latency. Both of the lines degrade similarly with network latency.

Figure 10.4 is as Figure 10.3 but the simulation is with both time warp and self-prediction on. In the graph, the slope of the two lines are almost flat, showing that with latency compensation on, player performance is not affected by network latency and equalizes at the local latency. The blue line is on top of the orange line, showing that player performance still degrades with local latency - latency compensation cannot mitigate the effects of local latency on player performance.

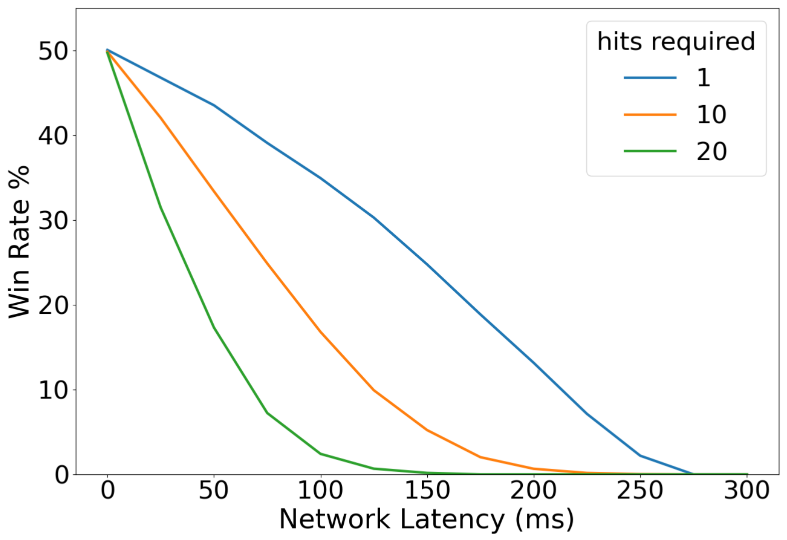


Figure 10.5: Simulated win rate for test player - number of hits required

10.4 Number of Hits

We vary number of hits required for a kill for both players from the base setting with three values: 1, 10 and 20.

Figure 10.5 depicts win rate versus latency with the different numbers of hits required. The trendlines are separated out by the three values: blue is for 1 hit (the same line as in Figure 10.1); orange is for 10 hits; and green is for 20 hits. In the graph, the green line has the steepest slope, the orange line has a shallower slope and the blue line the shallowest. Player performance is more impacted by latency the greater the number of hits required for a kill.

10.5 Firing Rate

We vary the weapon firing rate for both players from the base setting for three values: 250 ms (4 shots/sec), 500 ms (2 shots/sec) and 750 ms (1.33 shots/sec).

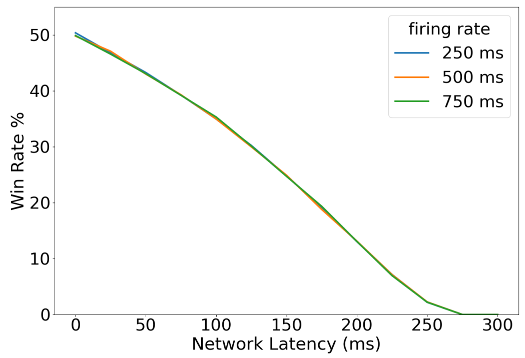


Figure 10.6: Win rate with firing rate (number of hits required to kill is 1)

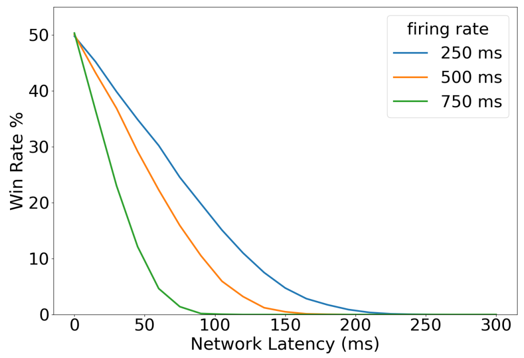


Figure 10.7: Win rate with firing rate (number of hits required to kill is 10)

Figure 10.6 depicts win rate versus latency with different firing rates when number of hits required to kill is 1. The trendlines are separated out by the three firing rate values: blue is for 250 ms (4 shots/sec) (the same line as in Figure 10.1); orange is for 500 ms (2 shots/sec); and green is for 750 ms(1.33 shots/sec). In the graph, the three line largely overlap with each other, showing that firing rate does not have a significant effect on the impact of latency on player performance when there is only one hit required to kill the enemy.

Figure 10.7 is as Figure 10.6 but the number required to kill is 10. This graph shows that the firing rate does affect how latency impacts player performance when more slots are required for a kill. In this case, high firing rates are significantly more affected by latency than low firing rates.

10.6 Map Size

We then vary map size from the base setting with three values - 9 x 9 m, 36 x 36 m and 50 x 50 m.

Figure 10.8 depicts win rate versus latency with different map size values. The trendlines are separated out by the three map size values: orange is for 9 x 9, blue

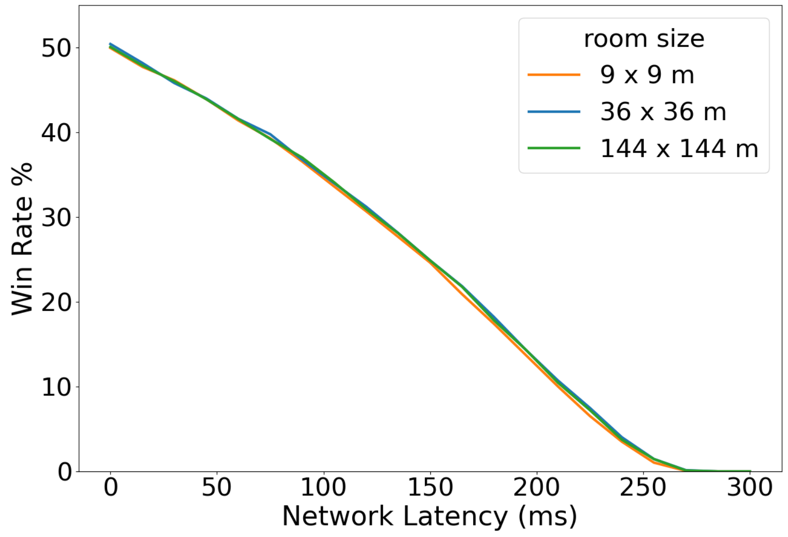


Figure 10.8: Simulated win rate for test player - map size

is for 36 x 36 (the same line as in Figure 10.1) and green is for 144 x 144 m. In the graph, the three line largely overlap, showing that map size does not have significant effect on the impact of latency to player performance.

10.7 Target Size

We vary target size for both players from the base setting with three values: 50 cm, 200 cm and 300 cm.

Figure 10.9 depicts win rate versus latency under different target size values. The trendlines are separated out by the three target size values: orange is for 50 cm, blue is for 200 cm (the same line as in Figure 10.1), and green is for 300 cm. In the graph, the three line largely overlap with each other, showing that target size does not have significant effect on the impact of latency to player performance.

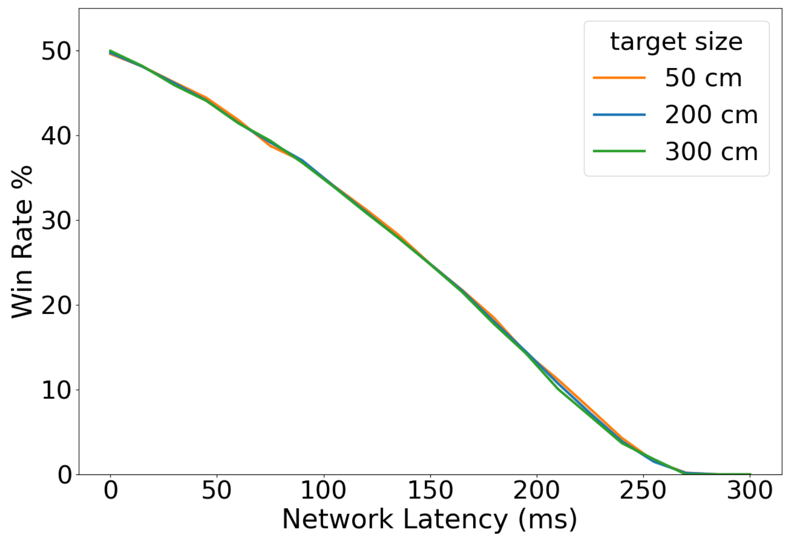


Figure 10.9: Simulated win rate for test player - target size

10.8 Player Skill

We vary player skill for both players from the base setting with two values - lower skill and higher skill.

Figure 10.10 depicts win rate versus latency with different player skills. The trendlines are separated out by the two player skills: green is for lower skill and orange is for higher skill. In the graph, both of the lines decrease with latency with orange line on the top. Although higher skill can help players performance better, player performance degrades with latency regardless of player skill. At the base condition, higher skill players are about 15% more likely to win. At about 300 ms of latency, lower skill players have almost no chance of winning comparing to about 450 ms of latency for higher skill players.

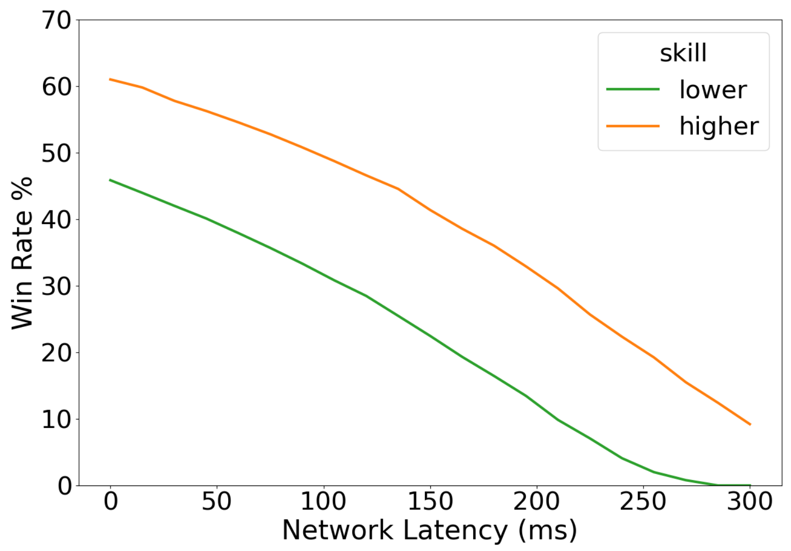


Figure 10.10: Simulated win rate for test player - skill

10.9 Comparison of Parameter Effects

To compare the relative effects of the parameters on player performance with latency, we update the parameter values an order of magnitude more than the base values (if applicable) one at a time, and observe the difference between the simulation results. To get the new values for the parameters, we multiply network latency, local latency, firing rate, number of hits and map size in the base condition by 10. For target size, we divide the value by 10 instead - in first-person shooter shooter games, this might represent a head as a target. For player skill, the updated value is high. Since number of hits matter for the effects of firing rate, we explored firing rate with 1 and 10 as the number of hits required to kill respectively. For latency compensation, the updated value is time warp and self-prediction both on.

Table 10.2 depicts the simulation results. In the table the first column includes the simulation parameter, the second column is the default value of the parameter - the same as the base condition - the third column is the simulation results with the default values of the parameters, the forth column is updated value for the parameter, the fifth column is the simulation results with the updated value of the parameter, and the last column is the absolute difference between simulation results with different values of the parameter - the absolute different between column 3 and column 5. The rows are sorted by the last column from high to low - the absolute difference. The rows with absolute difference over 20 are marked red, 10 - 20 are marked yellow and 0 - 10 are green.

By comparing the absolute difference caused by the parameters (column 6), network latency, local latency, latency compensation and number of hits have the largest impact on the effects of latency on player performance with absolute difference above 20. Player skill has lower impact with absolute difference at about 15. Firing rate, target size and map size have minimal impact.

Table 10.2: Simulation results for test player with default values and 10 x values of parameters

Parameters	Default values	Win rate %	Updated values	Updated win rate %	Absolute difference
Network latency	150 ms	24.98	1500 (ms)	0	24.98
Local latency	25 (ms)	24.98	250 (ms)	0	24.98
Latency compensation	none	24.98	both	49.5	24.52
Number of hits	1	24.98	10	4.64	20.34
Player skill	all	24.98	high	40.86	15.88
Firing rate (w/ 10 hits)	4 (shots/sec)	4.62	40 (shots/sec)	8.99	4.37
Map size	36 x 36 m	24.98	360 x 360 m	24.41	0.57
Target size	200 cm	24.98	20 cm	25.24	0.26
Firing rate (w/ 1 hit)	4 (shots/sec)	24.98	40 (shots/sec)	24.84	0.14

10.10 Exploration Summary

This chapter uses the simulations from Chapter 7 to explore the space of first-person shooter games by varying the parameters values. Self-prediction and time warp can help player performance and when applied together, players perform as if there is no latency. Player performance is impacted by latency the more number of hits required for a kill. Firing rate, map size and target size do not have significant impact on the effects of latency on player performance. Higher skill players are more resistant to network latency. Comparison of the parameters shows that network latency, local latency, number of hits, firing rate (when number of hits is high) and latency compensation have the largest effects to the impact of latency on player performance and player skill has slightly lower effects. Firing rate (when number of hits is low), target size and map size have minimal effects.

Chapter 11

Conclusion

People are increasingly turning to games for entertainment evidenced by the growth of the game industry. Many multi-player computer games connect players via a network meaning network latency can delay player actions from input to viewing the corresponding output, this delay coming in addition to any local latency from the end system. While previous studies have assessed the impact of latency on computer games, in general, and first-person shooter games specifically, such studies often evaluate games with built-in latency compensation techniques, thus the results might not be accurate for other games. Moreover, theses studies has focused on a specific game task or a specific game with pre-set conditions. There has been no good answer as to how latency degrades performance for first-person shooter gamers across different games and game configurations (e.g., weapons, avatar size and map size), latency types, latency values and latency compensation techniques. Understanding the effects of latency on first-person selection can help inform game design and development techniques to mitigate latency's effects.

This thesis presents an approach for exploring first-person shooter scenarios with simulations instead of expensive user studies. We first study the "building blocks" of player actions in first-person shooter games: navigation - getting in position to shoot or avoid being shot; and selection - shoot at and hit the enemy avatar on the screen. For each of the actions, we conduct user studies with a custom game that isolated the action, analyze the data and build models. For navigation, we build models for the in-sight time window length where players can see the opponent and out-of-sight time window length where players cannot be seen by an opponent. For selection, we build models for elapsed time which is the time to select the target. We use these models to simulate first-person shooter games. We simulate first-person shooter game scenarios across different latency and game configurations. We then collect data from user studies for a custom first-person shooter game study and two commercial games to validate our simulations. Finally, we explore various first-person shooter scenarios with the validated simulations.

The main contributions of this thesis are summarized as below:

- Comparison of local latency and network latency. Without latency compensation, local latency and network latency have about the same effects on player performance. In commercial network first-person shooter games, network latency is often mitigated with built-in latency compensation whereas local latency is hard to mitigate with latency compensation.

- Understanding the impact of latency on the navigation action on first-person shooter game players. Analysis of the results from the navigation user study indicates local latency and network latency have similar effects on navigation in the absence of latency compensation. Both subjective quality of experience (QoE) and objective player performance degrade linearly with total latency, where 100 ms increase in latency results in about an 8 percent decrease in player score and a 20 percent decrease in QoE. The impact of latency depends upon the navigation goal, however, with latency hindering seeking for an opponent far more than hiding from an opponent. Both latency compensation techniques investigated – time warp and self-prediction – can improve player performance and QoE and, when applied together, can nearly completely overcome the effects of latency on performance and QoE.

- Models for in-sight and out-of-sight time windows. With the navigation data, we derive analytic models to describe the time intervals for generalizing navigation with latency and latency compensation using these models in our first-person shooter simulations.

- Understanding the impact of latency on the selection action on first-person shooter game players. Analysis of the results from the selection user study indicates both subjective quality of experience (QoE) and objective player performance degrade linearly with total latency. A 100 ms decrease in latency results in about 30 percent improvement in time to select (shoot a target) and a 12 percent decrease in QoE. Both time warp and self-prediction can improve player performance and QoE and, when applied together, can nearly completely overcome the effects of latency on performance and QoE.

- A model of elapsed time (time to select). With the selection data, we derive analytic models of the time to select that generalize target selection with latency and latency compensation.

- Validated simulations. We use the models for navigation and selection in simulations of first-person shooters scenarios. We validate our simulations with two studies of a commercial game and one with a custom first-person shooter game. Although there is bias between CS:GO data and simulated data due to difference in game mode and built-in compensation techniques, the trend of latency and performance holds. The validated simulation can provide rich explorations in first-person shooter scenarios.

- Explorations with the simulation. With the explorations, we find that player performance degrades with latency and that with 275 ms of latency, players playing against an opponent without latency may find it nearly impossible to win one-shot duels. For latency compensation techniques, both time warp and self-prediction can mitigate the effect of latency on player performance. When applied together, players can play as if there is no latency. Moreover, latency compensation, number of hits required to kill and player skill have large effect on player performance for players with latency, while target size, map size and firing rate have lower impact. Game designers can take use of the exploration findings to better design first-person shooter games. Latency values experienced by players may not be predictable at the game design phase but the game can be designed to better mitigate the effects of latency. For example, game designers can use creativity on target size, map size and firing rate since these parameters do not effect the impact of latency, but if total latency are expected to be significant, the game should use a lower number of hits required for a kill. Moreover, for any network latency, game designers should consider using time warp and self-prediction.

Chapter 12

Future work

Future work can further enrich our models and simulations. For example, we logged whether the opponent is in sight and if so, the fraction in sight. Since fraction changes target size in sight, we can add in sight fraction to our models the combine it with target size in the simulations. Moreover, we can include number and time length of deaths into our simulation and simulate games where players might respawn and reset. Future work can also do hierarchical modeling. This could start with models of individual users, combining them into certain groups users (e.g., based on skills/behaviors) before moving to aggregate, top-level models.

In-depth exploration can be conducted on why users may perform more poorly under latency. For example, in the hider/seeker actions, seeking users do worse (have shorter time windows) with latency, whereas hiders do not. Also, during a game, some actions might be more difficult for players to accomplish with latency. For example, a user study could run the Hide and Seek game again and study how far off player paths deviate with the optimal path under different amounts of latency.

Future work can validate our simulation with more first-person shooter elements. For example, other types of weapons, maps with different terrains, games with different game mode and team work. Other future work could explore how to accurately access player skill data without the need of collecting player performance data in a first-person shooter game. Other future work could explore how would preferred player settings (e.g. mouse sensitivity) improve player performance in first-person shooter games.

Moreover, future work could apply the same methodology used in our paper to gaming actions in other games genres, e.g., Multiplayer Online Battle Arena (MOBA) games like *DOTA 2* (Valve, 2013) and *League of Legends* (Riot Games, 2019), and Real-Time Strategy (RTS) games like *Starcraft* (Blizzard, 1998). In such a case, individual game actions would need to be isolated for the game and then evaluated, such as navigation for moving an avatar from a third-person perspective.

Bibliography

[AGC$^+$18] Maha Abdallah, Carsten Griwodz, Kuan-Ta Chen, Gwendal Simon, Pin-Chun Wang, and Cheng-Hsin Hsu. Delay-Sensitive Video Computing in the Cloud: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(3s), June 2018.

[AJG$^+$13] Rahul Amin, France Jackson, Juan E. Gilbert, Jim Martin, and Terry 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*, Berlin, Heidelberg, 2013. Springer-Verlag.

[AMC18] Axel Antoine, Sylvain Malacria, and Géry Casiez. Using High Frequency Accelerometer and Mouse to Compensate for End-to-End Latency in Indirect Interaction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, page 1–11, Montreal, QC, Canada, 2018. Association for Computing Machinery.

[Arm03] Grenville Armitage. An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3. In *Proceedings of IEEE ICON*, Sydney, Australia, September 2003.

[AZ97] Johnny Accot and Shumin Zhai. Beyond Fitts' Law: Models for Trajectory-based HCI Tasks. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*, pages 295–302, 1997.

[AZ99] Johnny Accot and Shumin Zhai. Performance Evaluation of Input Devices in Trajectory-Based Tasks: An Application of the Steering Law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI, page 466–472, Pittsburgh, Pennsylvania, USA, 1999. Association for Computing Machinery.

[AZ01] Johnny Accot and Shumin Zhai. Scale Effects in Steering Law Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI, page 1–8, Seattle, Washington, USA, 2001. Association for Computing Machinery.

[BCL$^+$04] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003. In *Proceedings of ACM NetGames*, Portland, OG, USA, September 2004.

[Ben] Human Benchmark. Human Benchmark - Reaction Time Statistics. `https://humanbenchmark.com/tests/reactiontime/statistics`. (Accessed on June 15, 2021).

[BK06] Shayne Burgess and Michael Katchabaw. Design and Implementation of Optimistic Constructs for Latency Masking in Online Video Games. In *The 2nd annual North American Game-On Conference (GameOn'NA). Monterey, CA, USA*. Citeseer, 2006.

[BM14] Justus Beyer and Sebastian Möller. Assessing the Impact of Game Type, Display Size and Network Delay on Mobile Gaming QoE. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 37:287 – 295, 2014.

[BS99] David A Balota and Daniel H Spieler. Word Frequency, Repetition, and Lexicality Effects in Word Recognition Tasks: Beyond Measures of Central Tendency. *Journal of Experimental Psychology: General*, 128(1):32, 1999.

[BSB06] Jeremy Brun, Farzad Safaei, and Paul Boustead. Managing Latency and Fairness in Networked Games. *Communications of the ACM*, 49(11):46–51, November 2006.

[CC06] Mark Claypool and Kajal Claypool. Latency and Player Actions in Online Games. *Communications of the ACM*, 49(11):40–45, 2006.

[CC07] Kajal Claypool and Mark Claypool. On Frame Rate and Player Performance in First Person Shooter Games. *Springer Multimedia Systems*, 13(1):3–17, 2007.

[CC10] Mark Claypool and Kajal Claypool. Latency Can Kill: Precision and Deadline in Online Games. In *Proceedings of the ACM MMSys*, Scottsdale, AZ, USA, February 2010.

[CCC$^+$07] Ling Chen, Gen-Cai Chen, Hong Chen, Jack March, Steve Benford, and Zhi-Geng Pan. An HCI Method to Improve the Human Performance Reduced by Local-Lag Mechanism. *Interacting with Computers*, 19(2):215–224, March 2007.

[CCD06] Mark Claypool, Kajal Claypool, and Feissal Damaa. The Effects of Frame Rate and Resolution on Users Playing First Person Shooter Games. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference (MMCN)*, volume 6071, San Jose, CA, USA, 01 2006. Association for Computing Machinery/Society of Photo-optical Instrumentation Engineers.

[CCG20] Mark Claypool, Andy Cockburn, and Carl Gutwin. The Impact of Motion and Delay on Selecting Game Targets with a Mouse. *ACM Transactions on Multimedia, Computing, Communication and Applications (TOMM)*, 16(2), May 2020.

[CER17] Mark Claypool, Ragnhild Eg, and Kjetil Raaen. Modeling User Performance for Moving Target Selection with a Delayed Mouse. In *Proceedings of Springer MMM*, Reykjavik, Iceland, January 2017.

[Cla18] Mark Claypool. Game Input with Delay - Moving Target Selection with a Game Controller Thumbstick. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) - Special Section on Delay-Sensitive Video Computing in the Cloud*, 14(3s), August 2018.

[DGB09] Matthew Dye, C Green, and Daphne Bavelier. Increasing Speed of Processing with Action Video Games. *Current Directions in Psychological Science*, 18(6):321–326, December 2009.

[DM17] A. Depping and R. Mandryk. Why is This Happening to Me?: How Player Attribution Can Broaden Our Understanding of Player Experience. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*, Denver, CO, USA, May 2017.

[Dru71] Colin G. Drury. Movements with Lateral Constraint. *Ergonomics*, 14 2:293–305, 1971.

[DWW05] Matthias Dick, Oliver Wellnitz, and Lars Wolf. Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games. In *Proceedings of ACM NetGames*, Hawthorn, NY, USA, 2005.

[DZC$^+$21] Wouter Durnez, Aleksandra Zheleva, Mark Claypool, Klaas Bombeke, Mathias Maes, Jan Van Looy, and Lieven De Marez. Spaz! The Effects of Local Latency on Player Actions in an Desktop-Based Exergame. *IEEE Transactions on Games*, pages 1–1, 2021.

[ERC18] Ragnhild Eg, Kjetil Raaen, and Mark Claypool. Playing with Delay: With Poor Timing Comes Poor Performance, and Experience Follows Suit. In *Proceedings of the 10th International Conference on Quality of Multimedia Experience (QoMEX)*, Sardinia, Italy, June 2018. IEEE.

[FKS16] Sebastian Friston, Per Karlström, and Anthony Steed. The Effects of Low Latency on Pointing and Steering Tasks. *IEEE Transactions on Visualization and Computer Graphics*, 22(5):1605–1615, 2016.

[FRS05] Tobias Fritsch, Hartmut Ritter, and Jochen Schiller. The Effect of Latency and Network Limitations on MMORPGs: a Field Study of Everquest 2. In *Proceedings of ACM NetGames*, Hawthorne, NY, USA, October 2005.

[gam20] gamersdecide.com. The Most Popular FPS Games in The World (2020). `https://www.gamersdecide.com/articles/most-popular-fps-games`, Apr 2020. (Accessed August 16, 2021).

[GE88] Sandra G.Hart and Lowell E.Staveland. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. *Human Mental Workload (Editors Peter A Hancock and Najmedin Meshkati)*, 52:139–183, 1988.

[gee21] geekygamingstuff.com. Is It Better to Play FPS Games with a Controller or Keyboard and Mouse? `https://geekygamingstuff.com/is-it-better-to-play-fps-games-with-a-controller-or-keyboard-and-mouse`, May 2021. (Accessed August 16, 2021).

[Goo20] Google. Google Stadia Post-game Survey. stadia, 2020. (Accessed Oct 15, 2021).

[HCW+14] Eben Howard, Clint Cooper, Mike Wittie, Steven Swinford, and Qing Yang. Cascading Impact of Lag on Quality of Experience in Cooperative Multiplayer Games. In *Proceedings of the 13th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, Nagoya, Japan, 2014. IEEE Press.

[HFPG16] Oliver Hohlfeld, Hannes Fiedler, Enric Pujol, and Dennis Guse. Insensitivity to Network Delay: Minecraft Gaming Experience of Casual Gamers. In *Proceedings of the International Teletraffic Congress (ITC)*, Würzburg, Germany, September 2016. IEEE.

[HHS21] David Halbhuber, Niels Henze, and Valentin Schwind. Increasing Player Performance and Game Experience in High Latency Systems. *Proc. ACM Hum.-Comput. Interact.*, 5(CHI PLAY), oct 2021.

[HKS+22] David Halbhuber, Annika Köhler, Markus Schmidbauer, Jannik Wiese, and Niels Henze. The Effects of Auditory Latency on Experienced First-Person Shooter Players. In *Proceedings of Mensch und Computer 2022*, pages 286–296. 2022.

[HLT20a] HLTV. CS:GO Statistics Database – Distribution of Maps Played. hltv.org, 2020. (Accessed September 17, 2020).

[HLT20b] HLTV. CS:GO Statistics Database – Top Weapons. hltv.org, 2020. (Accessed September 17, 2020).

[HPM91] Andrew Heathcote, Stephen J Popiel, and DJ Mewhort. Analysis of Response Time Distributions: An Example Using the Stroop Task. *Psychological bulletin*, 109(2):340, 1991.

[Hum] Human Benchmark. Reaction Time Test. (Accessed April 13, 2021).

[IdKP13] W. Ijsselsteijn, Y. de Kort, and K. Poels. The Game Experience Questionnaire. Technical report, Technische Universiteit Eindhoven, 2013.

[ISGS15] 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 ACM SIGCHI Conference on Human factors in Computing Systems*, Seoul, Republic of Korea, April 2015.

[Jef85] David R. Jefferson. Virtual Time. *ACM Transsactions on Programming Language and Systems*, 7(3):404–425, July 1985.

[JSB05] 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 IEEE 7th Malaysia International Conference on Communications*, volume 1, pages 6 pp.–, 2005.

[JT07] Jeroen Jansz and Martin Tanis. Appeal of Playing Online First Person Shooter Games. *Cyberpsychology & Behavior*, 10(1):133–136, 2007.

[JT14] Benjamin F. Janzen and Robert J. Teather. Is 60 FPS Better than 30? The Impact of Frame Rate and Latency on Moving Target Selection. In *Extended Abstracts on Human Factors in Computing Systems*, CHI EA, page 1477–1482, Toronto, Ontario, Canada, 2014. Association for Computing Machinery.

[KGS07] Raghavendra S. Kattinakere, Tovi Grossman, and Sriram Subramanian. *Modeling Steering within Above-the-Surface Interaction Layers*, page 317–326. Association for Computing Machinery, San Jose, California, USA, 2007.

[Kos08] Robert J Kosinski. A Literature Review on Reaction Time. *Clemson University*, 10(1), 2008.

[LC15] Wai-Kiu Lee and Rocky K. C. Chang. Evaluation of Lag-Related Configurations in First-Person Shooter Games. In *International Workshop on Network and Systems Support for Games (NetGames)*, pages 1–3, 2015.

[LC17] Steven Lee and Rocky Chang. On 'Shot Around a Corner' in First-Person Shooter Games. In *Proceedings of the 15th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, Taipei, Taiwan, June 2017. IEEE.

[LC18] Steven W. K. Lee and Rocky K. C. Chang. Enhancing the Experience of Multiplayer Shooter Games via Advanced Lag Compensation. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys, page 284–293, Amsterdam, Netherlands, 2018. Association for Computing Machinery.

[LC21a] Shengmei Liu and Mark Claypool. EvLag: A Tool for Monitoring and Lagging Linux Input Devices. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 281–286, 06 2021.

[LC21b] Shengmei Liu and Mark Claypool. Game Input with Delay - A Model of the Time Distribution for Selecting a Moving Target with a Mouse. In *Proceedings of Springer MMM*, Prague, Czech Republic, June 22-24 2021.

[LCD$^+$20] Shengmei Liu, Mark Claypool, Bhuvana Devigere, Atsuo Kuwahara, and Jamie Sherman. 'Git Gud!' - Evaluation of Self-Rated Player Skill Compared to Actual Player Performance. In *Proceedings of ACM CHI Play*, Online (Virtual Conference), November 2020.

[Lee17] Nick Lee. Beyond 50/50: Breaking Down the Percentage of Female Gamers by Genre. Online: `https://quanticfoundry.com/2017/01/19/female-gamers-by-genre/`, Jan 2017. (Accessed September 5, 2021).

[LG18] Michael Long and Carl Gutwin. Characterizing and Modeling the Effects of Local Latency on Game Performance and Experience. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*, Montréal, Canada, 2018.

[LG19] Michael Long and Carl Gutwin. Effects of Local Latency on Game Pointing Devices and Game Pointing Tasks. In *Proceedings of the ACM*

SIGCHI Conference on Human factors in Computing Systems, Glasgow, Scotland, UK, 2019.

[LKS$^+$21a] Shengmei Liu, Atsuo Kuwahara, James Scovell, Jamie Sherman, and Mark Claypool. Comparing the Effects of Network Latency versus Local Latency on Competitive First Person Shooter Game Players. In *EHPHCI' 21*, Yokohama, Japan, Apr 2021. OSF Preprints.

[LKS$^+$21b] Shengmei Liu, Atsuo Kuwahara, James Scovell, Jamie Sherman, and Mark Claypool. Lower is Better? The Effects of Local Latencies on Competitive First-Person Shooter Game Players. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*, Yokohama, Japan, 2021.

[LSGH17] Huy Viet Le, Valentin Schwind, Philipp Göttlich, and Niels Henze. PredicTouch: A System to Reduce Touchscreen Latency Using Neural Networks and Inertial Measurement Units. In *Proceedings of the ACM International Conference on Interactive Surfaces and Spaces*, ISS '17, page 230–239, Brighton, United Kingdom, 2017. Association for Computing Machinery.

[Luc86] Duncan Luce. *Response Times: Their Role in Inferring Elementary Mental Organization*. Number 8. Oxford University Press on Demand, 1986.

[LXC21] Shengmei Liu, Xiaokun Xu, and Mark Claypool. Online Games and Network Latency Compensation Techniques. Technical Report WPI-CS-TR-20-01, Computer Science Department at Worcester Polytechnic Institute, May 2021.

[LXC22] Shengmei Liu, Xiaokun Xu, and Mark Claypool. A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games. *ACM Computing Surveys*, 1(1), February 2022.

[Mau00a] Martin Mauve. Consistency in Replicated Continuous Interactive Media. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, page 181–190, Philadelphia, PA, USA, 2000. Association for Computing Machinery.

[Mau00b] Martin Mauve. How to Keep a Dead Man from Shooting. In *Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[Mur21] Trent Murray. Top 10 Esports Games of 2020 by Total Winnings – The Esports Observer. `https://esportsobserver.com/top10-games-2020-total-winnings/`, Jan 2021. (Accessed September 2, 2021).

[MVHE04] 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.

[opt20] optimum.com. What Is Latency? Online: `https://www.optimum.com/internet/what-latency`, feb 2020. (Accessed Nov 15, 2021).

[PdI07] K. Poels, Y.A.W. de Kort, and W.A. IJsselsteijn. *D3.3 : Game Experience Questionnaire: Development of a Self-report Measure to Assess the Psychological Impact of Digital Games*. Technische Universiteit Eindhoven, 2007.

[PW02] Lothar Pantel and Lars C. Wolf. On the Impact of Delay on Real-Time Multiplayer Games. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 23–29, Miami, FL, USA, May 2002. Association for Computing Machinery.

[QBV$^+$13] Peter Quax, Anastasiia Beznosyk, Wouter Vanmontfort, Robin Marx, and Wim Lamotte. An Evaluation of the Impact of Game Genre on User Experience in Cloud Gaming. In *IEEE International Games Innovation Conference (IGIC)*, pages 216–221, 2013.

[QML$^+$04] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game. In *Proceedings of ACM NetGames*, Portland, OG, USA, 2004.

[rai21] raiseyourskillz.com. Why Are First Person Shooters So Popular? `https://raiseyourskillz.com/why-are-first-person-shooters-so-popular-11-reasons/`, jun 2021. (Accessed August 16, 2021).

[Ric14] Benjamin Richardson. Reaction Time Differences in Video Game and Non-Video Game Players. `https://digitalcommons.cwu.edu/cgi/viewcontent.cgi?article=1689&context=source`, 2014. (Accessed on 07/15/2021).

[RRP06] Richard Ryan, Scott Rigby, and Andrew Przybylski. The Motivational Pull of Video Games: A Self-determination Theory Approach. *Motivation and Emotion*, 30(4):344–360, 2006.

[SBB+19] 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 Technical Briefs*, pages 110–113. ACM, Brisbane, QLD, Australia, 2019.

[SC19] Jiawei Sun and Mark Claypool. Evaluating Streaming and Latency Compensation in a Cloud-based Game. In *Proceedings of the 15th IARIA Advanced International Conference on Telecommunications (AICT)*, Nice, France, June 2019.

[SG13] Cheryl Savery and T. C. Graham. Timelines: Simplifying the Programming of Lag Compensation for the Next Generation of Networked Games. *Multimedia Systems*, 19(3):271–287, June 2013.

[SGG10] Cheryl Savery, T. C. Nicholas Graham, and Carl Gutwin. The Human Factors of Consistency Maintenance in Multiplayer Computer Games. In *Proceedings of the 16th ACM International Conference on Supporting Group Work*, page 187–196, Sanibel Island, FL, USA, 2010. Association for Computing Machinery.

[Sha21] Steven Shaw. Valorant vs CSGO Player Count: Which FPS Game Has More Players in 2021? `https://stealthoptional.com/how-to/`

`valorant-csgo-player-count-which-fps-has-more-players-2021/`, july 2021. (Accessed August 16, 2021).

[SK05] Gabriel Shelley and Michael Katchabaw. Patterns of Optimism for Reducing the Effects of Latency in Networked Multiplayer Games. In *Proceedings of FuturePlay*, East Lansing, Michigan, USA, 2005.

[SKH02] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. Aspects of Networking in Multiplayer Computer Games. *The Electronic Library*, 20(2):87–97, 2002.

[SSSZ$^+$18] 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, Taichung, Taiwan, 2018.

[SSZ$^+$20] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Carsten Griwodz, and Sebastian Möller. Delay Sensitivity Classification of Cloud Gaming Content. In *Proceedings of the 12th ACM Workshop on Immersive Mixed and Virtual Environment Systems (MMVE)*, pages 25–30, Istanbul, Turkey, 2020. Association for Computing Machinery.

[Sta20] Statista. Distribution of Gamers Playing Selected Game Genres Worldwide as of January 2017, by Gender. Online: `https://tinyurl.com/yytrbj4d`, 2020. (Accessed September 17, 2020).

[SZM17] Steven Schmidt, Saman Zadtootaghaj, and Sebastian Möller. Towards the Delay Sensitivity of Games: There is More than Genres. In *Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2017.

[TAS07] Alexey Tumanov, Robert Allison, and Wolfgang Stuerzlinger. Variability-aware Latency Amelioration in Distributed Environments. pages 123–130, 01 2007.

[TG05] Thomas L Thornton and David L Gilden. Provenance of Correlations in Psychological Data. *Psychonomic Bulletin & Review*, 12(3):409–441, 2005.

[u/k19] u/khaniage. Counter-strike: Global Offensive – Map Sizes. Reddit, 2019. (Accessed September 17, 2020).

[wep22] wepc.com. Video Game Industry Statistics, Trends and Data In 2022. Online: `https://www.wepc.com/news/video-game-statistics/`, Jan 2022. (Accessed October 26, 2022).

[Whe08] Robert Whelan. Effective Analysis of Reaction Time Data. *The Psychological Record*, 58(3):475–482, 2008.

[WO00] Jiann-Rong Wu and Ming Ouhyoung. On Latency Compensation and Its Effects for Head Motion Trajectories in Virtual Environments. *The Visual Computer*, 16:79–90, 03 2000.

[ZAW04] Shumin Zhai, Johnny Accot, and Rogier Woltjer. Human Action Laws in Electronic Virtual Worlds: An Empirical Study of Path Steering Performance in VR. *Presence*, 13:113–127, 04 2004.