Exploring the Effects of Network Latency Compensation Techniques on Player Performance and Experience in FPS Games

A Major Qualifying Project
submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the degree of Bachelor of Science

by
Ivan Klevanski
Alexander Mitchell
Yihong Xu
Sitsanok Young

27 April 2023

Report submitted to:
Ben Boudaoud, Josef Spjut, Joohwan Kim
Sponsor: NVIDIA Corporation

Advisor: Professor Mark Claypool
Worcester Polytechnic Institute
Abstract

Network latency can significantly impact user experience in First-Person Shooter (FPS) games, which can cause the game to feel unresponsive and sluggish. To combat network latency issues, various latency compensation techniques have been developed. Despite the abundance of these techniques and their implementations in popular FPS games, their performance and impact on mitigating network latency have not been scientifically investigated. For our project, we implemented several latency compensation techniques and evaluated the impact of time warp and latency exposure on player performance and quality of experience in an open-source FPS game. Our results show that time warp is able to significantly reduce the effects of latency on player performance and quality of experience, while the effectiveness of latency exposure varies with the actual latency values.
Acknowledgements

We would like to express our sincere gratitude to everyone who supported us throughout this project. Firstly, we would like to extend our heartfelt thanks to Professor Mark Claypool of the WPI Computer Science Department for his advice, guidance, and unwavering support as we worked on the project. We would also like to thank the team at NVIDIA, including Ben Boudaoud, Josef Spjut, and Joohwan Kim, for developing FPSci, sharing their expertise, and providing guidance.

In addition, we would like to express our appreciation to our fellow WPI students who were working on FPSci, including Samin Shahriar Tokey, Alexander Hayden, and Benjamin Peters, for their ideas, code, and support as we worked together. Mattheus Faria, Miles Gregg, Jonathan Hsu, and Pari Nguyen also deserve a special mention for working alongside us. Moreover, we would like to thank Wesley Lo, Shiyu Wu, and Haojun Yan for collaborating with us on their data-driven music project and providing a video for our project presentation.

Finally, we are deeply grateful to all of the participants in our user studies. Without them, this project would not have been possible. Thank you all for your support and contributions.
Table of Contents

Abstract.................................................................................................................................................. i
Acknowledgements............................................................................................................................... ii
Table of Contents.................................................................................................................................. iii
List of Figures.......................................................................................................................................... v
List of Tables............................................................................................................................................ vi
1. Introduction.......................................................................................................................................... 1
2. Background.......................................................................................................................................... 3
   2.1 FPSci............................................................................................................................................... 3
   2.2 Authoritative Server....................................................................................................................... 4
   2.3 Network Latency............................................................................................................................ 5
   2.4 Latency Compensation Techniques............................................................................................. 7
      2.4.1 Extrapolation........................................................................................................................... 7
      2.4.2 Interpolation............................................................................................................................ 8
      2.4.3 Time Warp.............................................................................................................................. 9
      2.4.4 Latency Exposure.................................................................................................................... 10
      2.4.5 Latency Concealment............................................................................................................ 10
3. Implementation...................................................................................................................................... 12
   3.1: Prerequisite Implementations....................................................................................................... 13
      3.1.1: Authoritative Server.............................................................................................................. 14
         3.1.1.1: Initial Proposal................................................................................................................. 14
         3.1.1.2: Current Structure............................................................................................................ 18
         3.1.1.3: Positioning....................................................................................................................... 19
         3.1.1.4: Shooting.......................................................................................................................... 21
      3.1.2: Data Handler.......................................................................................................................... 23
         3.1.2.1: Server.............................................................................................................................. 23
         3.1.2.1: Client.............................................................................................................................. 24
   3.2: Latency Compensation Implementations.................................................................................... 25
      3.2.1: Time Warp............................................................................................................................ 25
      3.2.2: Extrapolation....................................................................................................................... 27
      3.2.3: Latency Exposure................................................................................................................ 30
      3.2.4: Latency Concealment......................................................................................................... 34
   3.3: Auxiliary Implementations.......................................................................................................... 34
      3.3.1: Shot-Around-The-Corner Detection..................................................................................... 34
      3.3.2: Logging.................................................................................................................................. 36
4. Evaluation............................................................................................................................................. 38
4.1: Methods...................................................................................................................................................... 38
  4.1.1: Recruitment........................................................................................................................................... 38
    4.1.1.1: Interest Form................................................................................................................................. 39
    4.1.1.2: Scheduling..................................................................................................................................... 40
  4.1.2: Protocols.............................................................................................................................................. 41
    4.1.2.1: Pre-Session Protocols.................................................................................................................. 41
    4.1.2.2: Current Session Protocols.......................................................................................................... 42
    4.1.2.3: Post-Session Protocols............................................................................................................... 43
    4.1.2.4: Additional Policies...................................................................................................................... 44
    4.1.2.5: Database Flags............................................................................................................................ 46
  4.1.3: User Study......................................................................................................................................... 46
    4.1.3.1: Description....................................................................................................................................... 46
    4.1.3.2: Data Collection............................................................................................................................. 48
  4.2: Results.................................................................................................................................................... 48
    4.2.1 Demographics...................................................................................................................................... 48
    4.2.3: Ping Display....................................................................................................................................... 51
    4.2.4: Time Warp......................................................................................................................................... 53
    4.2.5: Summary........................................................................................................................................... 59

5. Conclusion............................................................................................................................................... 61

6. Future Work............................................................................................................................................. 62

References.................................................................................................................................................... 64

Appendices.................................................................................................................................................... 67
  Appendix A: Interest Form............................................................................................................................. 67
  Appendix B: Session Form............................................................................................................................. 70
  Appendix C: Table of Authorship............................................................................................................... 79
  Appendix D: Technical Achievements........................................................................................................ 80
  Appendix E: Non-Technical Achievements................................................................................................. 81
List of Figures

1 Initial Dependency Graph........................................................................................................13
2 Initial Proposal: Client-Server Structure...................................................................................15
3 Initial Proposal: Two New Classes and Their "Binding"..........................................................17
4 Network Structure for Positioning and Responsibilities of Each Party..........................................19
5 Networked Shooting with Time Warp off (Miss Case)..............................................................22
6 Networked Shooting with “full Time Warp” on (Hit Case).........................................................26
7 Linear Extrapolation....................................................................................................................28
8 Quadratic Extrapolation 1............................................................................................................29
9 Quadratic Extrapolation 2............................................................................................................30
10 Quadratic Extrapolation 3..........................................................................................................30
11 Graphic Showing the Individual Program Threads for Latency Exposure..................................31
12 Screenshot of the Ping Indicator...............................................................................................34
13 Illustration of the Visibility Tester Algorithm...........................................................................35
14 Participant Ratings of FPS Experience with Keyboard and Mouse...........................................49
15 When Participants Last Played FPS Games with Keyboard and Mouse......................................49
16 Reaction Test Data......................................................................................................................50
17 How Ping Display Affects Quality of Experience.......................................................................51
18 Accuracy of Ping Display............................................................................................................52
19 Accuracy with Time Warp Off/On..............................................................................................54
20 Amount of Kills and Experience Per Round.............................................................................56
21 Amount of Kills and Experience Per Round Time Warp Off....................................................57
22 Amount of Kills and Experience Per Round Time Warp On.....................................................58
List of Tables

1 Authorities for Each Major Multiplayer Functionality................................................................. 18
2 Ping display accuracy ANOVA test................................................................................................. 53
3 Accuracy with Time Warp Off/On................................................................................................. 53
4 Accuracy with Time Warp Off/On Pairwise T-Test...................................................................... 55
6 Shot-Around-The-Corner............................................................................................................... 59
7 Partially Hidden........................................................................................................................... 59
1. Introduction

First-person shooter (FPS) games are one of the most popular genres of video games, with millions of players competing in multiplayer matches every day [1], [14], [15]. Network latency refers to the time it takes information to get from a host to another host over a computer network. Although the time delay may seem negligible to human perception, it can severely affect user experience in online games. Studies indicate that network latency can render the game unresponsive and sluggish, thus compromising the quality of gameplay [4], [5], [6].

Attempting to directly reduce network latency as a whole would involve massive hardware and software changes; as such, mitigating the effect of network latency solely in the games themselves would be more desirable, doable, and cost effective. To mitigate these unwanted effects, various latency compensation techniques have been developed. The purpose of these techniques is to not remove the latency but to improve game logic or trick the player so the impact of network latency is reduced. However, even though the compensation methods are popular in the industry, there has been limited public research and testing in this field.

For our project, we implemented various latency compensation techniques and evaluated the impact of time warp and latency exposure on player performance and quality of experience (QoE) in the experiment-centric FPS game First Person Science (FPSci), developed by NVIDIA for research purposes. Our team extended the game's existing client-server multiplayer networking and implemented the compensation techniques on top of the game's new network capabilities. We then conducted user studies to measure the effects of these techniques on player performance and QoE.
During our user study, we conducted game rounds with participants playing with and without latency compensation techniques enabled to establish a baseline value for each player's performance and evaluate the effectiveness of these techniques. The two primary techniques we tested were time warp and latency exposure. Through data analysis, we discovered a statistical significance between time warp and player performance, as well as between latency exposure and player QoE.

In this report, we aim to contribute to the understanding of how network latency compensation techniques affect player performance and QoE in FPS games and to provide insights that can aid in the development of more effective compensation techniques in the future. We first provide a comprehensive overview of FPSci, authoritative servers, network latency, and latency compensation techniques in Chapter 2. Following that, we discuss our extensions to FPSci in Chapter 3 and present our methods and results for the user study we conducted in Chapter 4. Finally, we conclude with our findings in Chapter 5 and outline future research directions in Chapter 6.
2. Background

This chapter begins by presenting the FirstPersonScience project, an open-source initiative, and explains the concept of an authoritative server and its significance. It then explains network latency and how it affects player performance and experience. Afterwards, it describes various latency compensation techniques such as extrapolation, interpolation, time warp, latency exposure, and latency concealment.

2.1 FPSci

FPSci is an open-source project developed by NVIDIA and designed to study a broad set of user interactions in FPS games [2]. FPSci achieves this by having multiple operational modes and adjustable values to simulate different game conditions, which reflect different scenarios that FPS players could face while gaming. Such configurations also allow per-session configuration files that can be modified to the desired base environment for a particular test. The frame rate and latency are also independently controlled to improve the testing of different features and techniques to optimize the efficiency of both.

The main limitation of FPSci is that it is a single player experience that only lets the player shoot floating target plates in front of them. While there are FPS games that are single player only, the most popular ones are almost always multiplayer [1]. To maximize the testing capabilities of FPS games, it would be preferable for FPSci to support and simulate multiple players playing at the same time.

The previous Major Qualifying Project (MQP) team to work on FPSci created a dedicated server application to manage communication between multiple players on
separate machines playing in the same game [13]; for instance, as player one moves across their local world state the server echoes those movements to all the other clients so they can also update player one’s location in their world state. If player one’s client determined that it shot and killed player two, the server tells all the connected players that player two has been shot and has died/needs to respawn. This is known as client authoritative control because the clients themselves determine what events happened and how they happened. The server simply echoes the packets that each client sends regarding its movement and information on whether or not the client player is shooting at another player to all connected clients; this provides a multiplayer experience and allows FPSci to begin testing multiplayer studies. In order to properly simulate a networked FPS game where logical game attributes, such as player locations and player shot information, are not decentralized among clients, we extended FPSci to run an authoritative server.

2.2 Authoritative Server

The previous MQP team’s changes to FPSci allowed for user testing with multiple players and created an experience similar to that of major online FPS games [13], but the implementation did not address certain issues that arise from enabling networking in such games. One of the primary issues includes not having a central hub that stores the “true value” of the world state, which can lead to conflicts that occur when two players move to the same position simultaneously due to network latency. Another issue is desynchronization between clients caused by packet loss, where a player informs others of an event like moving to another location or shooting another player, but the packet is lost on its way and never reaches some of the players. This can be solved by either using special
User Datagram Protocol (UDP) packets with reliable data transfer mechanisms that require delivery confirmation or a central point of contact like an Authoritative Server that stores the “true values” of the entire game world. By implementing a centralized structure such as the Authoritative Server, issues such as cheating can be mitigated. An example of cheating can include the falsification of the successful shooting of another player; in this case, the shooting player would only need to alter the results on their local client and claim that they shot the other player without proof. Cheating would be much more difficult if an Authoritative Server determines whether or not a player was shot since the cheater would not be able to just state that they shot another player as the hit resolution would be entirely managed by the server itself and not the clients. Despite this, given that the project was primarily designed to be an application with a focus on solely testing different FPS environments and events, cheating is considered to be of low priority and thus would not be an issue we try to avoid.

Apart from resolving the aforementioned issues that came with enabling multiplayer games, the Authoritative Server model also enables us to apply more advanced optimizations regarding latency, packet reordering, low server response rates, and other features.

### 2.3 Network Latency

Network latency, also known as “ping,” is a measure of the delay in communication between a client and a network server. It refers to the round-trip time it takes for a client to send a data packet to the server and receive a response [6], [7]. Network latency plays a crucial role in determining the overall performance of a network and is particularly
important in multiplayer online gaming, where even small delays can negatively impact the user's performance and QoE [4], [5], [6].

High network latency can result in several impacts on the user’s experience, such as delayed hit confirmations and delayed interactions. Delayed hit confirmations occur when a player's shot successfully hits an opponent, but the kill is not confirmed until later due to the delay in communication between the client and server. Delayed interactions occur when a player tries to perform an action, such as opening a door, but the visual response lags behind the actual mouse click [6].

There are several factors that contribute to network latency, including the transmission medium, distance, number of network hops, data volume, and server performance. The transmission medium refers to the physical path that data travels, such as a wired or wireless connection. Distance refers to the geographical location of the client and server and the number of network hops refers to the number of intermediate devices the data must pass through before reaching its destination. The data volume, or the amount of data being transmitted, and server performance also play a role in determining network latency [7].

It is important to differentiate network latency from local latency, which focuses on the time it takes for a user input to be displayed on the monitor. Network latency specifically refers to the delay in communication between the client and server. Addressing network latency and its impact on the user experience is vital for ensuring positive multiplayer gaming experiences [6].
2.4 Latency Compensation Techniques

To evaluate and mitigate the effects of network latency, several mechanisms for latency compensation have been proposed, developed, and refined to improve the end-user’s experience when playing online games. In games that follow the client and server network architecture, options for latency compensations exist on both client-side and on server-side environments. Examples of such latency compensation techniques include client-side extrapolation and interpolation as well as server-side time warp, alongside client-side latency exposure and concealment serving as visual latency quantifiers and indicators. The aforementioned techniques for combating latency have been implemented and used, most often in conjunction with one another, in a variety of different online FPS games including but not limited to: Quake 3, Halo, Overwatch, Valorant, as well as other examples [8].

2.4.1 Extrapolation

In client-side latency compensations, the client utilizes incoming server-side world state data to visually hide the presence of network latency for one individual client player as opposed to for all players existing in the game. The first of such latency compensation techniques is extrapolation in which the client predicts future world states, i.e. other players’ positions and movement directions, based on the currently observed behaviors from the present world state data that was sent by the server to the client [8]. The client takes data from a set of past world states and performs a series of operations that creates an inference of what the next world state would look like in the case where the server has a delay in transmitting the anticipated world state; once the client receives the actual world state, the client perceives a seamless experience by extrapolating the future positions of other players based on past observed behaviors.
state from the server, it can make minor corrections or adjustments to match the actual world state.

A common technique for implementing extrapolation is known as “dead-reckoning” in which the server-side world state data used for extrapolation contains a special DR (Dead Reckoning) vector that contains information on a player or object’s current cartesian position as well as trajectory information such as the object’s current velocity and acceleration. Using an implementation of the dead-reckoning technique for extrapolation, a team of researchers have found that the latency compensation technique has resulted in improvements in hit accuracy amongst players anywhere from 100 ms to 800 ms in a series of experiments that were conducted on the game BZFlag [10].

2.4.2 Interpolation

The second of the client-side latency compensation techniques is interpolation, which similarly to extrapolation predicts world-states based on data delivered from the server. However, instead of predicting future frames, interpolation calculates additional frames in between two already existing frames. This compensation technique is most commonly used to smooth out the movement of objects on the client if the client’s visual update rate exceeds the rate at which the server sends out world frames [8]. In a simple example, the client may take data from two consecutive frames such as a player’s position, then find a difference or delta between the two values after which it can then multiply some incrementing time variable from 0 to 1 to the difference and add it to the earlier of the world state data values to simulate a world state where the object’s data values are in-between the two initial world states by some time variable.
A more advanced technique for achieving interpolation involves the usage of local perception filters, which create temporal distortions for passive, non-player entities based on their distance away from the local player as well as other active remote player entities based on the remote player’s network delay [12]. Such a system enables all players to accurately perceive different dynamically moving objects in the scene regardless of latency, which can be useful in scenarios where players are able to accurately note the relative positions of slower moving ranged projectiles during first person shooter matches.

2.4.3 Time Warp

Unlike client-side latency compensations, server-side compensations do not visually alter the game to mitigate the effects of network latency; instead, such compensations usually do alterations to the world state in order to account for the different network delays of each player in the game. One of such server-side latency compensations is time warp, which performs a rollback in the server’s world state or virtual time during incoming server-bound information events from clients such as hit registration to accommodate for larger network delays [8]. Such a compensation technique allows a player in a first person shooter experiencing network latency to still be able to register hits based on his or her current latent world state perspective and have the server still count the shots as valid despite the less latent player already seeming to be out of sight of the shooter. While this may seem fair for the latent player that initiated the shot, less latent players that were shot may experience frustration as he or she may have already gone into cover; this represents the ‘shot-around-the-corner’ problem and would involve a careful consideration of the
players’ RTT values as well as their movement speeds to make such situations as fair for both sides as possible [11].

### 2.4.4 Latency Exposure

Besides being used in time warp, information on the RTT is also an integral dataset in latency exposure. Unlike regular latency compensation techniques, latency exposure does not perform any visual alterations in the game’s environment nor does it perform any behind-the-scenes adjustments on the server; instead, the goals of latency exposure are to simply provide the player with feedback on the magnitude of latency that exists between the client and server [8]. A visual quantifier of latency may consist of a numerical value for ping displayed on-screen that is updated real-time and can also consist of logged RTT data that can then be graphed to compare the fluctuations of the data over time. For online FPS games with multiple servers for the player to connect to, a player is unlikely to join a server with higher network latency than one with lower latency [9], which can be conveyed via a latency exposure mechanism and thus reduce the average latency across all players.

### 2.4.5 Latency Concealment

Alongside latency exposure as the second latency feedback provider, latency concealment acts, as the name suggests, as a mechanism for hiding the prevalence of latency from the player on the client. Such concealments may include animations or particle effects that fill in the delay caused by packet RTT before performing the actual task in question, which can include object / player movement or weapon fire registrations in the case of first person shooters [8]. Besides simple latency masking animations, other
advanced forms of latency concealment include late warps, which are most commonly used in virtual reality headsets to improve immersion and prevent the loss of balance as well as sicknesses during playtime.
3. Implementation

The following section describes our implementations used to assess the impact of latency compensations on player performance and QoE. The generalized list of additions made to FPSci to address this impact are in three primary categories: prerequisite implementations, latency compensation implementations, and auxiliary implementations. Prerequisite implementations include game changes that our team applied in order to implement a subset of the planned latency compensation techniques, primarily pertaining to time warp. This category includes the server-authoritative network model and specific data handling mechanisms. The latency compensations category lays out the specific implementation details of each latency compensation technique that was added to FPSci, those being time warp, extrapolation, latency exposure, and latency concealment. Lastly, auxiliary implementations include miscellaneous additions that did not fit into the first two categories but are nevertheless integral to the impact assertion of latency compensations. Such additions included an algorithm for detecting shots-around-corners as well as logging additional entries on the server and client databases.
3.1: Prerequisite Implementations

To implement many of the latency mitigation techniques, we implement Authoritative Server. Figure 1 is an initial graph for feature dependencies, building from the previous MQP’s network infrastructure with a broadcasting server.

Figure 1: Initial Dependency Graph
3.1.1: Authoritative Server

What we refer to as Authoritative Server ("AS") is the core of our implementation for the Server-Authoritative Client-Server structure. It consists of server-side management of clients and client data as well as the enforcement of server commands towards the clients. AS handles the two major events to be synced over the network in FPSci: movement and shooting; in addition, the server also handles some other aspects of the multiplayer game like round-switching and player config switching for user testing, but this is not part of AS we define here.

3.1.1.1: Initial Proposal

After reviewing the existing codebase from both the previous MQP and NVIDIA, we made a proposal of how we plan to implement Authoritative Server. This plan was later modified in tandem with the team’s deeper understanding of the codebase that we obtained while trying to implement AS.

As shown in Figure 2, we planned to allow each client to render what was locally predicted immediately instead of waiting one Round-Trip Time for the server to send an authoritative confirmation. The server would receive the user input, perform the physics simulation locally, and broadcast the authoritative world state results to each client.
The network loop of our initial proposal can be summarized as: 1. The client sends a UDP packet containing serialized user inputs every frame to the server, 2. The server receives the player input packet and deserializes the contents to be allocated in memory for server-side simulations, 3. The server-side simulation is performed, producing an authoritative world state for the current frame and cast shots, 4. The server sends back packets with the serialized world states back to each of the clients, 5. The clients receive server-side world state packets and update their old values with the new server-authoritative ones.

Following this early proposal, the first challenge we foresaw in implementing the feature was to introduce the presence of online players to both the server and the clients. In
the original single-player FPSci, the player was able to shoot floating disks against a wall with their weapon. Each disk was an instance of class “TargetEntity”, which is a subclass of the G3D “VisibleEntity” class whilst the player was an instance of the class “PlayerEntity”, which is another subclass of the G3D “VisibleEntity” class. This, along with how single-player FPSci handled weapon simulation, introduced a number of issues for us during the Authoritative Server implementation since our goal was to attempt to remain mergeable with the original codebase and avoid major rewrites to existing code. The specific issues we encountered included the following:

Issue 1: Player code to simulate movement and collision only existed in the PlayerEntity class, which meant that we had to use this class for movement and collision simulation on the server.

Issue 2: Weapon code to simulate weapon firing only acted on the TargetEntity class and there was no simple solution to modify it to support hitting PlayerEntities. The TargetEntity class as such also had to be used for weapon simulation on the server.
A new diagram (Figure 3) was introduced as part of our initial proposal to resolve this issue. We planned to use a NetworkedPlayer PlayerEntity for the server to simulate movement and collision, and a PlayerHitbox TargetEntity for weapon simulation on the server. The two entities were to be bound together so that whenever a NetworkedPlayer moves, the corresponding PlayerHitbox moves with it, meanwhile when a PlayerHitbox was shot, the NetworkedPlayer would be shot.
3.1.1.2: Current Structure

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Movement</td>
</tr>
<tr>
<td>Location</td>
<td>Server</td>
</tr>
<tr>
<td>Shooting</td>
<td>Server</td>
</tr>
</tbody>
</table>

Table 1: Authorities for Each Major Multiplayer Functionality

During implementation of the initially proposed structure, we found out that multiple integral parts of the game loop expected there to be only one PlayerEntity in the entire game to represent the current player controlling the game. Due to the complexity of migrating movement calculation to the server without being able to use multiple PlayerEntities in one game, we eventually decided to allow each client to perform local movement calculation to send to the server and only use a “NetworkedEntity” TargetEntity subclass to represent each online player. Since the world was immutable and cheat prevention was not a goal for this project, the player would always be able to calculate a correct location to move to and as such this decision would not impact the integrity of the project. Compared to solely having the server determine player movement, this structure has the added benefit of having an extremely low latency (≈ local latency) for the rendering of user input. The player would feel like the movement is instantaneous instead of being delayed as a result of having to wait for the server to respond.
3.1.1.3: Positioning

As mentioned previously, player movement is calculated by each client that is then sent to the server for validation and broadcasting. Figure 4 describes the role of each party in the network structure and how they interact with each other.

![Network Structure for Positioning and Responsibilities of Each Party](image)

**Figure 4: Network Structure for Positioning and Responsibilities of Each Party**

**Client-Authoritative Movement Calculation**

In our version of the game, player movement calculations have not been modified from the original FPSci game. Each client calculates movement of the next frame with world collision in mind, by sliding the player along a path and calculating when or if it would collide with the world. Despite this, collisions between players are not handled by this calculation and are instead authoritatively managed by the server in the form of movement validation checks.

**Server-Authoritative Movement Validation**

The authoritative server, upon receiving information about a client’s movement, performs validation checks to ensure that the client’s requests are normal. Currently, this check only confirms whether a player has moved too far compared to the last frame or if two players collided with each other. Movement speed checking is possible thanks to our
data handling infrastructure that stores a history of authoritative location information for each client. In the event that a past frame is not available, an earlier frame’s data is used up until an externally configurable limit, while the speed calculation adjusts for the extra time in between extra frames involved. Player-to-player collision detection was done by checking if the distance between two players is greater than the diameter of the player’s sphere model to save CPU time.

After a successful validation of speed and collision, the server stores this frame’s information and broadcasts it along with its frame number to all clients for an update. The client who sent this request also receives this update and performs a check with its local data handler using the attached frame number to find the equivalent value in its request history. Two equivalent values imply that the server has approved its request and the client safely ignores this.

If a validation check determined that a client’s movement request was invalid, the server broadcasts the client’s locational data from the last valid frame to initiate a “snapback”. This tells the client that its last movement request was rejected by the server and it has to start again from a new location (most commonly where the client was in the last frame). From the client’s perspective, they would see the screen flash and notice that they continued moving from a location in the past. For all other clients, it would appear as though the client stayed at the same location in this frame. When two players are colliding, it would appear as if they both stopped a tiny bit before they ran into each other without being able to continue, after which both players would adjust and continue walking.

We did not allow clients to perform local player-to-player collisions because it would create many fake collisions locally for each client.
3.1.1.4: Shooting

Shooting is the primary part of the experience in FPS games and our study. The previous MQP team implemented a broadcasting server that relays client-authoritative shooting results between connected clients. We moved this authority to the server and implemented Time Warp based, server-authoritative shooting.

Clients determine if they are able to shoot when the player clicks the left mouse button (part of the original FPSci code) and send a reliable UDP packet containing the current frame number to the server when they shoot. This information is then received by the server for physics simulation of the shot.

Originally in FPSci, each game instance can have only one “PlayerEntity” and one “Weapon”, both linked directly to the game instance. The server app had this limitation too since it is a subclass or the client app. To simulate weapon shooting on the server, we had to refactor the game to support multiple shootable weapons in the same game instance. Our implementation was to create a weapon instance for each player instance ("NetworkedEntity" TargetEntity) on the server side which follows the player’s position. For each frame, the server processes all movement data and simulates all shots by firing the respective “Weapon” associated with the shooting player in the current frame, against other players in the world using a projectile-less hitscan. When a player is hit, the server broadcasts an authoritative message and resets the round.

Figure 5 shows a top-down view for the shooter, server, and runner shown every 5 frames. On frame 70 the runner moved into the view of the shooter and sent a movement request to the server. This movement was received, validated, and broadcasted by the server on frame 75. On frame 80, the shooter received this updated information and
immediately fired a shot towards where they saw the runner in their view (ignoring hardware latency and human reaction time). This shot was immediately sent, but only received by the server on frame 85 where from the server’s perspective the runner had already ran out of the view of the shooter. The server casts a shot, getting a miss, and broadcasts that information to all clients.

Figure 5: Networked Shooting with Time Warp off (Miss Case)
3.1.2: Data Handler

To implement Time Warp, server-side movement validation, client-side extrapolation and self location buffer for server-authoritative location, we needed a central data structure to store and retrieve historic information about the players. The Data Handler (DH) infrastructure was implemented to store player information and provide APIs for important game/network features.

3.1.2.1: Server

The primary objective of the Server Data Handler (SDH) was to manage player data for every frame. This was achieved by maintaining a hash map that uses user IDs as keys and an array of ServerDataInput objects as values. At present, each ServerDataInput instance only contains two data points: a boolean for whether a player has fired a shot and the coordinate frame that records the player's position and rotation for a specific frame. We opted for an object structure to facilitate the addition of new information with ease. For instance, potential future features like player health, weapon type, ammo, power ups, etc., could all be stored in this structure and allow an authoritative historic world state to be kept. Additionally, the SDH will always reserve a user-configurable amount of data, allowing for its size to be adjusted based on client latency requirements.

Every frame, the SDH operates by removing the oldest entry from the table. As an example, if the table only stores data for 50 frames, any client action performed beyond the 50th frame is discarded. Once this deletion process is complete, the SDH appends a new empty set to the array. This ensures that incoming data can be easily added to the historical data.
The SDH also contains several other arrays and hashmaps to store information not directly related to a specific frame. These consist of the following variables: “Client Last Valid” stores the last server validated value for each client, which allows for compensation operations to always revert to the most recent valid world state. “Client Latest” stores the newest frame from a client that has not yet been validated, which ensures that if a frame is invalid, it does not overwrite the client’s last valid frame. “Unread Coordinate Frame Buffer” keeps track of all coordinate frames received in the current frame, enabling multiple frames received from the same client on the same server frame to be validated. Finally, “Unread Fired Buffer” stores a buffer of shots from clients for the current frame, allowing for multiple fired reports from the same client on different frames to be executed.

3.1.2.1: Client

Each client also has its own Client Data Handler (CDH), which operates differently than the SDH. The main difference between the two data handlers is that the CDH does not store all positions from each client. Instead, it only stores the ‘n’ most recent values, where ‘n’ is a user-configurable value. The reason for this is that a client does not need to know all of the past information for any calculations it may perform on a specific client. At present, we use three as the default maximum number, but we have left the option open to use more if necessary to facilitate future work. Additionally, the CDH records the client’s past historical movements and compares them with updates from the server. If the server’s position for the local client differs from its own historical value, the client’s position has been invalidated by the server and the client must move to where the server says it should be. The client then updates its own historical value.
Another crucial function of the CDH is to perform extrapolation calculations and store the results. We will delve deeper into this topic in a later section.

3.2: Latency Compensation Implementations

3.2.1: Time Warp

Time Warp is one of the major features in our implementation. It uses historic player location data instead of the most up-to-date data for shooting simulation so that the shooting player would not have to pre-aim (lead) their shots during high levels of network latency.

Our Time Warp implementation is greatly simplified in light of how server-authoritative shooting simulation was designed; thanks to the Server Data Handler, all recent information available to the server is stored for retrieval by the simulation. Before a server-side shooting simulation takes place, we reset the server-side world state to a certain frame number in the data storage based on whether Time Warp was enabled or not. If Time Warp was enabled, we move all clients to their position at the frame number when the shot was taken; if Time Warp was disabled, we move all clients to their position at the current frame number or the position where they were last seen.

Delay can be incurred during both cases of information exchange between the clients and servers, those pertaining to information sent from either the clients to the server or from the server to the clients. With our implementation of Time Warp, we effectively removed the effects of the one-way latency from the client to the server. We did this by casting the shot using the server world state from the client frame number when the
shot happened (in Figure 6, taking the server-side world state at frame 80 instead of 85), referring to this operation as: “half Time Warp”.

The effects of latency can be further mitigated with “full Time Warp” by using the frame number of the world state the client is looking at when the shot happened (using the server-side world state at frame 75, which is what the client is looking at on frame 80). This means taking in consideration of the latency from the server to the client too.

Figure 6: Networked Shooting with “full Time Warp” on (Hit Case)
3.2.2: Extrapolation

Extrapolation is a client-side latency compensation technique with the purpose of predicting the future position of other players in the game based on their past movements. This approach assists in reducing latency and enhancing the game's responsiveness for the local player, as they can observe a more seamless and consistent view of the game world should the client temporarily lose connection with the server as an example.

Extrapolation provides several benefits, such as allowing the local client to predict the movements of other players, reducing the requirement for pre-aiming, and improving overall accuracy. However, there are also potential drawbacks as sudden changes in a player's direction can result in inaccurate extrapolated positions. This can cause inconsistencies between what the local client observes and what is happening on the server, leading to missed shots among other problems. We currently have two primary methods of conducting extrapolation.

The first of these methods is linear extrapolation, which involves subtracting the two most recent coordinate values to obtain a vector for the player's next location. As illustrated in Figure 7, this extrapolation type utilizes the data in frame two and three to determine the slope and predict the player's new position. In the example, the player would be located at (4, 4) in the next frame using linear extrapolation.
Another approach for predicting a player's next position is through quadratic extrapolation. Unlike linear extrapolation, quadratic extrapolation is also capable of taking acceleration and deceleration into consideration due to the inclusion of a third data point, which in turn can help prevent discrepancies between the client and server regarding the player's location as well as improving the estimation of the player’s position. In the example represented by Figure 8, the player was accelerating between frame one and two, but slowed down between frame two and three. By factoring in the change in speed between the data points, the player's movement can be accurately predicted.
Although quadratic extrapolation has greater flexibility than linear extrapolation in terms of prediction capabilities, the method can sometimes produce unexpected results. As an example shown in Figure 9, a player is visibly increasing in speed, yet the prediction is asserting that the player will slow down. Similarly, in Figure 10 a player appears to be decreasing in speed, but the prediction states that they will speed up. As a result, it is important to consider the limitations of quadratic extrapolation despite its greater prediction capabilities.
There are many mathematical formulas available for implementing extrapolation with the two mentioned earlier being the first implemented before the project priorities were changed. Other possibilities include logarithmic and Proportional-Integral-Derivative controllers. Each extrapolation technique has its own strengths and weaknesses that must be taken into account when setting up game trials for user studies. When used in combination with other latency compensation techniques, it can lead to even more accuracy and is worthy of further investigation. The infrastructure has been set up to incorporate different calculation metrics and study their differences and benefits. However, due to time constraints, this aspect of the project could not be pursued further.

3.2.3: Latency Exposure

In the present network architecture that was implemented earlier for FPSci, all incoming server-bound packets are processed in an “onNetwork” event handler method.
that is called once per game tick and is responsible for processing all packets that have asynchronously arrived at the server from the other clients. Although this approach is practical for regular in-game functions such as player input processing and relaying server world information, where the exact arrival and departure times for the packets are negligible, it does not operate well with ping packets. The crux of the issue lies in the rate at which the “onNetwork” method is called; given that the rate of the method call is synchronized to the game’s tickrate, which in itself is bounded by the game’s framerate, any ping packets that would be processed on the main game thread would incur an internal bias from the game framerate and would thus not accurately represent pure network latency. To remedy this issue, our implementation of latency exposure runs on a completely different set of threads that is independent of the main game thread and is thus not bounded by the game’s tick / framerate. Additionally, the game also opens an additional socket and port exclusively for ping packet logistics in order to avoid competing for the socket with the main game thread.

![Diagram of individual program threads for latency exposure](image)

**Figure 11: Individual Program Threads for Latency Exposure**

31
The underlying structure of the latency exposure implementation can be summarized as follows: each client instance of FPSci starts up two additional threads while the server starts one additional thread. On the client, the first additional thread that is launched is the c2s ping thread (Figure 11) that is responsible for periodically sending out ping packets to the server instance at a configurable rate which by default is set to one second. The second client-side thread, the inbound ping handler thread (Figure 11), is responsible for handling arrived ping packets from the server and updating necessary data structures for the visual ping indicator as well as other functions. On the server, the s2c ping thread is responsible for receiving and processing all incoming ping packets from each of the connected clients. Once an incoming ping packet has arrived to the server, it is immediately relayed back to the origin address through the appropriate socket and port.

![Ping](image.png)

**Figure 12: Screenshot of the Ping Indicator**

In conjunction with the additional threads, the client also sends a set of ping statistics that were computed and allocated in the inbound ping handler thread once every half second to the server for logging to the server’s database. Each set of ping statistics contains the following statistical values: the latest recorded RTT between the client and the server, the simple moving average of the first ‘n’ queued RTTs where ‘n’ is set to 5 by default, the minimum recorded RTT in the game session, and the maximum recorded RTT.
in the game session. The ping statistics set is also logged to the respective client-side database and is displayed to the user via an on-screen GUI as shown in Figure 12.

In order for our latency exposure implementation to match the scalability of FPSci’s other features, many different configurations were added in order to allow for the facilitation of user studies and experiments with multiple different parameters and settings. Alongside the pinging frequency and simple moving average size specifications, our implementation also has configuration options for explicitly specifying the port number used for ping packet logistics, threshold ping values for when to change the color of the ping indicator based on the severity of the latency, universal startup toggles for both the display itself and the entire implementation as a whole (e.g. no pinging at all if specified), and settings pertaining to placebo ping modifiers.

Placebo ping modifiers allow test facilitators using FPSci to either apply mutations to the final displayed ping value on the indicator or temporarily prevent the indicator from being displayed on a per-round basis. Our implementation supports four placebo modifier types, each represented as an integer: -1: the indicator is hidden from the player, 0: the real ping value is completely replaced by the placebo value, 1: the placebo value is added to the real ping value, and 2: the placebo value is multiplied to the real ping value. The placebo types and modifier values are both stored as arrays of integers on the server-side experiment configuration files with each entry representing the set placebo type and modifier for that round. Once each round starts, the entries for a specified round are propagated among all of the clients and remain persistent until the next round starts.
3.2.4: Latency Concealment

There are multiple ways to conceal latency to reduce its effect on players. Aside from local movement calculation and immediate rendering, latency concealment in our project was implemented by allowing the user to hear a hit sound as soon as their local shooting simulation determined they made a hit on an opponent, instead of waiting one RTT for the authoritative server to determine and return the result. This allows the user to get immediate feedback on the fired shot from the game. This was implemented by keeping a local simulation of shooting for its prediction of the shot. When this prediction returns a hit, a shot hit sound is played to notify the player.

3.3: Auxiliary Implementations

3.3.1: Shot-Around-The-Corner Detection

As mentioned previously, the shot-around-the-corner problem is of considerable significance in the context of first person shooters and the time warp latency compensation. The detector algorithm that we have implemented allows testers to measure the frequency at which such shots occur during a game session by performing a server-side dual shooting simulation whenever a player shoots and hits another player. In the dual simulation, each shot is processed in both scenarios where time warp was either enabled or not, regardless of whether the compensation was actually active during the round. In order for the algorithm to be able to mark the shot as an ‘around a corner’ shot, three conditions have to be met first in the dual simulation: 1. Time warp was enabled during the round, 2. In the scenario where time warp is enabled, the shooter successfully hit the target player, 3. In the
scenario where time warp is not enabled, the shooter did not hit the target player. If all three conditions are met, an additional algorithm computes the target’s visibility when the shot was fired to determine whether the target was fully concealed (‘around a corner/wall’) or if the target was either partially or fully visible to the shooter, indicating that another type of shot was performed. Once all relevant information is collected and determined, it is appended to a data structure which stores all relevant information on the performed shot. The data structure is then logged to the server database upon being completely allocated.

Figure 13: Illustration of the Visibility Tester Algorithm

To determine whether or not a time-warp-scenario-only successful shot can be classified as a shot-around-the-corner, an additional visibility test is conducted to check whether the target was fully concealed from the shooter when the shot was fired. The algorithm for determining target visibility performs the check by casting two additional horizontal rays in the direction of the target that are slightly skewed to the left and right of the target minus a small epsilon value so as to avoid having the rays be strictly tangential to
the target. The directions for these two additional rays are determined as follows: 1. A ray directed from the shooter's center to the target's center is casted, indicated by the red solid line in Figure 13, 2. Two rays that are horizontally orthogonal to the first ray, the two gray dashed lines in Figure 13, are casted from the target's center and are set to have a length of the player sphere model's radius minus a small epsilon value referenced earlier, 3. From here, two additional rays, the two red dashed lines in Figure 13, are created and casted with a starting point at the shooter's center towards the endpoints of the two orthogonal auxiliary rays. If both of the additional rays do not intersect with the target in the visibility check, the target is determined to be fully concealed or 'not visible' from the shooter and hence the shot is classified as an 'around the corner' shot. In addition to detecting shots around corners, the visibility checker also supports other 'time warped' shot types. If both of the additional rays intersect with the target, the target is marked as 'fully visible'; this type of shot can occur when the target is running out of a concealed location and is shot while already being entirely visible to the shooter. If one of the additional rays intersects with the target, the target is marked as 'partially visible'; this type shot can occur when a target is running towards an obstacle and is shot whilst being partially concealed by the obstacle. The visibility types are also stored and logged to the server database alongside all the other relevant shot information.

3.3.2: Logging

With the addition of all the new latency compensation techniques and other necessary implementations, our team has added several new log entries to FPSci’s logging implementation that are imperative in data analysis. These new entries included: toggles on
each of the latency compensation techniques associated with the current game round number, additional player / client specific configuration settings sent at the start of each round, all information and statistics pertaining to the current network latency for the exposure compensation, and remote shot / fire inputs for shot-around-the-corner detection. In terms of the client-side logged player configuration settings, new added entries include: placebo ping settings which consist of the placebo type and modifier alongside a flag stating whether time warp was enabled or not during the round. Network latency information is logged on both the client and server-side databases and contains all of the ‘raw’ individual ping statistics, those being the latest RTT, simple moving average of the RTTs, the minimum RTT, and the maximum RTT. The client-side database also exclusively includes a duplicate set of the statistics whose values were altered by placebo modifiers meanwhile the server-side database’s entries are truncated from a 32-bit integer to a 16-bit integer so as to reduce the packet size when sending the statistics to the server. Remote fire inputs are logged on the server-side database and consist of the following entries: the shooter and target’s positions, the server-side frame number during which the shot occurred, the target’s visibility, the hit resolutions based on each of the dual simulation’s scenarios, the target and shooter’s identifications, and a boolean entailing whether or not the shot was a shot-around-the-corner.
4. Evaluation

This section details the methodology of our user study in order to evaluate the impact of our latency compensation implementations on player performance and QoE as well as the results of our study. We conducted a user study to assess the efficiency of latency compensation techniques, specifically time warp and latency exposure, in FPS games. The study involved participants playing a total of 20 rounds of FPSci in an on-campus computer laboratory, with and without the application of the latency compensation techniques.

4.1: Methods

4.1.1: Recruitment

To recruit participants for our user study, we sent email invitations to students enrolled in the Computer Science and Interactive Media & Game Development programs at WPI. These emails contained a link to an interest form that prospective participants were asked to complete prior to the game session. After screening for eligibility based on predetermined criteria, we extended an invitation to eligible participants that met our skill and frequency requirements. This invitation included instructions to read and sign the informed consent, options for a preferred time slot, and a review of the provided background information, if desired. Lastly, we offered an incentive for participation in the study, including a $10 gift card of their choice and optional class credit. We believe that these incentives helped to increase participation and motivated participants to provide their best effort during the game sessions.
4.1.1.1: Interest Form

As previously mentioned, prospective participants were asked to complete an interest form (See Appendix A). The interest form included several questions aimed at gathering information about the potential participants' experience with FPS games using a keyboard and mouse, which consisted of a list of pre-selected options as well as a custom field for games that were not from the prior options. The following two questions were used to determine eligibility:

1. Rate your experience with FPS games with a keyboard and mouse on a 1-5 scale from low to high

2. When was the last time you played a FPS game with a keyboard and mouse?
   a. Within the past week
   b. Within the past month
   c. Within the past six (6) months
   d. Within the past year
   e. None of the above

Lastly, the interest form included an optional section where participants could ask any questions or express any concerns they had about the study. This allowed participants to clarify any issues they had before committing to participating in the study.

We extended the study scheduling invitation to participants that had both a 3-5 experience rating and played FPS games in the last 6 months.
4.1.1.2: Scheduling

As previously noted, eligible participants were invited to participate in the study if they met the predetermined criteria. The invitation included instructions to select a preferred time slot, ideally with another participant, in order to facilitate player-versus-player gameplay.

Due to the complexity of our user study, we have listed some criteria for the service we use to schedule prospective playtesters. The criteria for the service included the following:

1. Contain no hard requirements for email sign-ups for guests.
2. Allow multiple (2) people to sign up in one time slot.
3. Allow guests to see available time slots and names of people who have already signed up for a slot.
4. Allow the team to easily modify existing time slots and add slots in bulk by preference.

Optionally, we preferred the service to:

1. Allow the automation of confirmation emails, calendar invites, rescheduling notifications, and post-study follow-up emails.
2. Support linking with the team’s calendar for dynamic availability.
3. Allow the user to see/select who will be facilitating their playtest.
4. Allow file uploads for our informed consent and background information that were to be viewed before the playtest.
5. Support questionnaires during sign-up.
After trying out and evaluating four potential tools including Microsoft Booking, Fantastical Scheduling, Slotttr, SignUpGenius, we opted to use SignUpGenius. This platform allowed participants to sign up for the same time slot and allowed their names to be visible to other participants, which has made it easy for participants to know who they will be playing against and sign up with someone they knew, if they desired.

Since our team shared a lab with other teams, including an Interactive Qualifying Project (IQP) team working alongside us, we took appropriate measures to prevent scheduling conflicts. Specifically, we shared our sign-up sheets as well as sign-up notifications with the IQP team and removed any registered time slots to avoid double-booking in real time to ensure a smooth scheduling process for all participants.

4.1.2: Protocols

The study has different protocols for different temporal portions of the user study sessions to ensure that the data collected is accurate and unbiased. The user study session protocol contains three sections: pre-session protocols, current session protocols, and post-session protocols.

4.1.2.1: Pre-Session Protocols

Prior to scheduling participants, all the hardware that was to be used by the participants was ensured to be identical across the two clients as well as performant enough to handle the local game simulation and completely mitigate factors such as local visual latency. The game was set to run at 60 hz with identical configuration settings across both clients for consistency. To prevent any potential bias caused by participants viewing
the game before the practice rounds, the facilitator ensured that there was no ongoing
session prior to the start of his or her own scheduled session. This involved waiting outside
the lab with the participants until the current session was completed along with all of the
data collection.

Once the ongoing sessions were concluded, the facilitator opened Chrome on the
three designated PCs. For the two client PCs, Chrome accounts were created and
bookmarks were set up for the session form and reaction time test. On the server PC, the
facilitator accessed the interest form to collect the players’ IDs. In the case where the
participants had not provided their informed consent, the facilitator ensured that the
informed consent document was readily available to be read and signed. Lastly, the
facilitator sanitized the equipment before having the participants take their seats to begin
the study.

4.1.2.2: Current Session Protocols

The session starts with participants entering their player IDs into the session form,
which was completed via Google Forms. They were informed that they would use the form
throughout the session to answer questions before the game, after each round, and after the
game session. Participants who had not previously read and signed the informed consent
document were given the opportunity to do so. An explanation of time warp and latency
was provided in the session form for those who had not yet read it, along with an optional
video for better understanding. The facilitator ensured that participants were comfortable
with the topics before proceeding. Moreover, the facilitator emphasized that participants
could ask questions at any point during the session.
After completing the informed consent process, the session form asked pre-game questions along with a link directing to the reaction time test that was to be completed before the start of the first game round. Next, the facilitator opened up the server on the designated PC and launched the clients on the PCs for the participants.

The facilitator would then inform the participants that the goal was to kill the opponent as many times as possible and to avoid hiding from each other. After each round, post-round questions were given to each participant to be answered.

Once all the rounds were completed, the participants answered final post-game session questions. Finally, the participants were able to select a gift card of their choice and the option to receive playtesting credit if desired.

4.1.2.3: Post-Session Protocols

After the session ended, the facilitator copied the database file from the designated path: {root game path}/results. It was emphasized that both the clients’ and server databases were to be copied to a USB and/or another location to prevent potential data loss and desync between game sessions. The databases were renamed according to the participants’ identifications using the naming scheme: (logicalSide playerID.db), with the logical side being set as whitespace if it was a client database. For instance, given two clients with IDs 1 and 2, the client database associated with an ID of 1 would be named as 1.db while the server database would be named as Server 1 2.db.

The facilitator collected the clients’ reaction time files and renamed them following the naming scheme: (playerID.txt), for example, 1.txt. After all the specified changes were applied, the reaction time and database files were then uploaded to their respective Google
Folders. Such file naming conventions ensured that all the data sets were properly organized per user study session. Before closing Chrome on all machines, the facilitator collected any data that had not been uploaded to the Google Drive Folders yet. Lastly, the facilitator left the lab in the same condition as it was prior to arriving; in the case where the lab was previously untidy, the facilitator would clean up the place for incoming participants.

4.1.2.4: Additional Policies

In order to preserve the homogeneity of the collected data sets as much as possible, several additional policies were put in place. These policies are subdivided into four main categories: equipment policies, participant policies, game settings policies, and other miscellaneous policies.

For equipment policies, should there be an inconsistent amount of non-essential present lab peripherals between the clients, the peripheral that was available to one of the clients was to be removed in order to maintain equality of available hardware. The only exception to this were peripherals that were integral to running the game which can include but not necessarily limit to: monitors, mice, and keyboards. The facilitator would be responsible for postponing the user study session should one of the essential peripherals be missing for one of both of the participants.

For participant policies, all participants that were in the game session were to be treated equally so as to avoid any potential favoritism bias that could have arised. In addition, should any one of the participants be disruptive and act as a major inhibition to the study as a whole, the facilitator would be responsible in dismissing the participant from the study without compensations and either take place of the participant or postpone the
session to a later date depending on the availability of other participants. In this case, the database files associated with the dismissed participant were to be appropriately flagged as ‘biased’. In the case where both of the participants are disruptive to the study as a whole, the facilitator would be responsible for dismissing both of the participants without compensation whilst invalidating the session from data analysis as a whole.

For game settings policies, all of the client-side options for all of the configuration files were to be identical across all of the clients. Such client-side options included but not limited to: player movement rates, player respawn positions, toggles on certain latency compensation techniques among other internal features, as well as other visual settings.

Miscellaneous policies pertain to courses of action should there be an unanticipated game crash in the middle of the session. In the case of a game crash, the facilitator was responsible for saving all of the client-side and server-side database files, from there shifting the artificial latency, placebo type, and placebo modifier arrays in the server-side experiment configuration file by the index of the round number before the game has crashed. From there, the server and clients are restarted and the session is continued normally. At the end of the game session, both the backed up client and server-side databases along the new databases are appropriately flagged to signify that the associated session experienced a game crash.

In terms of the equipment policies, all essential lab peripherals were available to all participants, requiring no sessions to be postponed. For the participant policies, none of the participants were reported to be disruptive and no game sessions had to be canceled in regards to these policies. Client-side configuration options for the game settings policies were checked by the facilitator before the start of the first game session for each scheduled
day of user studies and were ensured to not have deviations. Lastly, a minute amount of in-game crashes have occurred during the game sessions, with the facilitator for each respective session flagging the collected data accordingly.

4.1.2.5: Database Flags

To properly signify any unanticipated events that could have occurred during the user study session, an indicator would be added to each of the database files in the form of special character flags at the end of the session. The naming convention for such flags is as follows: ‘logicalSide signUpNumber [flags].db’, where the special character flags are inserted inside the square brackets and replace the ‘flags’ placeholder text. Our protocol contains four specific flags: N: data collected uses an updated or newer version of the game, R: one of the participants was replaced due to not being compliant with instructions and was disruptive to the session, B: the data is potentially biased due to one or more possible reasons, and C-#: the game has crashed at round ‘#’.

4.1.3: User Study

4.1.3.1: Description

To evaluate the impact of latency compensation techniques on player performance and QoE in FPSci, we conducted a user study involving 20 short, 50-second player-versus-player close-quarter combat rounds. The rounds were divided into two groups, each consisting of 10 rounds with the time warp settings either being enabled or disabled. Throughout the different game rounds, we applied and tested various latency values from a selection of 0 ms, 250 ms, and 500 ms, which included unaltered visual
latency values alongside placebo-modified values to evaluate the impact of showing real vs. fake values on the ping indicator. The first round in each group was classified as a test round and was discarded from the final analysis.

We quantified player performance by the number of successful shots that each player hit in the game session in tandem with which latency compensation was enabled at the time the shots were fired. Participants were asked to complete survey questionnaires before the game, after each round, and after the game (See Appendix B). The pre-game questionnaire aimed to assess participants’ FPS experience prior to the user study, including their views on the importance of ping display, how frequently they notice lag in games, success in shooting while lagging, and being shot despite being out of view of the shooter. The post-round questionnaire consisted of questions about the participants’ perspective of the accuracy of the ping indicator, their network latency experience, whether they felt that time warp was enabled, and whether they felt that they were shot despite hiding behind a wall. Finally, the post-game questionnaire asked about the impact of the ping display on gameplay and performance from the participants’ perspective.

Our user study aimed to evaluate the efficiency of two latency compensation techniques, time warp and latency exposure; each of the techniques was explained in detail to the participants before the game. By comparing the results of the rounds with and without the application of these techniques, we aimed to assess the impact of these latency compensation techniques on player performance and QoE.
4.1.3.2: Data Collection

In our data collection process, we utilized three different methods of storing information. Real-time gameplay data was captured and recorded in an SQLite database, which included important details such as player position, rotation, shots, hits, and latency values among other entries. Player reaction times were stored in a separate plain text document while their responses to the session form questions were recorded in CSV format. By connecting each of the three distinct records using a client ID, we were able to derive metrics such as accuracy and ping values, which enabled us to initiate data analysis.

4.2: Results

This section discusses the results of our user study and statistical significance of our implementations. It is important to note that after data collection we discovered an internal error that caused a significant increase in latency of around 50±20ms.

4.2.1 Demographics

Our user study involved a total of 42 participants; however, technical issues in FPSci resulted in 24 datasets being excluded from the final analysis. As a result, we had a total of 18 valid participants with an average experience level of 4.056 and a standard deviation of 0.8726 in playing FPS games with a keyboard and mouse. As shown in Figure 14, a histogram depicting participants’ responses to the question "Rate your experience with FPS games with a keyboard and mouse" summarizes the population sizes for each experience type. The responses were rated on a scale of 1 to 5, with 1 indicating low experience and 5 indicating high experience. Moreover, all participants reported playing FPS games within
the past year of completing the interest form, as depicted in Figure 15 – a histogram illustrating the responses of user study participants to the question "When was the last time you played a FPS game with a keyboard and mouse?"

![Figure 14: Participant Ratings of FPS Experience with Keyboard and Mouse](image)

![Figure 15: When Participants Last Played FPS Games with Keyboard and Mouse](image)
In Figure 16, we present the player reaction time test statistics for the study’s participants. The average reaction time was approximately 200 ms, with maximum values exhibiting a wide range centered around 240 ms, while minimum values had a narrow range centered around 160 ms. It is important to note that we excluded several maximum values that exceeded 600 ms. These outliers were due to a misunderstanding of the reaction test and were not representative of the actual reaction speed of the participants.

Figure 16: Reaction Test Data
4.2.3: Ping Display

Figure 17 displays the three latency groups utilized in the study, expressed in milliseconds. The three groups represent a round trip time of 0 ms, 250 ms, and 500 ms respectively. Due to the effect of an internal error mentioned previously, the actual round trip times were deviated to be around 50 ms, 300 ms, and 550 ms respectively. Each latency group had up to three potential latency display types: ‘actual latency’, a ‘lower placebo latency’, and ‘latency not being displayed’, with the 0 ms latency group having only ‘actual latency’ and ‘latency not being displayed’ types. Initially examining the ping values that were hidden from the player can provide a reference point for evaluating players' QoE. Consistent with expectations, player experience diminishes as latency increases. However, when comparing the hidden ping values with the actual values, we observe slightly differing outcomes. In the case of low latency, player experience improves when they are presented.
with the actual value instead of the hidden ping. However, at high latencies, players’
experience worsens when they are shown the actual ping value.

Another approach used in the study involved using ping values with placebo
modifiers. As an example, for a player’s actual ping value, such as 250 ms, the value shown
on the indicator to the player would be 175 ms. We observed a comparable trend, where at
moderate values, the player experience remained slightly poorer, while at high latency
values the player experience improved when exposed to the placebo ping value.

In our study we also investigated whether the participants could distinguish
between a genuine and placebo value. Figure 18 depicts the same latency groups as before,
with players’ evaluations of the accuracy of the ping display presented on the y-axis.
Generally, players perceived the ping display to be occasionally inaccurate. However, often
they were unable to differentiate between actual and placebo values.

Figure 18: Accuracy of Ping Display
A one-way ANOVA was conducted to examine the effect of placebo ping on player perception at latency levels of 250 ms and 500 ms. The results showed a statistically significant effect in both latency conditions, with \( p < 0.05 \). Specifically, at a latency of 250 ms, the \( F(1, 200) = 3646.41, p < 0.0001 \), and at a latency of 500 ms, the \( F(1, 135) = 3079.48, p < 0.0001 \).

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>df</th>
<th>F</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>(1, 200)</td>
<td>3646.41</td>
<td>&gt;0.0001</td>
</tr>
<tr>
<td>500</td>
<td>(1, 135)</td>
<td>3079.48</td>
<td>&gt;0.0001</td>
</tr>
</tbody>
</table>

Table 2: Ping Display Accuracy ANOVA Test

### 4.2.4: Time Warp

Comparing time warp on and off we can see noticeable accuracy improvements at both 250 and 500 latencies, while 0 ping stays relatively the same.

<table>
<thead>
<tr>
<th>Latency (ms)</th>
<th>Accuracy time warp off</th>
<th>Accuracy time warp on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35.5%</td>
<td>35.0%</td>
</tr>
<tr>
<td>250</td>
<td>16.4%</td>
<td>20.6%</td>
</tr>
<tr>
<td>500</td>
<td>13.0%</td>
<td>17.6%</td>
</tr>
</tbody>
</table>

Table 3: Accuracy with Time Warp Off/On
By visualizing Table 3 (Figure 19), we see that at a low latency there is not a large variation in accuracy as expected; however, at higher latencies we see a much larger increase in accuracy, anywhere between 5-10% increase in accuracy per round. Using several pairwise t-tests to investigate the significance in the three latency groups with time warp off and on. The pairwise t-tests were conducted for time warp off and on conditions at different latency levels with \( p < 0.05 \). The results showed no significant difference between the time warp on (\( M = 0.350, SD = 0.126 \)) and time warp off (\( M = 0.355, SD = 0.138 \)) conditions at latency 0 ms, \( t(53) = 0.246, p = 0.403 \). At a latency level of 250 ms, there was a significant difference between the time warp on (\( M = 0.206, SD = 0.122 \)) and time warp off
(M = 0.164, SD = 0.089) conditions, t(71) = 2.83, p = 0.003. Similarly, at a latency level of 500 ms, the pairwise t-test results indicated a significant difference between the time warp on (M = 0.176, SD = 0.110) and time warp off (M = 0.130, SD = 0.089) conditions, t(53) = 3.00, p = 0.002.

<table>
<thead>
<tr>
<th>Latency</th>
<th>Time warp</th>
<th>mean</th>
<th>Std.</th>
<th>df</th>
<th>t stat</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>on</td>
<td>0.350</td>
<td>0.126</td>
<td>53</td>
<td>0.246</td>
<td>0.403</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>0.355</td>
<td>0.138</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>on</td>
<td>0.206</td>
<td>0.122</td>
<td>71</td>
<td>2.83</td>
<td>0.003</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>0.164</td>
<td>0.089</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>on</td>
<td>0.176</td>
<td>0.110</td>
<td>53</td>
<td>3.00</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>off</td>
<td>0.130</td>
<td>0.089</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Accuracy with Time Warp Off/On Pairwise T-Tests

A one-way ANOVA was conducted to compare the effect of a player perceiving if time was enabled and if it was actually enabled. There was a significant effect at p<0.05 level for the two conditions [F(1, 718) = 6.52, P = 0.01087]. This result indicates that players possess the ability to perceive whether time warp is enabled or disabled.

Taking a step back, Figure 20 depicts the evolution of player experience and kills per round. The average value per round is represented in a solid line, The shaded region is the confidence interval and the dotted line is the trend line. Players tend to improve their skills and achieve more kills as they progress through each round. Despite this however, their experience level remains relatively constant during the entire session. This highlights that regardless of the latency, players will enhance their gameplay with time, but the level of enjoyment they derive from the game may not necessarily increase.

55
Examining the difference when time warp is disabled, Figure 21 demonstrates that player experience has a negative slope as the rounds progress. Nevertheless, the number of kills per round still increases as indicated by the positive correlation. Even with time warp off, players may improve their gameplay, but may begin to develop aversion towards the game due to the latency issues.
Figure 21: Amount of Kills and Experience Per Round Time Warp Off

In contrast, Figure 22 portrays a more pronounced positive slope for kills per round, accompanied by a positive slope for player experience as well. This indicates that, with compensation techniques like time warp, players will tend to continuously enhance their gameplay and derive more pleasure from the game as time passes.
A one-way ANOVA test was performed to investigate the correlation between a player's QoE and the number of kills they obtained in a round. There was a significant effect at $p<0.05$ yielding a $p$-value of $[F(1, 718) = 125.11, P = >0.001]$, indicating statistical significance. This implies that a player's perception of their network experience is greatly influenced by their performance in a given round. A closer examination of the previously displayed graphs reveals that the peaks and dips in the player's kills and quality of experience are closely related to one another.

Out of the total number of shot instances, there were 133 cases of shots fired around the corner. Shot around the corner accounts for 15% of all hits. Additionally, there were 36 instances where a player was hit while being partially in cover due to time warp
compensation. Partially hidden accounts for 3.6% of all hits. It is important to emphasize that occurrences at 0 are due to the presence of unaccounted network latency that was not discovered until after the study was complete. We do not think these values are exactly correct and further analysis is needed to understand why.

<table>
<thead>
<tr>
<th>Latency</th>
<th>Occurrences</th>
<th>Population of hits %</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>67</td>
<td>23.0%</td>
</tr>
<tr>
<td>250</td>
<td>34</td>
<td>9.6%</td>
</tr>
<tr>
<td>500</td>
<td>52</td>
<td>14.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Occurrences</th>
<th>Population of hits %</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>3.8%</td>
</tr>
<tr>
<td>250</td>
<td>17</td>
<td>4.8%</td>
</tr>
<tr>
<td>500</td>
<td>8</td>
<td>2.2%</td>
</tr>
</tbody>
</table>

Table 5: Shot-Around-The-Corner

Table 6: Partially Hidden

### 4.2.5: Summary

In summary, the display of ping in a game can influence how players perceive the game's latency. Notably, whether or not ping is displayed affects how much latency players perceive, with the game appearing less latent when ping is hidden for high latency values, yet appearing more latent when ping is hidden for low latency values. Further research, such as conducting similar studies with lower ping values, could provide insight into how players perceive the game based on their ping at more typical values.

Additionally, the study revealed that time warp can provide an increase in close quarter combat accuracy at high ping thresholds, with accuracy improving by an average of 5-10%. Moreover, players have shown a learning curve earning more kills across the game rounds regardless of whether time warp is utilized or not, as indicated by their
performance improving in getting more kills over time despite the latency. Despite this, time warp was observed to offer a unique advantage in this regard by shortening the learning curve for shooting and increasing in kills per round, thus enabling players to enhance their abilities at a faster rate. Alongside improvements in the rate of skill enhancement; the study also demonstrated that time warp, in combination with different latency exposure metrics, can also significantly change a player's enjoyment of the game.
5. Conclusion

The purpose of this project was to investigate the impact of latency compensation techniques on player performance and quality of experience in the FPS game FPSci. Our investigation involved implementing and extending FPSci's client-server multiplayer networking to include multiple latency compensation techniques, including time warp, latency exposure, extrapolation, and latency concealment. We then conducted a user study to measure the effects of time warp and latency exposure on player performance and experience. Our results showed that time warp was able to significantly reduce the effects of latency on player performance and experience, while the effectiveness of latency exposure varies with the actual latency values.

In conclusion, our study highlights the importance of addressing network latency in FPS games and the potential benefits of implementing effective compensation techniques. Our findings suggest that time warp can be a useful technique for mitigating the effects of latency on player experience and performance, and future research can explore how it can be further optimized. Overall, our study contributes to the understanding of how network latency affects player experience in FPS games and provides insights for improving the user experience in the future.
6. Future Work

Short Term

Due to time constraints, we could not thoroughly examine all the recorded data and its impact on players. Furthermore, there are numerous other studies we could conduct, but we only had the opportunity to conduct one. We could perform various user studies with different experimental variables and implementations, including those we tested in our study as well as those we could not investigate in time.

Moreover, there are some data validations that need to be addressed, such as instances where a player shoots their gun, dies before the packet gets to the server and respawns, while their shot still registers as hitting the target. There are a few other minor inconsistencies in various areas that could be fixed if given enough time.

Medium Term

Player-to-player collision is a feature that could be refined and addressed in greater detail. Presently, we determine the distance between two players on the server-side and push them back to their previous location if they intersect with each other. While this method yields the most accurate result, it is susceptible to network latency. To mitigate its effects, client-side player-to-player collision can be added along with client-side extrapolation of other players’ movements to help reduce the effects of network latency in collision detection by performing the calculation locally.

We developed our code to be easily customizable, allowing for quick implementation and testing of various versions of time warp and extrapolation without requiring significant rewrites. By adding another case statement and function, another research team could use
different metrics and methods to conduct studies. A future MQP could prioritize enhancing the robustness of our versions and incorporating a wider range of compensation tools, testing them for their effectiveness, enjoyment, and fairness.

Long Term

FPSci is useful for testing a single player experience but does not facilitate multiplayer well. Future teams could explore the possibility of developing their own game engine or utilizing a pre-existing engine with tools to simplify implementation. This approach could create a robust and customizable structure similar to what currently exists while abstracting some of the existing software's necessities that cause issues in a multiplayer setting. Using a pre-existing engine could also provide players with a closer feel to what industry games may be running on, being able to then introduce simplified player models and maps to the game. Lastly, they could create a new structure for implementing various latency compensation methods, with a strong emphasis on customizability and multiplayer support. This game would work similarly to FPSci with a focus on research as well as on multiplayer experimentations instead of just single player.
References


Appendices

Appendix A: Interest Form

FPSci User Study - Interest Form

Interested in *getting paid* to play a video game?
Get your friend to sign up so you can play against them!

Please *sign up by Friday, March 24* to indicate your interest.

Details:
- Play a **1v1 match against another participant** lasting around 30-35 mins
- Fill out a demographic questionnaire
- Provide survey opinions on the gameplay
- Receive a **$10 gift card of your choice** for participating
- Receive **IMGD playtesting credit**, if needed
- Voluntary study with little to no risk
- Approved by the Institutional Review Board (IRB) at WPI
- All COVID-19 precautions will be taken

For all questions, contact gr-fpisci-mqp-22-23@wpi.edu.

Sign in to Google to save your progress. Learn more

* Indicates required question

WPI Email *

Your answer
Rate your experience with FPS games with a keyboard and mouse. *

1  2  3  4  5
Low  ○  ○  ○  ○  ○  High

When was the last time you played a FPS game with a keyboard and mouse? *

○ Within the past week
○ Within the past month
○ Within the past six (6) months
○ Within the past year
○ None of the above
What FPS games with a keyboard and mouse have you played? (Optional)

☐ CS:GO
☐ VALORANT
☐ Apex Legends
☐ Overwatch
☐ PUBG
☐ Call of Duty
☐ Tom Clancy's Rainbow Six Siege
☐ Other: ____________________________

Any questions or concerns? (Optional)

Your answer ____________________________
Appendix B: Session Form

FPSci User Study - Session Form

Sign in to Google to save your progress. Learn more

* Indicates required question

Informed Consent

If you have not already sent a signed copy of this Informed Consent:

1. Read the document
2. Once you are done reading, please notify us to sign.

Player ID *

Please ask for your player ID.

Your answer
Network Latency can cause shots to miss
In online games, there is often a delay between the player and the server, which can cause the server’s version of events to be slightly ahead of what the player sees. This can result in missed shots when a player tries to shoot another player who appears to be in range, but in reality, has already moved out of the way on the server.

Time Warp can help!
Time warp is a technique used in online gaming to help players with latency issues. It works by telling the server to process a player’s shots as if they had taken place in the player’s world timeline, rather than in the server’s world. This compensates for the delay and allows the player to successfully hit other players, even if they have moved on the server. Essentially, time warp “warps” the server’s time to match the player’s time during the shot.

Without time warp, players may need to lead their shots to compensate for the delay and hit other players. This can be difficult and require a lot of practice, but with time warp, it is not necessary.

A Video Explaining Time Warp & Latency Compensation
Pre-Session Questions

How important is the ping display to you? *

1  2  3  4  5
Not Important  ○  ○  ○  ○  ○  Important

How often do you notice lag or delay in your games? *

1  2  3  4  5
Never  ○  ○  ○  ○  ○  Often

How often have you been in a situation where you are lagging but you manage to * successfully shoot another player?

1  2  3  4  5
Never  ○  ○  ○  ○  ○  Often

How often have you been in a situation where you were shot despite seemingly * being out of view of the shooter?
For example: when you just ran behind a wall, but you were still shot.

1  2  3  4  5
Never  ○  ○  ○  ○  ○  Often
Reaction Time Test

1. Open the **Reaction Time Test** tab.
2. Complete the test.
3. Hit save.
4. Notify us when you are done.
Post-Round Questions

1. Did you feel that the value displayed by the ping indicator was accurate? Only select if the ping display was on.
   - 1 2 3 4 5
   - Not Accurate
   - Very Accurate

2. Rate your network latency experience: *
   i.e. How bad were you lagging?
   - 1 2 3 4 5
   - Terrible (very laggy)
   - Excellent (little to no lag)

3. Do you feel that time warp was on? *
   Note: Time warp makes it so that you don't have to lead your shot when latency is high.
   - Yes
   - No

4. Did you get shot even though you hid behind a wall? *
   - Yes
   - No
Post-Round Questions were the same for all 20 rounds of gameplay. The percentage shown in the form corresponds to the round the participant was on.
### Post-Game Questions

#### How much did the ping display affect the way you played? *

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not at all</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

#### Do you feel that the ping display helped you with your overall performance? *

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not at all</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

#### Following the previous question, why or why not? *

Your answer

---

76
Choose the kind of $10 gift card you would like! *

We would appreciate you for choosing one of the specified ones, but if you select "Other," please keep in mind that we might or might not be able to accommodate and might need to fall back to one of the specified ones — we'll contact you if this is the case.

**Digital:** sent to your email from giftcards.com

**Physical:** purchased at Price Chopper, dropped to your WPI mailbox, or scratched and get image of code sent through email

- [ ] Physical - Amazon
- [ ] Physical - Apple & App Store
- [ ] Physical - Starbucks (available if 2 more people select)
- [ ] Digital - Grubhub
- [ ] Digital - Subway
- [ ] Digital - Google Play
- [ ] Digital - Game Stop
- [ ] Digital - Spotify
- [ ] Digital - Roblox
- [ ] Digital - Chipotle
- [ ] Digital - The Cheesecake Factory
- [ ] Digital - AutoZone
- [ ] Digital - IHOP
- [ ] Digital - ACE
- [ ] Other: ________________________________
Which email or WPI mailbox number do you want your Gift Card to be sent to? *

Your answer

Do you want IMGD playtesting credit? *

- [ ] Yes
- [ ] No
# Appendix C: Table of Authorship

<table>
<thead>
<tr>
<th>Report Section</th>
<th>Author(s)</th>
<th>Editor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>Sitsanok, Ivan, Alex, Yihong</td>
<td>Ivan, Yihong, Alex</td>
</tr>
<tr>
<td>Background</td>
<td>Alex, Ivan, Yihong, Sitsanok</td>
<td>Ivan, Sitsanok</td>
</tr>
</tbody>
</table>

## Implementation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Author(s)</th>
<th>Editor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoritative Server</td>
<td>Yihong</td>
<td>Yihong, Ivan</td>
</tr>
<tr>
<td>Data Handler</td>
<td>Alex</td>
<td>Yihong, Ivan</td>
</tr>
<tr>
<td>Time Warp</td>
<td>Yihong</td>
<td>Ivan</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>Alex</td>
<td>Ivan</td>
</tr>
<tr>
<td>Exposure</td>
<td>Ivan</td>
<td>Ivan</td>
</tr>
<tr>
<td>Concealment</td>
<td>Yihong</td>
<td>Ivan</td>
</tr>
<tr>
<td>S.-A.-T-C. Detector</td>
<td>Ivan</td>
<td>Yihong</td>
</tr>
<tr>
<td>Logging</td>
<td>Ivan</td>
<td>Ivan</td>
</tr>
</tbody>
</table>

## Evaluation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Author(s)</th>
<th>Editor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td>Sitsanok, Alex, Ivan</td>
<td>Ivan, Sitsanok, Yihong</td>
</tr>
<tr>
<td>Results</td>
<td>Alex, Sitsanok</td>
<td>Alex, Ivan, Sitsanok</td>
</tr>
<tr>
<td>Conclusion</td>
<td>Sitsanok</td>
<td>Ivan, Yihong</td>
</tr>
<tr>
<td>Future Work</td>
<td>Alex</td>
<td>Ivan, Sitsanok</td>
</tr>
</tbody>
</table>
## Appendix D: Technical Achievements

<table>
<thead>
<tr>
<th>Technical Implementation</th>
<th>Primary Contributor(s)</th>
<th>Secondary Contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structure and APIs</td>
<td>Alex</td>
<td>Yihong, Sitsanok</td>
</tr>
<tr>
<td>Server Data Handler</td>
<td>Alex</td>
<td>Sitsanok</td>
</tr>
<tr>
<td>Client Data Handler</td>
<td>Alex</td>
<td>Sitsanok</td>
</tr>
<tr>
<td>Authoritative Server</td>
<td>Yihong</td>
<td>Sitsanok</td>
</tr>
<tr>
<td>Server-Authoritative Location</td>
<td>Yihong</td>
<td>Alex</td>
</tr>
<tr>
<td>Server-Authoritative Shooting</td>
<td>Yihong</td>
<td>Alex, Ivan</td>
</tr>
<tr>
<td>Server-Side Validations</td>
<td>Yihong</td>
<td>Ivan, Sitsanok</td>
</tr>
<tr>
<td>Latency Compensation Techniques</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Warp</td>
<td>Alex, Yihong</td>
<td>Ivan</td>
</tr>
<tr>
<td>Shot-Around-Corner Detection</td>
<td>Yihong, Ivan</td>
<td></td>
</tr>
<tr>
<td>Latency Exposure (Ping Display)</td>
<td>Ivan</td>
<td></td>
</tr>
<tr>
<td>Latency Concealment</td>
<td>Yihong, Sitsanok</td>
<td>Alex</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>Alex</td>
<td></td>
</tr>
<tr>
<td>Additional Logging Entries</td>
<td>Ivan</td>
<td></td>
</tr>
<tr>
<td>Quality Testing</td>
<td>Alex</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix E: Non-Technical Achievements

<table>
<thead>
<tr>
<th>Research Facilitation</th>
<th>Primary Contributor(s)</th>
<th>Secondary Contributor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Playtests</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Marketing Material</td>
<td>Sitsanok</td>
<td>Yihong</td>
</tr>
<tr>
<td>Invitation</td>
<td>Sitsanok</td>
<td>Yihong</td>
</tr>
<tr>
<td>Interest Form</td>
<td>Sitsanok</td>
<td>Yihong</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Yihong, Sitsanok</td>
<td></td>
</tr>
<tr>
<td>Session Form</td>
<td>Sitsanok</td>
<td>Yihong, Ivan</td>
</tr>
<tr>
<td>Session Protocol</td>
<td>Ivan, Sitsanok</td>
<td>Yihong, Alex</td>
</tr>
<tr>
<td>User Testing</td>
<td>Sitsanok, Yihong</td>
<td>Alex, Ivan</td>
</tr>
<tr>
<td>Playtester Rewards</td>
<td>Yihong</td>
<td>Sitsanok</td>
</tr>
<tr>
<td>Data Analysis</td>
<td>Alex</td>
<td>Yihong</td>
</tr>
<tr>
<td><strong>Organization</strong></td>
<td>Sitsanok</td>
<td>Yihong, Ivan</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td>Ivan, Yihong, Sitsanok</td>
<td>Alex</td>
</tr>
<tr>
<td><strong>Executable Compilation</strong></td>
<td>Alex</td>
<td>Ivan</td>
</tr>
<tr>
<td><strong>Demonstration Material</strong></td>
<td>Yihong, Alex, Sitsanok, Ivan</td>
<td></td>
</tr>
</tbody>
</table>