Time Warp and Prediction Analysis in a Top-Down Shooter Game

A Major Qualifying Project
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfilment of the requirements for the
degree of Bachelor of Science
In
Computer Science
by

Ben Hetherington

Alex Osler

Date:
6 March 2020

Professor Mark Claypool, Advisor
Worcester Polytechnic Institute

Abstract:

Online video games have grown in popularity, bringing latency as a challenge for game developers. We studied the impact of two latency compensation techniques, time warp and prediction, on player performance, player experience, and system load by developing a networked, top-down shooter with both techniques. We conducted a user study with 32 participants, varying the latency and enabled compensation techniques across 16 game rounds and experiments with up to a hundred AI players. While neither latency compensation technique substantially impacted player performance, prediction made latency less noticeable. Based on our data and time warp's constraints it may not be worth including in a top-down shooter, whereas prediction is more effective as it reduces players' perceptions of latency.

Table Of Contents:

1. Introduction

The round trip time, or latency, a packet takes when traveling to and from a server will add an inherent delay to player interactions in any traditional client-server game. This delay can vary significantly, but it can easily affect a player's experience and performance in time sensitive tasks. While the impact of latency can be mitigated by improving bitrate, one can never entirely remove its effects. Many users nowadays expect their applications to work over the Internet, and this is especially true with regard to video games. Making video games playable over the Internet substantially broadens the pool of potential players for a multiplayer game, but also increases the impact of network latency on the game.

In many games latency has a definite effect on player performance, and as such latency compensation techniques are an important area of study (Quax et. al., 2004, p. 156). There is no consensus on which techniques should be used, but the five options we have chosen to consider are time warp, prediction, world manipulation, visual manipulation, and displaying ping.

Time warp operates by changing the server's game world so it emulates the client's. Once the server receives an event from the client, the server calculates what game state the client was at when the event took place. The server then turns its game world back to that state. It applies the update to the game world and fast forwards the game back to its original state (Juckett, 2016). Using this technique, the client gets a reactive experience with a reduced number of distracting visual glitches.

Many popular games such as "Counter-Strike: Global Offensive" (Valve Corporation, 2012) and "Overwatch" (Blizzard Entertainment, 2016) currently implement a form of latency compensation known as prediction (Ford and, Orwigg, 2016) (Source Multiplayer Networking, 2001). With prediction, the client takes the last game state and attempts to predict where the

other entities in the game will move. It then displays these to the player, and allows the player to interact with them. Once a new server state is received, it moves the out of place entities to their correct locations, and starts the process again (Source Multiplayer Networking, 2001). This creates a smooth experience for the player, even if their latency is high or the server updates infrequently.

Both prediction and time warp have been researched for their effects on player performance, but these studies have stayed consistently separate. When prediction was studied in a paper by Pantel and Wolf, they found that it had a beneficial effect on player performance in a wide variety of game types (2002, p. 84). Similarly, when Palant, Griwodz, and Halvorsen tested prediction on a simple tank game, they found that it meaningfully improved player accuracy (2006, p. 5). Time warp has been shown to have similar positive effects. When studied by Sun, the author found that time warp increased users scores in a simple 2d game (2017, p. 29). In another study, Lee and Chang found that "Counter-Strike: Global Offensive" players had higher accuracy when playing with time warp (2015, p.3).

While all of these studies provide evidence as to the effectiveness of these techniques, none set out to test them together. Even in Lee and Chang's paper, where both techniques were studied, they specifically disabled time warp when testing prediction, and tried to minimize prediction's effect on their testing of time warp. To fill this gap, the goal of our project is to evaluate latency's effect on player performance, and specifically how effective prediction and time warp are when used in conjunction.

We test these methods by adding latency to a network game and testing with human participants. We created a relatively simple top down player vs. player shooter, which creates a situation where players interact in a controlled environment. Using this game, we gathered and

examined metrics from players, such as their hit rate, to measure player performance. We collected these metrics under a variety of different latency and latency compensation situations, and used them to determine how differing latencies and compensation techniques affect player performance. Additionally, we gathered subjective data on players opinions of latency, and the visual glitches latency causes.

This study was performed with 32 participants. The data from this made it clear that latency does not have a significant effect on player accuracy when using projectile weapons, however hitscan weapon accuracy decreases as latency increases. These results hold true with both forms of latency compensation on and off. Prediction does, however, decrease how noticeable latency is while time warp does not have the same effect on perception.

In this paper, we first provide a chapter on the background information needed to understand our study and its results. It then outlines how our game was implemented, and why we choose time warp and prediction. Next it describes our study design, and a few performance experiments we performed on our game. Finally, it analyzes our study and experiment data, and provides conclusions based on this data.

2. Background

With the Internet's growth, users are expecting more and more technologies to be interconnected with others around the world. This is especially true with video games, where multiplayer games make billions of dollars (Arif, 2018). With the size of this industry, gamers expect a smooth and enjoyable experience. Unfortunately, the multiplayer aspects of these games have an inherent issue with latency, where the distance between players creates a delay in information transfer between them. This chapter discusses what latency is, what causes it, how it affects players, and how developers can compensate for it.

2.1. Latency

Latency is commonly used to describe the time it takes a packet to travel from the user's computer to the server, or back to the user's computer (Hoffman, 2017). There are many factors that contribute to latency. The speed of a user's computer affects the time it takes to send a packet. Once it has left the computer, the Internet adds latency, due to routing and physical propagation time. Finally, when the packet reaches the server, its response is also delayed by processing time.

2.1.1. Client Side Latency

Latency on the client side can have many causes. One reason is that the client's computer may be saturated with other processing requirements. When the computer is busy, it delays network operations. All games also run in discrete steps, known as frames. Generally, input is gathered once a frame, so there is inherent latency depending on the speed at which frames are generated. Additionally, the monitor adds delay in displaying the frame. Finally, input devices

poll inputs discrete intervals which can cause a delay in processing input, adding to the total latency (Ivkovic, Stavness, Gutwin, Sutcliffe, 2015, p.136-137).

2.1.2. Internet Latency

Another source of Internet latency is physical distance, combined with the limit of the speed of light. If the server is across the US from the client there will be a minimum latency of approximately 16ms as the signal will take at least that long to travel across the country. In addition to this, there will be some unavoidable overhead from the Internet's routing process. Each routing node the packet passes through will have to make a decision on which direction to send the packet. Additionally, other users may be sending data through these routers, so with competing traffic, the router may be forced to buffer packets it receives until later, so it can have time to process all of them (Ng, 1997). These elements mean that congested routers can add latency.

2.1.3. Server Side Latency

Once a packet finally reaches the server, it can be subject to many of the same issues that added latency at the client. For instance, the server is a computer as well, so if it is overloaded then the newly received packet may face delays in processing. A server can also experience network bottlenecks similar to the client when sending a response back, and these could cause further slowdowns for the same reasons mentioned above. Additionally, many different server configurations can affect the speed of the packet. For example, the most common architecture, lock step, functions by waiting until all users have sent in their current game states to send out the new, updated state (Bernier, 2001). This means that clients are delayed by the

client with the highest latency to the server, and so lower latency clients can experience slowdowns as the server simply waits for the other clients. Additionally, in a time warp configuration the server can only process one packet at a time, even if there is excess processing power (Juckett, 2016). Due to this, a client could experience more latency waiting behind another client's packet before theirs is processed.

2.2. Effects of Latency on Gamers

The effects of latency on gamer performance have been well studied and provides a compelling argument for why minimizing or compensating for latency is important to gamers. For a general baseline, when Raaen, Eg, and Griwodz performed a latency test involving a simple USB button, they found the median perceptible latency value was 90ms, with some users able to detect values as short as 26ms (2014, p. 3). These values are generally significantly shorter than the latency of networked games, so when working on these games the latency must be kept in mind. A separate study, done by Armitage, researched the effects of lag on players of Quake 3 (2003, p. 140). In this study, they concluded that, so long as they have an alternative, users will generally stop playing on a games server if the latency becomes higher than 150-180 ms. Additionally, numerous studies have found that latency has an adverse effect on players performance in games. In a study done on Unreal Tournament, the authors found that latency had a notable impact on player performance (Quax, Monsieurs, Lamotte, Vleeschauwer, Degrande, 2004, p. 156). Similarly, work done by Pantel and Wolf found that latency also negatively affects players in racing games (2002, p. 29). Overall latency has a negative impact on gamers, decreasing enjoyment and performance.

2.3. Latency Compensation Techniques

As has been shown, latency has a significantly negative effect on user performance. To improve players' experience, the effect of latency must be mitigated. While some of this can be solved by hardware (having enough servers, making sure the servers are powerful enough, putting in faster networks), much of it is unavoidable, and must be solved in other methods. To this end, developers have created a range of latency compensation techniques, which all aim to lessen the negative effects of latency.

2.3.1. Time Warp

Time Warp, or rollback, is a technique where a server changes its state to adapt to the clients' state. Once a packet comes in, the server calculates approximately how long ago the client actually pressed the inputs sent in the packet. The server then turns its game state back to when the action was performed, and simulates said action on that state. Once the result is determined, it then returns the server to the state it was in when the packet was received, and continues the process (Juckett, 2016). This technique allows for players to have responsive gameplay, even with high latency, giving the client a smooth experience. Unfortunately, time warp does have a few drawbacks. First of all, it can be challenging to implement. Second, this turning back and fast forwarding time can require considerable processing and therefore requires a powerful server to run. Finally, time warp can cause some noticeable visual glitches when multiple clients send conflicting states into the server.

Sun tested time warp on a simple networked computer game, where a player's ship flew up and down in a 2d plane and shot at incoming enemies. The author determined that users achieved higher scores with time warp implemented as compared to without (2017, p. 29).

Authors Lee and Chang performed a study where they tested multiple latency compensation techniques on the popular first person shooter game, “Counter-Strike: Global Offensive”. When testing time warp, they found a statistically significant increase in player’s accuracy while playing (2015, p.3).

In an attempt to overcome some of the negatives of time warp, authors Lee and Chang published a paper detailing a new time warp implementation that attempts to fix the issue of players being “shot behind cover” by adding an extra check into the technique. (2018, p. 292-293) With this check, the server asks the player being shot if the shot was invalid, i.e. blocked by some object. The player then responds with the validity of the shot. If the player claims the shot was invalid, the server runs the same algorithm to confirm, and then cancels the shot. By using this one check, the authors effectively reduce the occurrences of visual glitches, without compromising player accuracy.

2.3.2. Prediction

Prediction involves the client extrapolating the current game state from its last received state, and displaying that to the user. This can be done in different ways, but the simplest one is to repeat the previous action continually until a new state is received. By doing this, the game can estimate where other players and objects appear to be on the server and give players a more accurate and responsive game state. Once the new state arrives, if the received state is substantially different than the current one, the game will correct the differences. Prediction can, unfortunately, cause issues if another user acts in a different way to the predicted state. When this happens, the client can experience visual glitches, such as shots going through players and being shot through walls. For some games, such as racing games, this is quite uncommon as

players generally do not need to make quick changes. In others, such as twitch shooters, however, it can often create issues as other players and objects change directions and states often (Source Multiplayer Networking, 2001). Fortunately, prediction is a relatively simple implementation in most game states. A traditional client generates a new state based off a given set of inputs, so the developer can predict upcoming inputs for the next state, and display the potentially predicted state.

Overall, the negatives from prediction are relatively small compared to the performance benefits. When authors Pantel and Wolf studied prediction they investigated a wide variety of games and prediction styles. These games varied from sports, to 3d action, to simulator and racing games. On each of these games, they tested seven types of prediction, varying from simply repeating the current velocity and/or acceleration to predicting movement based on the user's given input. With all of these tests, they found that prediction reduces the impact of latency and improves the consistency of the game (2002, p. 84). In a second study Palant, Griwodz, and Halvorsen tested prediction on a basic, 3d tank fighting game they had created. When compared against no prediction, they found that prediction provided a benefit to players accuracy, and so recommended that it be used (2006, p. 5). These results support the effectiveness of prediction, and demonstrate why it is common in the industry.

Even relatively recently, there has been significant research into improving prediction. Much of this research has focused on improving the actual prediction of the player's next moves, and one such example is Kharitonov's article about a motion-aware algorithm for determining player positions (2012, p. 259-260). By taking into account the player's recent movements, the algorithm tries to fit them to a common motion model and predicts their next location based on that. Using this model, the program can then more accurately predict locations, while sending

fewer location updates. Kharitonov's tested this model without latency and found that this algorithm provided a significant increase in prediction accuracy while reducing the number of packets sent. In further testing done by Fernando, Almeida, and Felinto, the authors found that even when relevant latency was added, this new algorithm performed better than the previous, motion agnostic one (2018, p. 145-146).

2.3.3. World Manipulation

World manipulation is a latency compensation technique whereby parts of the game's world are modified depending on the amount of latency. The theory is that, by doing this, the developers can make the game easier for players with higher latency (Lee I., Kim, Lee B., 2019, p. 1). This works by changing parameters which directly affect the player's ability to play. For instance, in a shooting game, a developer could increase the size of a player's hitbox depending on their latency. This would make it easier for the player to hit the enemy, thereby making them perform better. This technique is also easy to implement in most engines, as changing small parts of the game world is a common feature, and is included in game engines such as Unity and Gamemaker Studio (Unity, 2019) (YoYoGames, 2018). Unfortunately, this technique does not actually address the fact that latency makes a game feel unresponsive. This can leave the game feeling sluggish, even if the players' scores are comparable across latency values.

This negative does not, however, mean the technique is ineffective. Lee I and Lee tested the concept on a popular game known as "Flappy Bird" (2019, p. 10). In this game, the user must click the screen to navigate their avatar of a small bird through openings of varying heights. The authors implemented world manipulation by changing the size of these openings in proportion to latency, and they found that this method did effectively improve the performance of players with

latency (2019, p. 10). In another study written by Sabet, Schmidt, Zadltootaghaj, Gridwodz, and Moller the authors tested world manipulation on an open source game called Somi, where a player runs forward on a 2d plane and shoots upwards at enemies at various heights (2018, p.118-119). To manipulate the world the authors changed the time limit on the game and/or the size of the enemy's hitboxes in accordance with the latency. This study, as well, showed that world manipulation can have noticeable improvements on the performance of players.

2.3.4. Visual Manipulation

Visual manipulation is a relatively simple technique where latency is hidden by displaying an immediate response. This can take the form of a car revving up when a button is pressed, but not actually moving the car until the server responds, or in a fireball spell beginning the chant but not shooting a fireball until the server is notified (Armitage, Claypool, Branch, 2006, p. 97). Unfortunately this technique may work for specific situations and not others. For instance, this technique would not work in a fast FPS, where the player expects the gun to fire as soon as the button is pressed. Even if the game makes some noise upon input, the player will notice that the gun is not firing, and will then feel that the game is unresponsive. In the situations where it is applicable, visual manipulation effectively hides latency from the player, and so it is used in many modern games.

2.3.5. Displaying Ping

The last latency compensation technique we describe is the simplest one. With this technique, the developer displays the latency of the connected players. While this does not do anything to actively compensate for latency, it does give the players themselves the ability to

adjust their play too compensate. When the players can see the latency of themselves and other players, they get a better understanding of how the game is functioning and they can better compensate for it in their play (Gutwin Benford, Dyck, Fraser, Vaghi, Greenhalgh, 2004, p. 507).

2.4. Summary

Latency has negative effects on gamers by lowering their performance and enjoyment of games. To evaluate the benefits of latency compensation, we will be implementing both time warp and prediction in a game of our own creation, and running user studies to determine the effectiveness of these techniques.

3. Methodology

Our goal is to determine the effectiveness of both time warp and prediction when used together. To this end, we have implemented a simple top down multiplayer shooter titled “Six Shooter”. Inside of this game, we have implemented time warp and prediction, as well as a method of simulating latency. Using this game, we conducted user studies to measure user performance and enjoyment with time warp and prediction compared to the game with no compensation.

3.1 Six Shooter

“Six Shooter” is the basic multiplayer top-down shooter we implemented. In this game, players travel around a closed map and attempt to shoot at each other and increase their score. Players can use a variety of weapons, and their goal is to have the highest score at the end of the one minute round. A screenshot of this game is provided below as Figure 3.1.

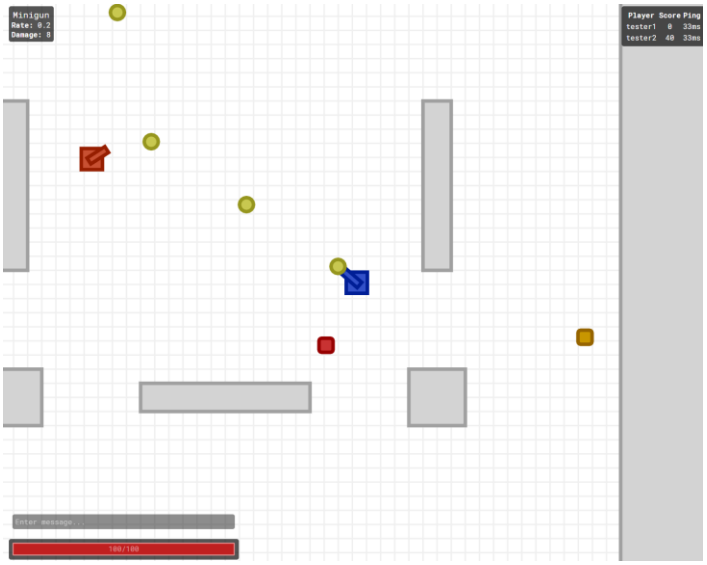


Figure 3.1. A screenshot of the game “Six Shooter”

3.1.1 Game Design

A full game design document is in Appendix A, but we summarize the main design goals here. “Six Shooter” is meant to be played with four players, as that provides a balance between action and playability. Combat is done on a single square map with strategically placed barriers to facilitate interesting play. Each player is assigned a corner to spawn at, where they are initially placed and where they respawn if killed. From here, they can traverse the map and search for different weapons, which appear as small colorful squares. Once they have a weapon, they only lose it by dying, or purposefully dropping it with “Q”. Each weapon has infinite ammo, and can be used to damage other players. Points are assigned with damage and kills. Players have one hundred health that does not regenerate. Each point of damage dealt increases their score by one. Additionally, each kill gives a flat one hundred points on top of the score from damaging other players.

3.1.2 Game Engine

The full code of our game is available in the GitHub repo linked in Appendix B, but an overview of the games design and implementation is provided here. Our game runs on an Ubuntu server with a custom game engine we developed in JavaScript. We choose JavaScript for its multiplatform support through web browsers, quick development cycle, and included networking infrastructure. For networking specifically, we used websockets, a network protocol that works over TCP, and a central Node.js server for the code to run on. The graphics are drawn using Two.js, and only appear on the client side.

The engine itself runs an event based game loop. The game generates events periodically as the game runs, and sixty times per second, the game engine processes all the queued events to

create the current game state. A simplified pseudo code example of this game loop is written below.

1. Game loop starts from timer
2. While the event queue has events:
    a. Take the first event from the event queue
    b. Broadcast the to all listening handlers, which perform the game logic
3. Create new step event and explicitly handle it to notify objects of time passing
4. Increment clock
5. Sleep until next step is called by the timer

After each iteration of this loop, the changes from the previous game state are sent to all connected players, at which point the clients update their game states appropriately. By default, the client has very few functions, as it simply sends the player entered key states to the server and updates the game world whenever it receives a new state. Additionally, as the game is running the server logs every event into a separate file for later analysis. Each part of this process is encapsulated by a class, as described below.

- The GameManager class handles the game logic and running the game loop
- The NetworkManager class handles connections with the current players and sending out or receiving information
- The AudioManager handles the games sound effects
- The LogManager logs all events and any information we specifically need to be logged
- The WorldManager manages the state of the game world and its entities
- The Entity class is what all game entities inherit from, including the player, weapons, weapon pickups, and projectiles

3.1.3. Game Logic

Inside the game almost all of the event creation and game logic is driven by entities. This means that most every part of the game is represented as an entity, even the intangible ones. For instance, the weapons each player controls are actually a weapon entity, which the hero entity controls. Whenever a hero collides with a pickup entity, they are given the corresponding weapon entity, and the pickup entity is removed. The hero entity specifically has significant control over the rest of the game. We use this hero entity for most operations we have to perform for specific players, including storing networking information and their score. Movement as well is tied to the entity.

Each frame the game engine invokes a move method on every relevant entity, moving the entity the appropriate distance according to their current acceleration, velocity, and friction. This provides nuanced movement and interesting mechanics, such as bouncing bullets. In addition, whenever an entity moves, the movement is in two “steps”, checking if its bounding box intersects any other entity’s bounding box at each of these steps. This gives a finer resolution when checking for collisions during fast movements. We additionally make some optimizations during this phase. Entities that are not collidable simply move their full movement speed in a single step without ever checking for collisions.

3.1.4. Network Implementation

Much of the low level networking is handled by websockets. We have provided some layers on top of this, however, to efficiently send objects and game states over the network. The client sends key-down and key-up inputs to the server to keep the keyboard’s state updated.

Using this information, the server generates each new game state. Once a state has been generated, each entity is serialized by converting them to JSON objects and excluding fields that can be determined from other existing data. Each JSON object is then converted to text, and transmitted together to the client. Once the client receives the JSON test, it deserializes it by re-deriving the removed fields, and then it updates the corresponding entities based on the received data. After that process is completed it displays the new state to the client, and the loop repeats.

3.2. Time Warp

As described in Chapter 2.3.1, time warp is a latency compensation technique, whereby the server's game world is changed to emulate the client's. To implement this technique, the server must be able to estimate the approximate game state the client was at when they performed an action. Then, whenever the player sends a command, the server reverts its state back to what the player's state was when the command was created, performs the command, and then fast forwards the game back to its current state. From there, the game simply continues as normal.

3.2.1. Why Implement Time Warp

We choose to implement time warp because it is commonly used by commercial games making it more likely our results would pertain to existing games. Additionally, time warp allows players with highly disparate latencies to play the same networked game.

3.2.2. Time Warp Implementation

To implement time warp, the engine stores previous game states so it can revert back when needed. The engine store's all the serialized objects that get sent out to the game clients, so these objects can be used to revert the game world. These objects are stored in one separate, game state object. These game states are then stored in a queue which can contain up to thirty states, which allows the server to revert about 500ms backwards. In addition to these objects, the engine stores all input events and step events in a separate queue. This allows it to fast forward the game state once the state has been reverted. To estimate player latency, clients send in a timestamp with their input events. This time stamp is then used to keep a running estimate of how delayed the client is compared to the server. Using this infrastructure, when a packet comes in, the engine performs time warp in the following process. First, the engine receives a packet that is supposed to have time warp performed with it. Second, the server determines the approximate time this action was performed, and iterates through the game state queue until it has the closest game state to its estimation. Then, the event queue is iterated over as well, to gather all the events that have occurred since that game state. Third, the server is reverted to that game state, and the client's new event are added into the events to be handled. Finally, the server handles all the events and recreates the new, current game state.

Figure 3.2 illustrates how time warp works, using the perspectives of a client and a server. On the client side, the client shoots the enemy when said enemy comes into the client's crosshairs. This shoot command is then sent to the server, but by the time the server receives it the enemy has moved past the crosshairs. Time warp then activates, and rolls back the state to what it believes the client saw. The shoot command is then added in, and the server re runs the

game until it reaches its previous server state. At this point, both the client and the server have the same state, and time warp finishes.

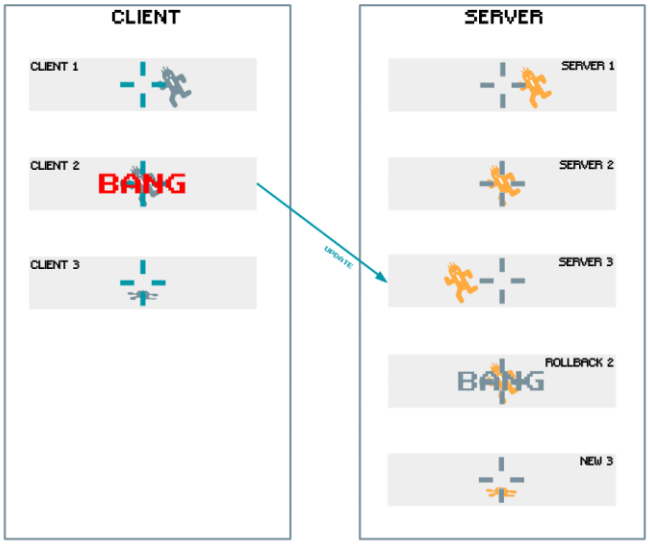


Figure 3.2. Time Warp illustration

Original Art by Meseta, Accessed on 12/9/19, Available at

https://medium.com/@meseta/netcode-concepts-part-1-introduction-ec5763fe458c

3.3. Prediction

Chapter 2.3.2 provides an explanation of prediction, a technique where the client predicts the position of game objects it does not have current data for, and displays that prediction to the player. With prediction, the client performs the game logic of moving an object. Then, the client updates each object every frame, moving them in their predicted direction of travel. The client must also be able to reconcile a new, contrary game state received from the server with their current, potentially incorrect state.

3.3.1. Why Implement Prediction

We implemented prediction since it is commonly used by commercial games. Additionally, it allows players with latency to have a game state similar to the server's game state.

3.3.2. Prediction Implementation

When implementing prediction we split it into two separate forms, player and opponent prediction. For player prediction, the engine predicts the client's movement so that when the player presses a direction, they will see themselves moving immediately. To perform this, the client simply processes user input before it receives movement confirmation from the server. Once the server responds with its estimated player position, the engine checks to see if the client's position is sufficiently close to the server's. If they are close enough, the client then tells the server to update its position to the clients. Opponent prediction, on the other hand, is used to predict the trajectories of bullets and enemy players. Bullets continue on their trajectory until the client receives an updated position. For enemy players, the engine keeps their input and acceleration the same so they continue moving in the same direction until it receives a new position for them. When a new position is received for either entity, they are simply snapped to their correct location, and the prediction process is restarted. With these two prediction forms, the client can show predicted states for all moving objects and make the game more responsive.

3.4. User Study

To study the effectiveness of time warp and prediction together, we designed a user study to isolate each technique. This study consists of two distinct parts. First we had participants fill

out a demographics and game experience survey. Second, we had participants play our game with different amounts of latency and latency compensation on or off and provide subjective data at the end of each round. Using this study, we evaluate how effective both latency compensation techniques are when used in tandem, compared with no latency compensation.

3.4.1. Survey Design

Our survey was handed out before the study participants began playing the game and was designed to give us some additional background information about the participants. The survey itself is pictured in Appendix C, and began by asking the participants to report their age and gender. The survey then asks them how many hours a week they play video games, and how good they believe they are at PC action games, on a scale of one (low) to five (high).

3.4.2. Study Procedure

Our study consists of sixteen one minute rounds of our game. Four participants join in and play each round out sequentially. There is a small three question subjective form at the end of each round, which is administered inside the game and is available at Appendix D. In this form, we ask the players how disruptive visual glitches were, how noticeable the latency was, and which players they believed had high latency. During the study, the game adds constant amounts of latency for some rounds. When latency is added, one player is always left with zero ms of latency. For instance, when a round has 75ms of latency, that actually means that three players have 75ms of latency, and one player has 0ms of latency. For all the rounds of the study, we go through a set order of latencies and latency compensation techniques which are shown in Figure 3.3.

Rounds	Latency Compensation Techniques	Simulated Latencies (ms)
1-4	None	0, 75, 150, 300
5-8	Prediction	0, 75, 150, 300
9-12	Time Warp	0, 75, 150, 300
13-16	Time Warp and Prediction	0, 75, 150, 300

Figure 3.3 Table of study rounds

3.4.3. Study Environment

This study was administered in Kaven 203, a computer lab on WPI's campus. Inside this lab, players sat at four computers lined up in a single row. While they were sitting, we would hand out the consent forms and surveys, and then we would explain the study procedure and game controls. The clients had 64 bit Windows 10 Enterprise installations, 1920x1080 monitors, Google Chrome installations, Intel i7-8700 CPUs, 16GB of Ram, and Ethernet connections to the WPI network. These computers were acting connecting to our main game server, which was running at another place on campus. This server was equipped with an Ubuntu installation, Intel i7-3770 CPU, 11GB of Ram, and an Ethernet connection on the WPI network.

3.5. Experiments

In addition to the user study, we conducted performance experiments on our system and the latency compensation techniques we used. These experiments measured how our implementation of time warp affected the CPU load, memory use, and how our network traffic scaled up with increasing numbers of players.

3.5.1. Performance Procedure

To test time warp's performance, we implemented a basic AI that moves randomly and shoots the closest enemy within line of sight. Additionally, we implemented a command to rollback a specific amount of time. When the AI spawned, it would either have a machine gun that did no damage, or no weapon at all. We spawned different amounts of these AI, and recorded the performance as the game ran with them. We varied the number of AI, the frequency with which time warp was called, and how the AI's weapon worked. We did multiple different tests with this set up, which are described in Figure 3.4.

Test	**Amount of AI**	**AI Weapon Bullets**	**Time Warp Frequency (per second)**
Performance Test	10, 20, 30, ...100	Yes	Never
Time Warp Test	5, 10, 15, ...50	No	Equal to the number of AI
Network Test	10, 20, 30, ...100	Yes	Never

Figure 3.4 Table of experiment parameters

The first two tests were to determine the performance with and without time warp. To record performance, we called the Linux top command once per second, using grep to select just the line which contained the performance of our server. We piped this output into a text file, and once the test was over we imported the file into a Microsoft Excel file, so we could analyze it. For each testing round we recorded one minute of performance information using this setup. Inside the game, we simply let the AI interact while moving our character to a corner of the map, so we would not affect them. When we added in time warp, we sent commands for rolling back 50ms into the server using our implemented chatbox. To send these in at the correct speed, we used the keyboard macro program AutoHotKey.

Our third test was designed to determine the networking performance of our game. For this test we used Wireshark on the client's computer to record all packets sent to and received from the server. This data was then exported to a csv, which was further imported into a Microsoft Excel file for analysis. During the test we would spawn the correct amount of AI, and move around them to simulate a normal match. Like the previous test, we recorded one minute of network data for each round.

The final test we performed was to determine the amount of time it takes to time warp, depending on the total number of entities. To be more precise, we disabled the machine guns so the number of entities on the server would stay constant. From here, we called the time warp command at different time warp amounts, and different entity counts. Our command reports the amount of time that the time warp operation took in seconds. We sampled the amount of time it took to time warp for each combination of entity count and time warp amounts 10 times and put all of the data in a spreadsheet to analyze. The different combinations of AI and time warp amounts are shown, in Figure 3.5. All combinations of Time Warp and AI are run.

Number of AI	10, 20, 30, 40, 50
Time Warp Amounts (ms)	0, 75, 150, 300

Figure 3.5. Table of time warp duration experiment parameters

4. Results

This chapter analyzes the data from our study and our experiments. Both of these methods provided data, analyzed for its relevant insights below.

4.1. Study Data

Our main source of data was the user study we performed with our game. This is detailed in Section 3.4, but in summary it consists of two main parts. The first part is a demographic survey we asked every participant to fill out before the test began. The second part was the main test, where we had participants play our game with varying latencies and latency compensation techniques. We performed this test with a total of 32 people, and we outline our findings below.

4.1.1. Survey Data

For this survey participants were asked to fill in their age, hours playing video games each week, gender, and estimated proficiency at PC action games, on a scale of 1-5. Generally, our test subjects were 20 year old college students, with relatively average gaming proficiency who played games fairly regularly. This data is display in Figure 4.1. Additionally, we found that most of our participants were women, with 66% self reporting as female, while only 34% reported as male.

	Age (Years)	**Hours of Video Games Per Week**	**Estimated PC Game Proficiency**
Mean	19.59	7.38	2.40
Median	20	5.5	2
Mode	20	0	3

Figure 4.1. Table of Survey Data

4.1.2. Effects of Latency

During our main test, participants were given multiple weapons to choose from in gameplay. Here, we focus on the data from two of these weapons. The first is a pistol, which fires one medium speed projectile when the player clicks. The second is a raygun which acts as a hitscan weapon by firing one laser per click. This raygun additionally had time warp enabled during the appropriate testing rounds. Figures 4.2-4.5 graph the accuracy of these two weapons over a variety of latencies and latency compensation techniques. For each graph, the blue line shows the accuracies of unlagged players, while the orange line shows lagged players' accuracies. The x axis shows the lagged players' latencies, and the y axis shows the percentage of shots hit. Additionally, error bars indicate the standard error of the mean. For each weapon we have two graphs, one displaying accuracy during rounds with no latency compensation, and the other showing accuracy when rounds had latency compensation enabled. In Figures 4.2 and 4.3 the pistol's accuracy stays relatively constant throughout the different latencies, showing that the performance of the pistol is not strongly affected by latency. Figures 4.4 and 4.5 show the raygun consistently decreasing in performance as the latency increases

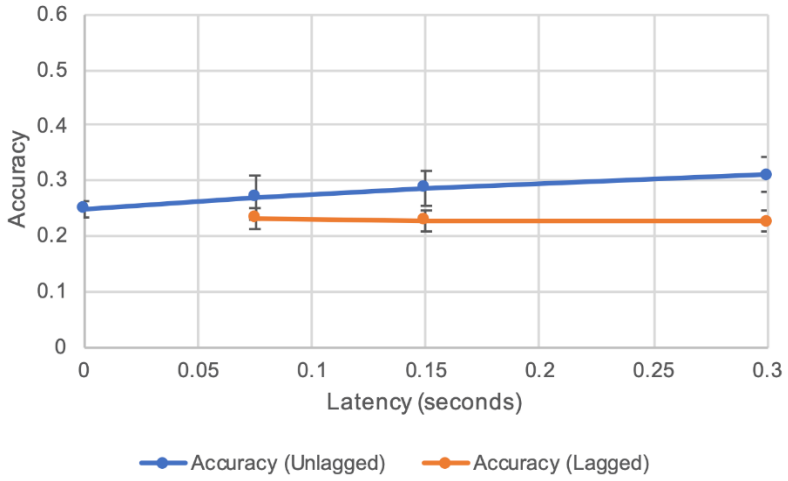


Figure 4.2. Graph of Pistol Accuracy without Latency Compensation

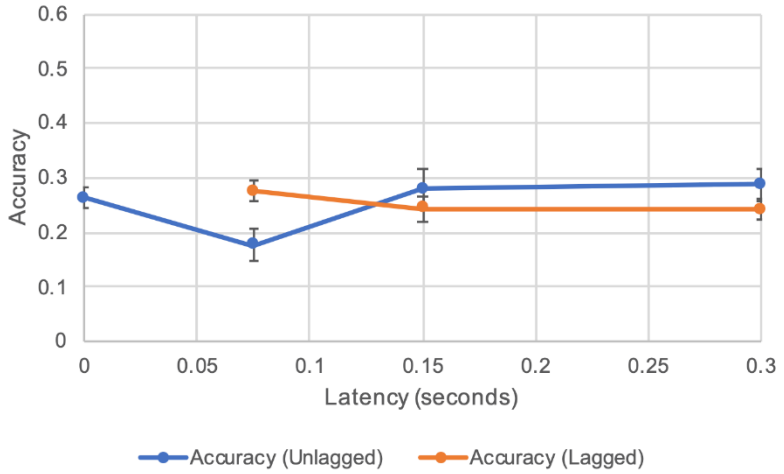


Figure 4.3. Graph of Pistol Accuracy with Latency Compensation

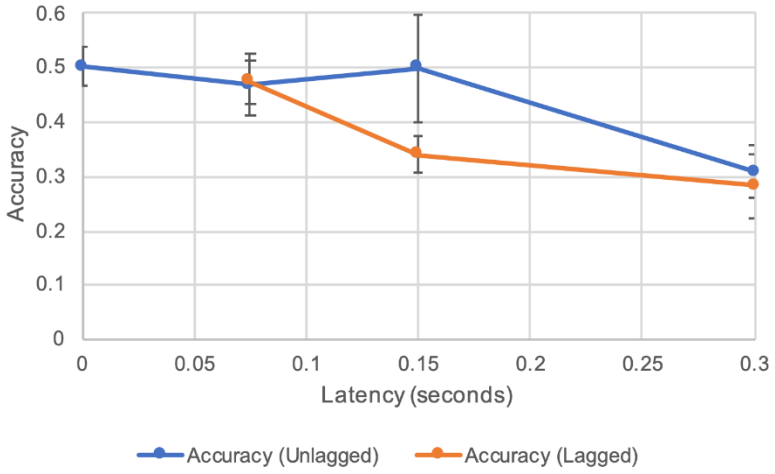


Figure 4.4. Graph of Raygun Accuracy without Latency Compensation

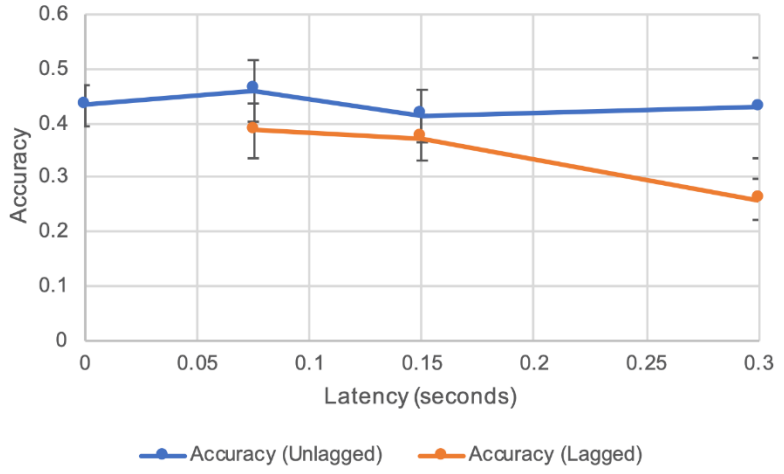


Figure 4.5. Graph of Raygun Accuracy with Latency Compensation

4.1.3. Effects of Latency Compensation

In Figures 4.6 and 4.7 the latency of the players is graphed against their accuracy with the pistol and the raygun. The blue line in these graphs shows the accuracy during rounds with no latency compensation, and the orange line shows the accuracy during rounds with latency compensation. As before, error bars show the standard error of the data. In both figures, the blue and orange lines are similar, demonstrating how players did not have higher accuracy when latency compensation techniques were enabled.

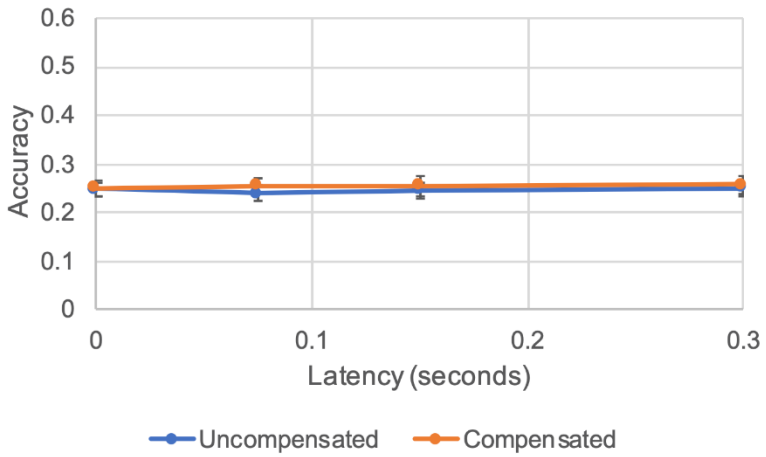


Figure 4.6. Graph of Overall Pistol Accuracy

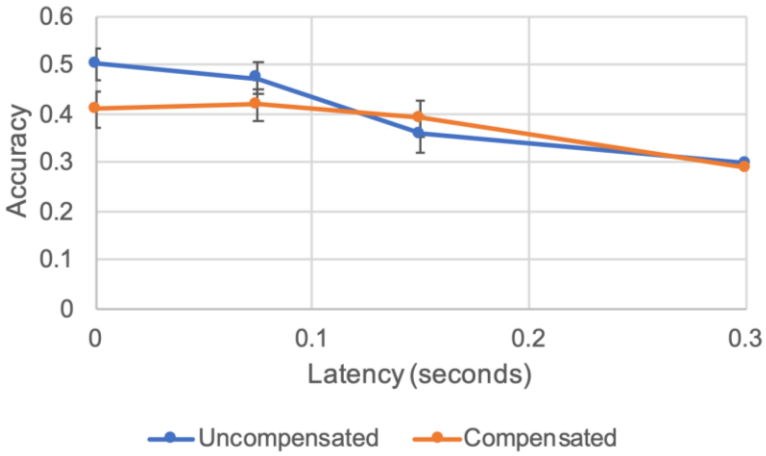


Figure 4.7. Graph of Overall Raygun Accuracy

At the end of each round, we surveyed players on how noticeable the latency was, on a scale of 1 to 5. In Figures 4.8 and 4.9 we have graphed the latency of the game round against the players rating of how noticeable latency was. The blue line shows the results in uncompensated rounds, while the orange line shows results in rounds with the given latency compensation. In Figure 4.8 this orange line is prediction, while in Figure 4.9 it is time warp. In the first figure both lines stay separate, with the prediction consistently causing a lower reported score. In the second figure, however, notice how both of the lines stay closer together and converge on very similar values.

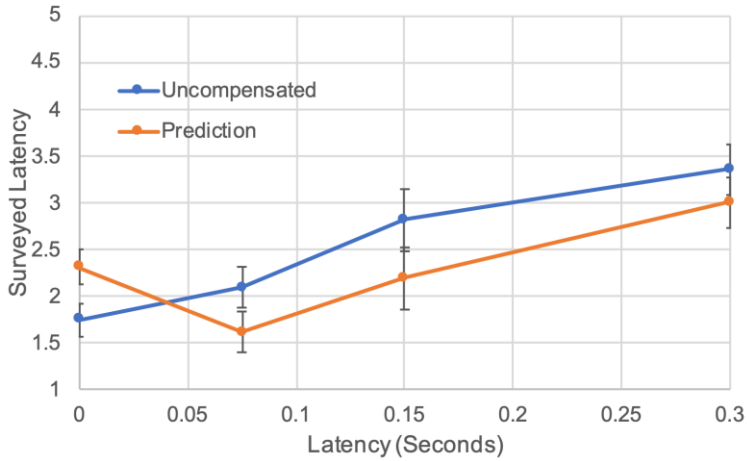


Figure 4.8. Graph of Perceived Latency against Latency Amounts with Prediction

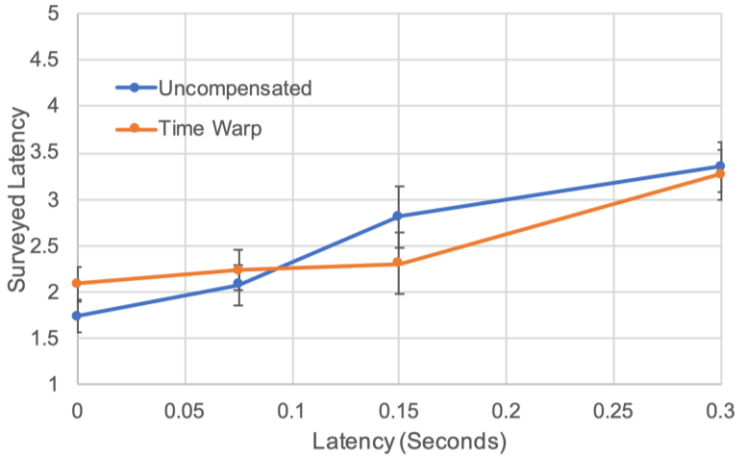


Figure 4.9. Graph of Perceived Latency against Latency Amounts with Time Warp

4.2. Experiments

We performed three experiments on our game to determine the performance and network requirements of our game. The first experiment tested the GPU load of time warp, the second tested the processing time of time warp, and the third tested the network impact of the game.

4.2.1. Time Warp Performance

To determine time warp's effect on performance, we recorded CPU load with varying numbers of AI players. We then performed this test again, but performed regular time warps during the test to see how it affected the load. Figure 4.10 is a graph of the CPU load against the number of AI. The blue line shows the test with no time warp, and the orange line shows the test when time warp was performed. Notice how in the graph the orange line is much steeper, and stops at 45 AI, as our system could not handle any more.

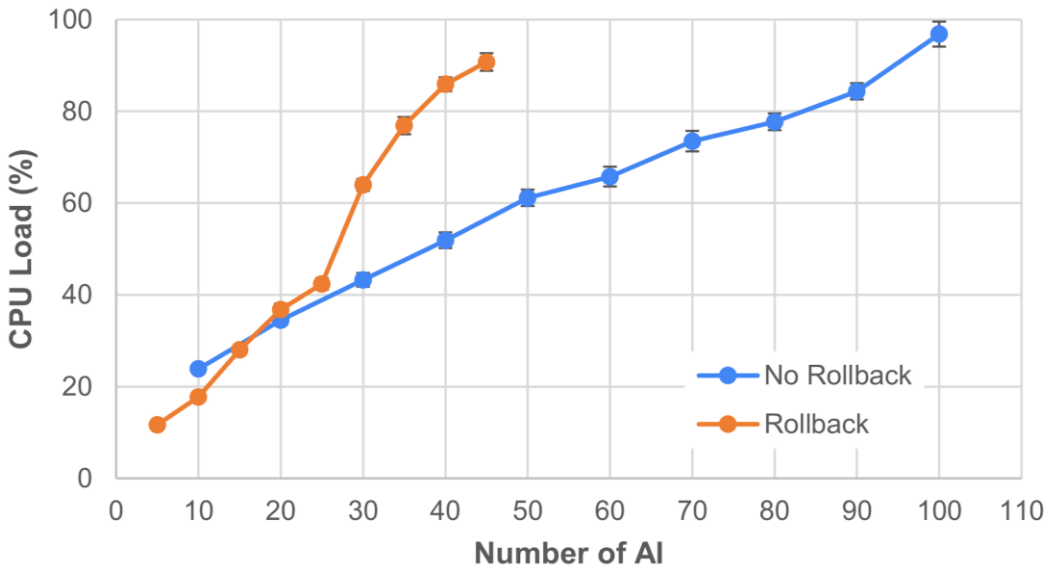


Figure 4.10. Graph of CPU Load With and Without Time Warp

4.2.2. Time Warp Duration

To test how long time warp takes, we conducted an experiment where we varied the number of entities and the amount of time rolled back, and recorded how long each time warp took under these changing parameters. Using this data, we determined that time warp duration increases at a rate proportional to $n * m^2$, where n is the amount of time rolled back, and m is the number of entities. To illustrate this relation, Figure 4.11 is a graph of the average time warp duration against the result of this equation, with the standard error represented as error bars. This produces a straight line with a correlation of 0.995, as both equations increase at the same rate.

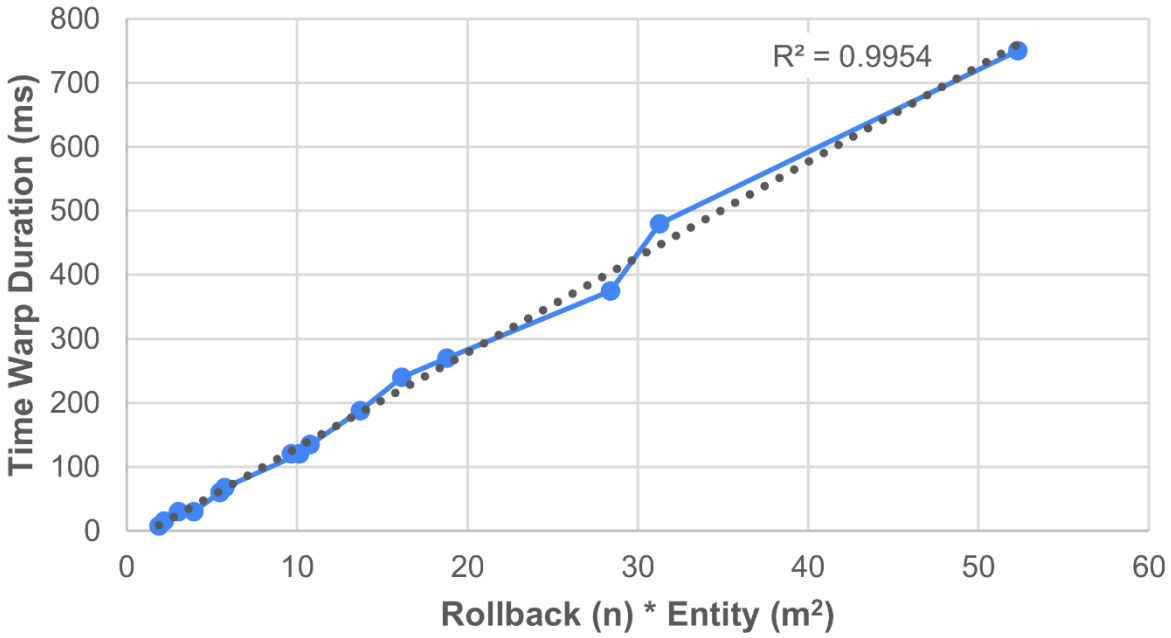


Figure 4.11. Graph of Time Warp Duration against the Predicted Rate of $n * m^2$

4.2.3 Networking Performance

To evaluate networking performance, we conducted the performance test without time warp, and used Wireshark to record the packets coming into and out of the. In Figure 4.12 graphs the amount of data sent per second against the number of AI. The data is separated by packet direction, so the blue line is inbound, server created packets, while the orange line is outbound, client created packets. In this graph client bitrate stays steady, while server bitrate increases linearly with number of AI.

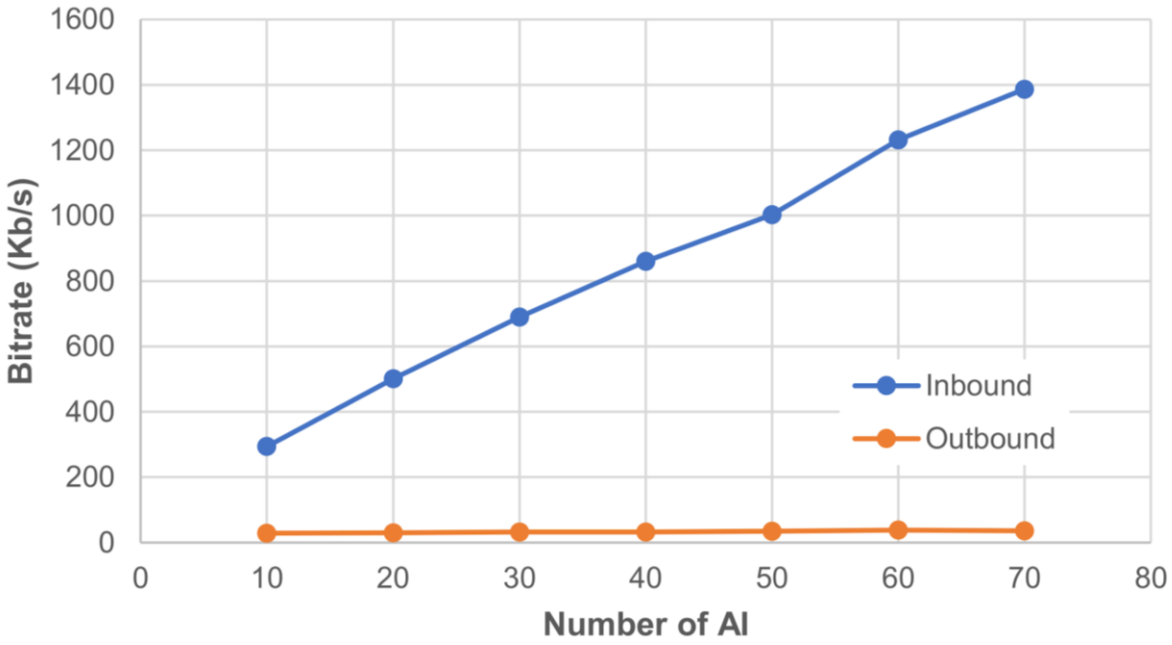


Figure 4.12. Graph of Inbound and Outbound Bitrates on Client

Figure 4.13, provides a cumulative distribution graph of the packet sizes. As with the previous graph, the blue line here represents inbound packets while the orange line shows outbound packets. From this graph, most of the clients outbound packets are relatively small. In comparison, the packets inbound from the server are bimodal, with many small packets and many large packets, but few in between.

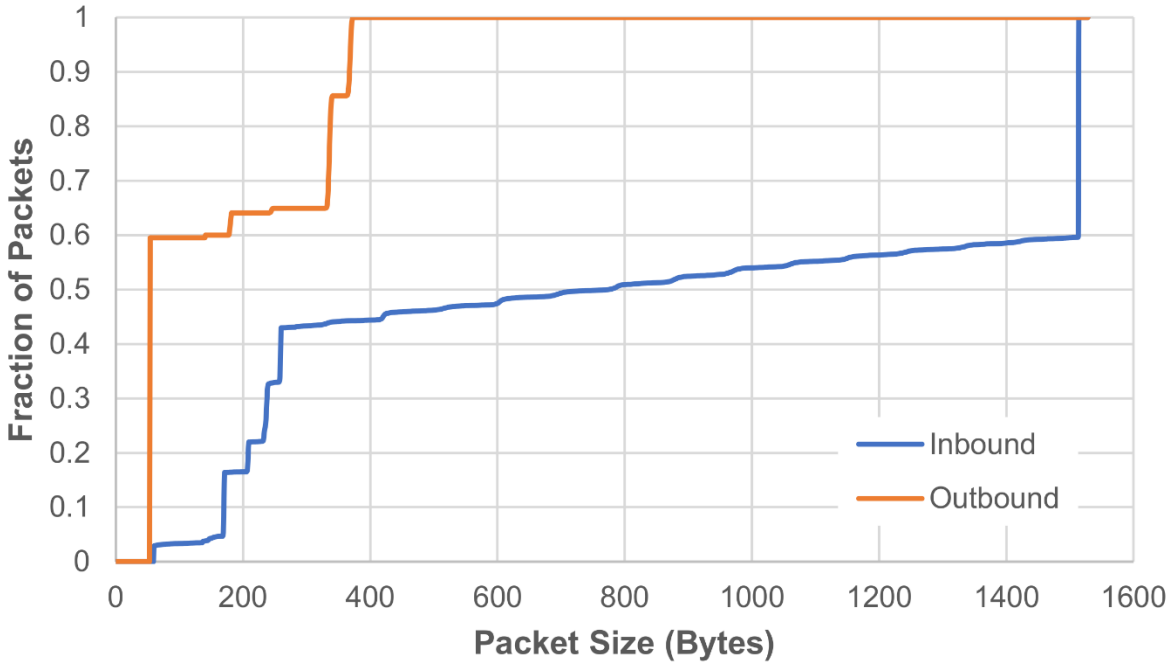


Figure 4.13. Cumulative Distribution Graph of Inbound and Outbound Packet Sizes on Client

5. Conclusion

Latency is an unavoidable obstacle in many systems, such as networked games. While it can be minimized in some cases, such as by playing a game over LAN, it is fundamentally impossible to avoid at larger distances. With that in mind, we set out in this paper to assess how best to mitigate the negative effects of latency by creating a networked game and implementing two latency compensation techniques: time warp and prediction. We then performed user studies in our networked game with 32 participants to analyze the impact of these different techniques on both their in-game performance and on their perception of the latency.

5.1. Analysis of the Impacts of Latency Compensation

Without latency compensation, player performance degraded in our game, and latency was quite noticeable. In particular, there was a steady decrease in the effectiveness of the hitscan weapon included in the game. Projectile-based weapons, however, remained relatively consistent across different latency values as long as both the shooting player and the target player had the same latency. We suspect that this is due to the symmetrical effects of the latency, these being that while aiming is more difficult, it is also more difficult to dodge out of the way of a projectile. By contrast, with the raygun, it is more difficult to aim with latency, but as the weapon cannot be dodged the other player's latency makes no impact on its accuracy.

5.1.1. Effects of Time Warp

In our testing, time warp did not make a significant impact on players' performance. Player accuracy with the raygun, the only weapon affected by time warp, followed the same downward trend with increasing latency both with time warp enabled and disabled. In addition, time warp is performance intensive, and imposes major restrictions on game design. Because

time warp needs to be able to rerun parts of the game and produce consistent results, it requires that the game be entirely deterministic. Factors like randomness must be carefully implemented so that they behave consistently when run again. This becomes even more difficult to handle when the initial state that the random effects start from may not be the same as it was the first time it was run, as a new event will have been inserted into the event queue. For these reasons, we do not believe that time warp is an effective latency compensation technique for a game such as ours.

5.1.2. Effects of Prediction

As with time warp, we did not notice a substantial improvement in player performance with prediction enabled, however player accuracy was consistently very slightly higher. While objective performance may not have increased considerably with prediction enabled, the latency was less noticeable to the players. This is likely because with prediction enabled, players get immediate feedback from their actions, such as movement. This can lessen the feeling of the game being sluggish without strictly improving the players' performance.

In comparison to time warp, prediction's impacts on the performance of the game itself are quite different. Whereas time warp puts a major strain on the server, each client handles prediction on its own. As, outside of graphics, the clients do not need to compute much, they can often handle the extra cost of performance. Even if they cannot handle it weaker clients could simply not run prediction, because prediction works on a player-by-player basis. As we found in our study, while this will make latency more noticeable for clients without prediction, it will not seriously impact their performance relative to players with prediction enabled. Because of this, we believe that prediction is worth implementing in a game such as ours for its positive impact on players' impressions on latency.

5.1.3. General Conclusion

We found that neither prediction nor time warp made a significant impact on players' performance in our game. Despite this, prediction did have a positive impact on players' perception of latency. Because of these findings, we believe that for games such as ours prediction is a more effective method of compensating for latency than time warp.

5.2. Future Work

While we could not conclude that in our game these latency compensation techniques had a significant impact on player performance, we also believe that our game may not have been the best suited to testing such techniques. We think that time warp in particular would be better tested with a different type of game, such as a first person shooter. As an example, in a first person shooter visibility is a major component of gameplay. Suppose that one player peeks out from behind a corner to shoot at another player. From the perspective of the player being shot at by the time they see the other player peek out from behind the corner, the shooting player may have already gotten back behind cover, making it impossible for the targeted player to react. With time warp, if they reacted and shot at the peeking player, the peeking player could be rolled back to their position outside of cover, and so the targeted player would be able to react. Such an interaction could not be accurately recreated in our game, as players cannot hide from one another. While a player cannot shoot another player who is behind cover, they can still see them, and so could react when they see the other player moving out of cover, but before they actually get out of cover.

In addition to this, our data on time warp was limited by our game design. Out of five weapons available in the game, only the raygun, a hitscan weapon, had time warp enabled.

Because of this, only up to one player at a time could even experience time warp. This caused our sample size of players who had time warp enabled for them to be considerably lower than that of players who did not use time warp. It is possible that time warp could have an impact on player performance, but with the amount of data we were able to collect, we were not able to find that impact.

References

Arif, Shabana. “GTA 5 Has Made More Money Than Any Film, Book or Game, Says Analyst.” IGN, IGN, 9 Apr. 2018, https://www.ign.com/articles/2018/04/09/gta-5-has-made-more-money-than-any-film-book-or-game-says-analyst.

Armitage, G. “An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3.” *IEEE International Conference on Networks, ICON*, IEEE Computer Society, 2003, pp. 137–41.

Armitage, Grenville, Mark Claypool, and Philip Branch. *Networking and Online Games : Understanding and Engineering Multiplayer Internet Games* . John Wiley & Sons, 2006.

Bernier, Yahn W. “Latency Compensating Methods in Client/Server In-Game Protocol Design and Optimization.” Valve Developer Community, Valve, 2001, https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization.

“BoxCollider.size.” Unity | Documentation, Unity Technologies, 16 Sept. 2019, https://docs.unity3d.com/ScriptReference/BoxCollider-size.html.

Gutwin, Carl, et al. "Revealing delay in collaborative environments." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2004.

Guulay, Biniam Gebregergs. *CheesePi: Measuring Home Network Performance Using Dedicated Hardware Devices*. 2015.

Hoffman, Chris. “How Latency Can Make Even Fast Internet Connections Feel Slow.” How-To Geek, How-To Geek, 11 July 2017, https://tinyurl.com/v8p4mx4

"image_xscale." GameMaker: Studio, YoYo Games Ltd, 2018, https://docs.yoyogames.com/source/dadiospice/002_reference/objects and instances/instances/instance properties/image_xscale.html.

Ivkovic, Zenja, et al. "Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games." *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, vol. 2015-, ACM, 2015, pp. 135–44, doi:10.1145/2702123.2702432.

Juckett, Ryan. "Rollback Networking in INVERSUS." *Gamasutra*, 8 Dec. 2016, https://www.gamasutra.com/blogs/RyanJuckett/20161208/287162/Rollback_Networking_in_INVERSUS.php.

de Almeida, Luis Fernando Kawabata, and Alan Salvany Felinto. "Evaluation of the Motion-Aware Adaptive Dead Reckoning Technique Under Different Network Latencies Applied in Multiplayer Games." *17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2018.

Kharitonov, Vasily. "Motion-Aware Adaptive Dead Reckoning Algorithm for Collaborative Virtual Environments." *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, ACM, 2012, pp. 255–61, doi:10.1145/2407516.2407577.

Lee, I., S. Kim, and B. Lee. "Geometrically Compensating Effect of End-to-End Latency in Moving-Target Selection Games." *Conference on Human Factors in Computing Systems*, New York, NY, May 2019, Association for Computing Machinery, 2019. 1–12.

Lee, Steven, and Rocky Chang. “Enhancing the Experience of Multiplayer Shooter Games via Advanced Lag Compensation.” *Proceedings of the 9th ACM Multimedia Systems Conference*, ACM, 2018, pp. 284–93, doi:10.1145/3204949.3204971.

Meseta. “A Diagram Showing How Network Latency Can Cause Disparate Gameplay between Players.” *Medium.com*, 14 Apr. 2018, https://medium.com/@meseta/netcode-concepts-part-1-introduction-ec5763fe458c.

Ng, Y-Shen. “Designing Fast-Action for the Internet.” *Gamasutra*, 5 Sept. 1997, www.gamasutra.com/view/feature/131635/designing_fastaction_for_the_.php.

Palant, Wladimir, Carsten Griwodz, and Pål Halvorsen. “Evaluating Dead Reckoning Variations with a Multi-Player Game Simulator.” *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM, 2006, pp. 1–6, doi:10.1145/1378191.1378196.

Pantel, Lothar, and Lars Wolf. “On the Impact of Delay on Real-Time Multiplayer Games.” *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM, 2002, pp. 23–29, doi:10.1145/507670.507674.

Pantel, Lothar, and Lars Wolf. “On the Suitability of Dead Reckoning Schemes for Games.” *Proceedings of the 1st Workshop on Network and System Support for Games*, ACM, 2002, pp. 79–84, doi:10.1145/566500.566512.

PlayOverwatch. “Developer Update | Let's Talk Netcode | Overwatch” *Youtube*, Commentary by Tim Ford, Philip Orwig, 4 Apr. 2016, https://youtu.be/vTH2ZPgYujQ

Quax, Peter, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. “Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and

Jitter on a Recent First Person Shooter Game." *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, 2004, pp. 152–56, doi:10.1145/1016540.1016557.

Raaen, Kjetil, Ragnhild Eg, and Carsten Griwodz. "Can Gamers Detect Cloud Delay?" *2014 13th Annual Workshop on Network and Systems Support for Games*. IEEE, 2014. 1–3. Web.

Sabet, S., Schmidt, S., Zadtootaghaj, S., Griwodz, C., & Moller, S. "Towards Applying Game Adaptation to Decrease the Impact of Delay on Quality of Experience." *IEEE International Symposium on Multimedia (ISM)*. IEEE, 2018. 114–121. Web.

"Source Multiplayer Networking." Valve Developer Community, Valve, https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

Sun, Jiawei. Drizzle: Design and Implementation of a Lightweight Cloud Game Engine with Latency Compensation. Master's Thesis Worcester Polytechnic Institute, 2017.

Wai-Kiu Lee, and Rocky K. C Chang. "Evaluation of Lag-Related Configurations in First-Person Shooter Games." In *International Workshop on Network and Systems Support for Games (NetGames)*, 2016-:1–3. IEEE, 2015.

Appendix A: Game Design Document.

Six Shooter [Working Title]
This game design document describes the details of a basic 2d top down multiplayer shooter.

We intend to have a simple game where 1-4 players can travel around a large map and shoot weapons at each other, until they run out of lives or the timer runs out. Each player has a certain amount of health, which they will not regenerate. When the timer ends, the player with the most points (kills) wins. Games should last around 1 minute in most scenarios. This can be adjusted as the game development continues. The game will be drawn with basic shapes, and the players will only be able to move in 8 directions, the cardinal directions and the directions halfway between them. Players will have the option to collect various weapons from the map, which will change how their projectile functions. In the default gamemode, players will all fight each other in a deathmatch. In team deathmatch, players will fight in a team to kill the opposing team. In kill confirmed, players will fight in a team to kill the other team, and collect a "kill confirmation" token from where the enemy died. The enemy will only lose a life once this token collected.

Core Goals:

- 1-4 players
- Map with basic barriers
- Players have health
    - No regeneration
- FFA deathmatch
- Respawning
    - Invincibility on rebirth

Art Assets Needed:

- Title Screen
- Heads up display
    - Score
    - Lives
    - Health
- Player
- Barriers
- Default projectile

Expanded Goals:

- Different weapon types
    - Hitscan weapon
        - One projectile shot at a limited rate
        - Immediately hits first enemy it would interact with
        - No travel speed
        - 3rd fastest fire rate
    - Projectile weapon (start weapon)
        - One projectile shot at a limited rate
        - Fastest travel speed
        - Fastest fire rate

    - Shotgun weapon
        - Multiple projectiles shot out at same time in a cone
        - 2nd Fastest travel speed
        - 2nd Fastest fire rate
    - Rocket launcher
        - One projectile shot at a time which explodes in an aoe effect on contact
        - 3rd Fastest travel speed
        - 3rd Fastest fire rate
- Allow users to discard current weapon to return to the start weapon
- Basic AI
    - AI simply travels randomly
    - Shoots at first enemy it sees
    - Has some randomness/delay in shooting to make it fair

Art Assets Needed:

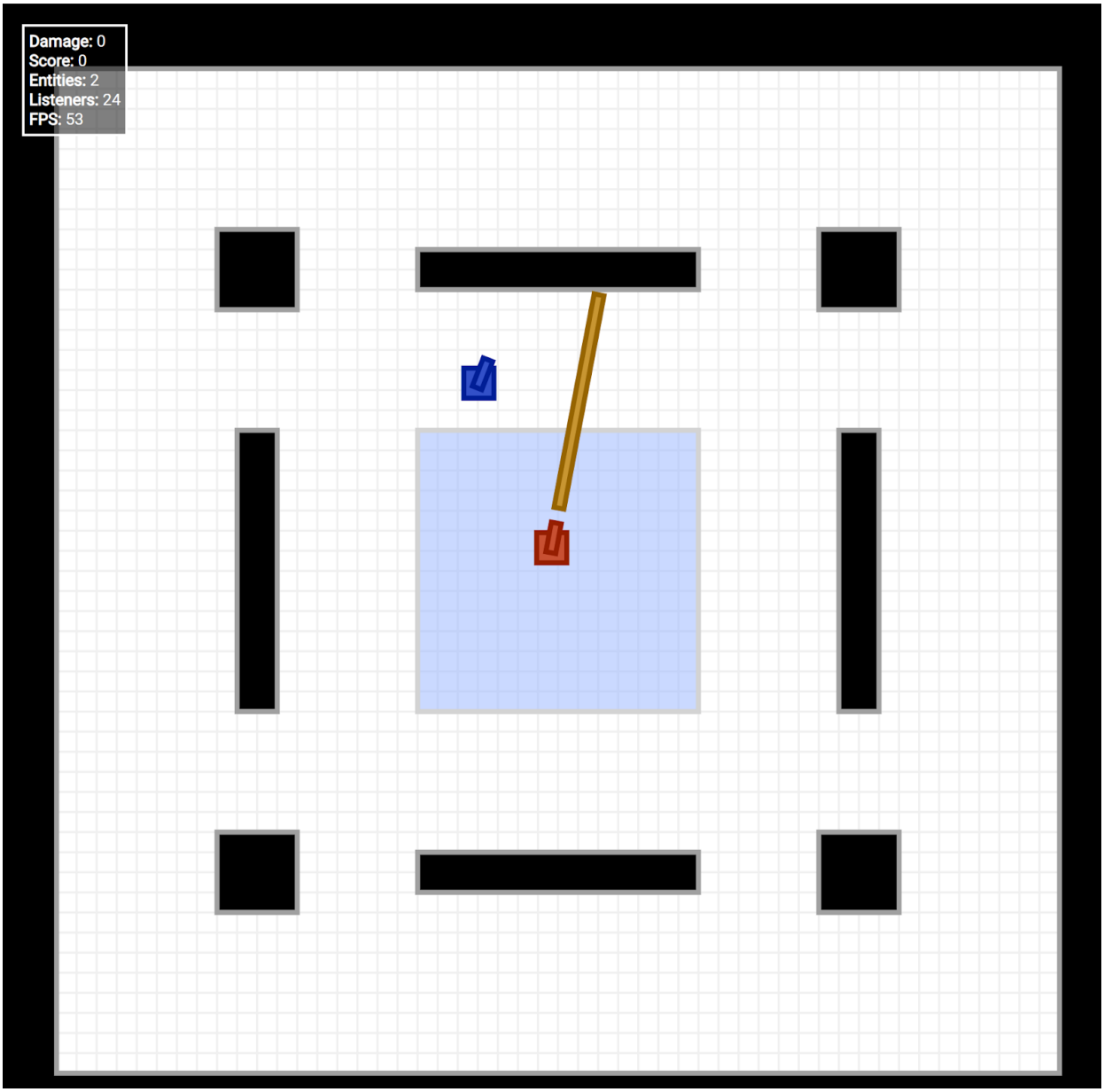
- Hitscan weapon
    - Sprite
    - Projectile
- Projectile weapon
    - Sprite
    - Projectile
- Shotgun weapon
    - Sprite
    - Projectile
- Rocket weapon identifier
    - Sprite
    - Projectile

Stretch Goals:

- Different Game modes
    - Team deathmatch
        - 2 teams attempt to wipe each other out before the time runs out
    - Kill Confirmed
        - 2 teams attempt to kill each other and collect “kill confirmed” tokens from where enemies died
        - Players only lose lives when their “kill confirmed” token is collected

Art Assets:

- Team colors
- “Kill confirmed” token


Damage: 0
Score: 0
Entities: 2
Listeners: 24
FPS: 53

Appendix B: Github Repo.

https://github.com/bwhetherington/dragonfly-js

Appendix C: Study Survey.

Determining the Effects of Network Latency on Video Game Players
Questionnaire

1) Participant Name:
_____

2) What gender do you Identify as?

_____

3) How old are you?

_____

4) How many hours a week do you play video games?

_____

5) Rate your skill with PC action games on a scale of 1-5, with 1 being unskilled and 5 being highly skilled:

_____

Appendix D: In Game Survey.