Impact of Latency on The Player in Online Games

Ian McNeil

Matthew Beyler

Advisor: Mark Claypool

Abstract

As online gaming grows in popularity, it becomes increasingly important to recognize and assess the impact of high latency on player performance and overall game satisfaction, as well as the efficacy of methods to deal with them. In this study, we created a system to test the impact of two popular latency compensation systems (time warp and prediction) at varying degrees of latency upon player performance and enjoyment of the game. It was found that both time warp and prediction improved player performance and that time warp was more effective than prediction.

Table of Contents

List of Figures

List of Tables

1 Introduction

The growing popularity of computer games and more specifically online multiplayer computer games has led to an interest in studying the effects of latency upon the users of these systems and the efficacy of methods of compensating for this latency. Latency can have a large effect on the end user's experience in online computer games. Latency causes the user's commands to be delayed so that by the time the command is received and executed by the server the game state has changed. The delay between the server and client also means that the user's view of the game world is out of date, hampering the user's ability to make proper decisions in the game. An example of this would be a user shooting at someone in a first person shooter and by the time the shot has been fired the intended target has gone around a corner. This delay not only has a negative impact on the user's performance, but also on the user's game experience.

Many different methods of compensating for this delay have been developed in order to mitigate the effect on gameplay. The effectiveness of these options can be measured by adjusting latency and measuring the quality of the resulting game play, either by some metric such as score or by asking for feedback about the user's experience. Some of these include prediction of future game states and executing user actions as if the action happened when sent by the user rather than when received by the server. By predicting future game states, the client can display a more accurate representation of the game state on the server, allowing the

user to make more informed decisions. For example, if a projectile is shot, its movement is predictable so the client can display its location properly even if the current game state is out of date. This can be applied to more complex movement but there is the possibility for some error in which case warping can occur, in which objects teleport to their correct locations regardless of where they previously appeared to be.

Another method is time warp, in which commands are executed as if they were received when they were sent by the client, and then resolve this with the current game state. This allows the user's actions to seem like they were done exactly when they were commanded, however it can also lead to other players being affected retroactively. A good example of this is a player who moves around a corner later being shot because another player issued the command to shoot them before they turned the corner.

In this study we created a simple game-like system, implement multiple different methods of latency compensation, and compare their performance across different amounts of latency by comparing the scores. In this system the user is asked to click on multiple targets which are being moved by a bot. The score is defined to be the number of targets hit in a set amount of time.

1.1 Hypotheses

H1. The implementation of latency compensation techniques in games will improve the performance of players.

The use of techniques like time warp and prediction will minimize the effects of network latency on the player's performance, leading to the player being less impacted by the delay between sending a command and the execution of that command.

H2. The efficacy of these latency compensation techniques will decrease at higher levels of latency.

This is due to the fact that as the delay between the command and response increases, the chance that the game state has changed in a manner differing from what the client predicted becomes higher. If the game state has changed in a manner different from what it was predicted to be, the techniques become ineffective.

H3. Time warp will allow players to play the game as if there was no latency on their end.

Since time warp affects another player's gamestate based off the first player's gamestate regardless of network latency, this player's actions will feel as if there is no delay between when an action is taken and when the game recognizes it. Even though the game is still hosted online, the effects of latency will be nearly negated for that player.

H4. Latency compensation will have a more noticeable effect if game targets are smaller, move faster, or both.

Small targets require more precise focus which are more susceptible to changes of latency. Similarly, fast-moving targets require quicker response times, which are affected heavily by high latency. As such, challenges involving these types of targets will notice a larger benefit from the compensation techniques involving slower, larger targets.

H5. Latency jitter will have a more noticeable effect on quality of gameplay than latency.

Although large amounts of latency will have a negative impact on a player's performance, a player can become used to it and begin to negate its impact (for example, a player in a shooter leading shots to compensate for the delay between the command being issued and being executed). With jitter, however, latency changes frequently, making each action respond in a different amount of time. This makes it much more difficult to predict performance and respond accordingly.

H6. The player's enjoyment of the game will start to decrease before any reduction of performance.

Small amounts of latency, especially if they stay at a consistent level, can be compensated for by the player. However, even if that player's performance remains relatively consistent, the strain of having to account for latency with every action taken will leave the player frustrated with the performance of the game.

1.2 Latency Compensation Techniques

While there are many ways to compensate for latency, this study looks at two major methods: prediction and time warp. Prediction is when latency is hidden by predicting the location and actions of game objects so their position on the client is more accurate to the position on the server between updates. Time warp is when the server rolls back the game state to when an action was sent by a client and acts as if the action was taken then. This new game state is then reconciled with the current game state.

1.2.1 Prediction

Prediction is a compensation technique in which an object in the game state's behavior is predicted and simulated when a network connection is interrupted. In its most basic implementation, prediction takes an object's current velocity and direction of movement and has the object continue with that pattern of movement to give the client an estimation of the object's actual position in the up-to-date game state. This allows the player to take action upon a game state that is relatively close to the "official" game state, which helps to minimize the effects of higher latency on the player's performance in an online game.

One of the major advantages of prediction is that it allows the game flow to feel uninterrupted despite latency issues. Prediction estimates where a target is going to be based on its current movement, allowing the player's actions to roughly correlate with the official game state on the server. This allows that player to make more accurate actions than he would without prediction while under the effects of lag. Without prediction, game objects may come

to a complete stop or have severe jittering in their movement, leading to a loss of flow and reduced player satisfaction with the game. Prediction allows objects' movements to continue uninterrupted even with latency, preventing interruptions in gaming from the client's side and leading to increased user satisfaction. Another advantage of prediction is that it allows the player to act upon an estimation of the updated game state even with latency problems.

Figure 1 compares prediction working correctly, on the left, versus no compensation, on the right. The black circle shows the location on the client-side while the red circle shows the object's actual location on the server. In this case prediction of the object's movements between server updates allows the client's view to more accurately show the actual gamestate.

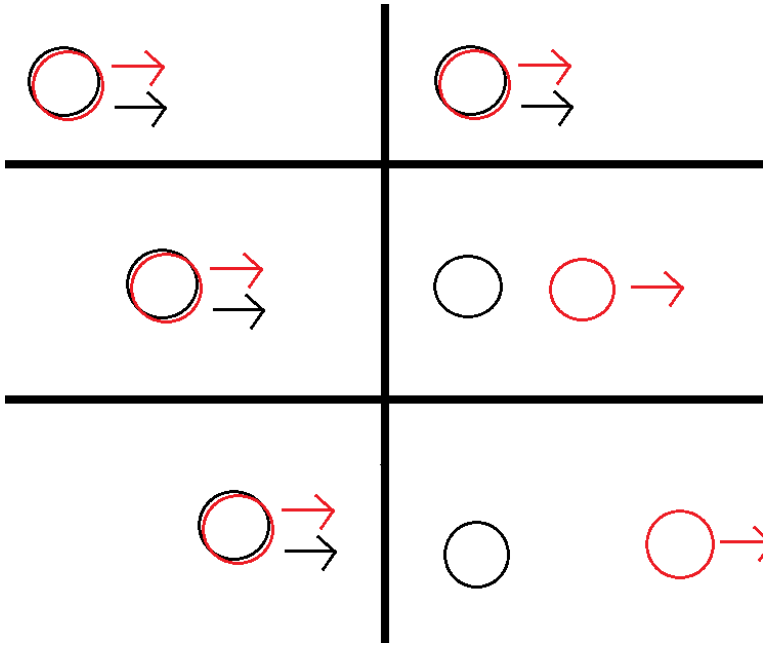

Figure 1: A situation where prediction improves player performance. The black circle represents the player's view of the game state, while the red circle represents the actual position of the circle on the server. On the left, the client circle continues moving in the velocity it had been moving in on the server state, allowing the user to make accurate actions despite their latency problems. On the right, the circle simply remains still while waiting for a server response, putting it out of sync with the official game state.

Prediction has several downsides, however. One major issue with prediction is that it is substantially less effective with more complicated forms of movement. Game objects that frequently change direction or move in a particularly unpredictable way are much more difficult to simulate the movement of, and may lead to the player's game state being substantially different than the actual game state it is supposed to represent. This leads to the player attempting to act on a game state that is not representative of the official state, leading to poor performance and player frustration. This is particularly notable in genres like real-time strategy and first person shooters, which involve players making many small movements and changing direction frequently.

Figure 2 shows an instance of when prediction fails, on the left, versus no compensation, on the right. The black circle shows the players view of the gamestate and the red circle shows the actual gamestate on the server. In this example the object changes directions between server updates, causing the prediction to approximate the object's location incorrectly.

Another issue with prediction is that it becomes much less effective at compensating for latency as the latency reaches particularly high levels. As the delay between an action and response increases, it becomes increasingly likely that the object whose behavior is being predicted has acted in a matter differing from what the compensation technique has estimated. As such, the larger the latency, the less likely it is that the player is acting on a reasonable estimation of the actual game state.

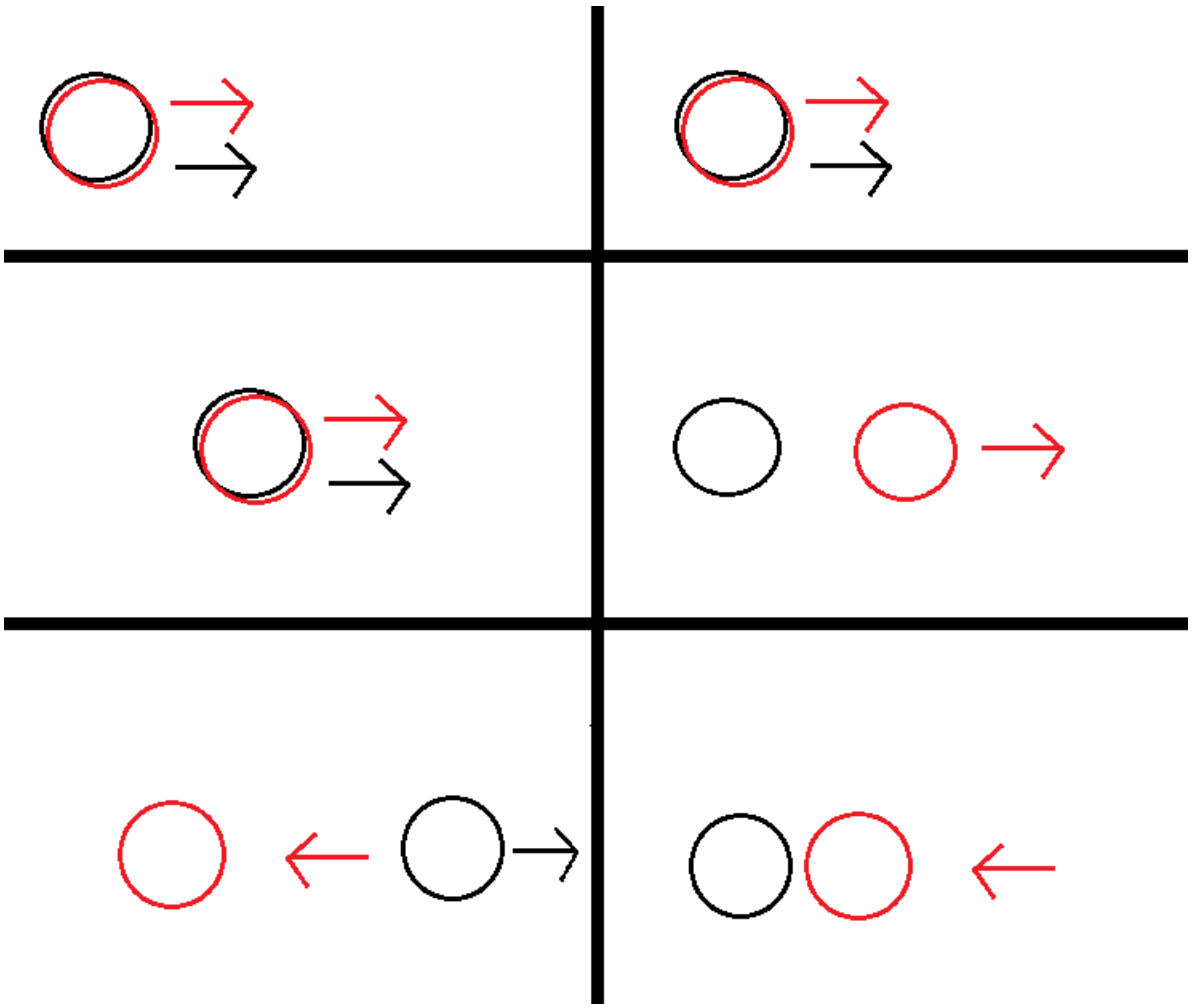

Figure 2-A situation where compensation works poorly. The black circle represents the player's view of the game state, while the red circle represents the actual position of the circle on the server. In this situation, the target changes direction midway through; the player client has no way of knowing this, however, and continues to move in the original direction, leaving the representation of the target farther off than if no compensation technique at all had been applied.

1.2.2 Time Warp

Time warp is a compensation technique in which the accuracy of a player's actions are recorded in relation to the game state on the client, rather than in comparison to the "official" game state recorded by the server. This allows a player who is experiencing latency issues to continue to play the game based off of the client's representation of the current game state, helping to minimize the effect of latency on performance and allowing the player's game to feel

much more responsive. The other players' game states are then retroactively updated to align with the changes that player made to the game state while lagging.

Time warp's major advantage is that the game state the player sees is always the one that his or her actions are recorded against, regardless of latency. For example, a player of a shooter with extreme lag may have to lead shots without the use of a compensation technique, leading to a decrease in both performance and player satisfaction. With time warp, however, the player can shoot where the target is on screen and have the hits register if they are aimed well, even if the server's location of the target does not line up with where it is represented on the client. This allows the player to continue to perform well regardless of latency. It also benefits from the client's side at any latency—unlike prediction, which drops off substantially in efficacy at higher latency levels, time warp will be just as efficient at higher latencies. Figure 3 shows how time warp works, the player model is the location of the player on the server, the red wireframe is the location of the player's hit box on the client side when the shot was fired, and the blue wireframe is the rolled back location of the player's hit box when hit detection is done.

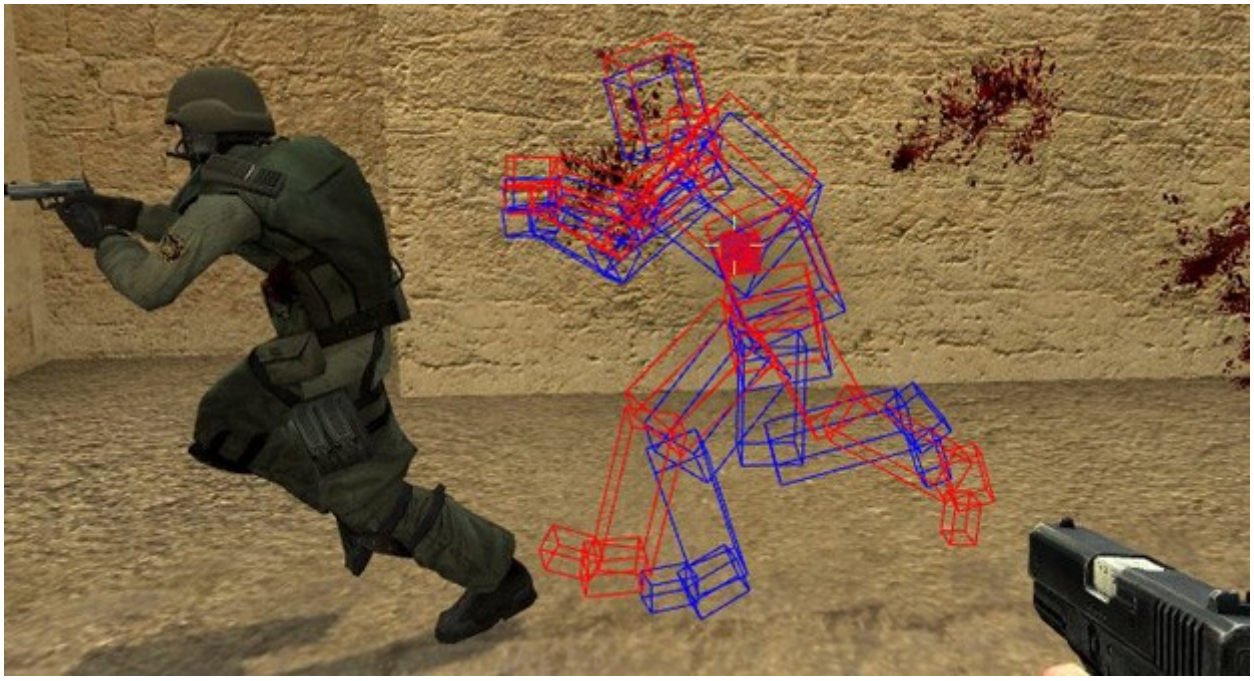

Figure 3-- This is a screenshot from Valve's Counter-Strike: Source demonstrating how Time Warp works. The red wireframe is where the lagging player saw the opponent to the left (representing the official server state). When the command is registered, however, the game "rewinds" to where the opponent was when the command was fired (blue wireframe). (Lag Compensation, developer.valvesoftware.com)

There are several downsides to time warp, however. One of the biggest downsides is the impact that time warp has on other players on the server. Time warp's updated game state may cause a player who was not experiencing latency problems to have their games state changed in a negative fashion. For instance, a player in a shooter who had gotten away on his client would end up dead if a lagging player shot him, since time warp would reconcile that player's game state with the server. This can lead to frustration among players whose latency is at more manageable levels, as well as provide opportunities for abuse (for instance, a player may intentionally induce lag spikes to stop an opponent from moving temporarily, allowing an easier kill).

2 Previous Studies

The growing popularity of computer games and more specifically online multiplayer computer games has lead to an interest in studying the effects of latency upon the player in these systems and how effective these methods of compensating for this latency are.

Mark and Kajal Claypool have put forward a method of classifying actions in online games by how much latency effects them (Claypool, 2010). They suggest two axes by which to measure an action, precision, or the degree of accuracy the action requires, and deadline, the amount of time required to complete the action. As these measures increase the action is more affected by latency. They continued this attempt to determine what situations are most affected by latency by measuring the effect of latency under different viewpoints. They were able to conclude that viewpoint had a large effect on how latency affected game play (Claypool, 2009).

One method for examining latency is to look at commercial games and to induce latency and measure the effects. James Nichols and Mark Claypool measured the number of yards per attempt at running under different amounts of latency in order to determine the efficacy of Madden Football's latency compensation techniques (Nichols, 2004). In addition they measured some network traffic information such as bitrates. Beigbeder, et. al. have looked at Unreal Tournament 2003 and the effects of latency on different types of actions within that game, as well as looking at the effect of latency and packet loss on the network traffic of the game (Beigbeder, 2004).

Another method is create a simple game-like system and implement latency compensation and examine that system. Buchheit created a simple game with delay compensation implemented and conducted player trials and used score to measure the effect of turning the delay compensation on and off. This study showed that latency compensations techniques could be quite effective and this general approach could be used to compare different techniques as well (Buchheit, 2004).

3 Program Description

A program was created which approximates a video game by being based around a core type of video game interaction, while being simple enough to reduce confounding factors and ease creation. The program created focuses on the use of the mouse in computer games to select video game elements by clicking on them. The program is two dimensional and a circular target is moved around the window by a computer player, or bot, and the user must click on it, this destroys the clicked target, awards the user points, and spawns a new target. This continues until a time limit is reached, at which point the score is used to determine performance. The score awarded for each target depends on the location of the click relative to the center of the target, such that clicking closer to the center yields more points; and more targets clicked means more points. This way both the speed and precision of the user's clicks determine performance.

The program is implemented using a server-client structure in which a central server controls the game state and facilitates communication between the clients. The user is a client and attempts to click on targets, while a bot spawned by the server is another client that moves the targets. The server and bot were implemented as java applications and the server spawns bots as they are needed, while the client is a java applet hosted on a website which also handles the collection of responses to pre- and post-test questions. Communication between the server and clients is done through UDP, or User Datagram Protocol, a common method of sending data over the internet which is commonly used for video games. Packets of a known

size and format are sent between the clients and server. Latency is simulated on the server using WIPFW (Windows Internet Protocol Fire Wall) and DummyNet, these tools allow latency to be simulated for incoming and outgoing packets according to their source and destination. This means that each client can have different latencies compared to each other and depending on whether the packet is incoming and outgoing.

Packets going from the client to the server contain commands from the user for the server to execute. The possible commands include a command to start the game, a click, and the user disconnecting. Packets from the server to the clients include updates on the gamestate. Possible updates are the target moving, a target being destroyed, a missed shot, a target being created, and the game ending.

3.1 Server

The server was implemented as a java application which listens for connections at a known IP address and port number. When a client connects to the server, a new game state structure is created which stores information about the status of the game and the clients connected to it. In addition, the server spawns a new bot as its own process which then connects to the server as well and the server sets up WIPFW and DummyNet for both the user and the bot. WIPFW and DummyNet are the tools used to delay incoming and outgoing packets between the server and both the client and the bot. Once the game has started the server listens for messages from the clients, processes them, and sends out changes in the game state to both clients. The server also maintains a timer for the end of the game, at which point the clients are notified that the game is over and further actions are ignored.

An important function of the server is to maintain the correct latency between the clients and the server despite unknown base latency from the user's connection to the server and despite this latency's varying over time. In order to do this the server must first attempt to measure the current latency between client and server. A packet containing the current system time of the server is sent to each client which then sends back an identical packet. When the server receives this response it measures the round-trip latency using the current system time assuming symmetrical latency and halves it to find the latency. This latency is compared to the target latency and the time that WIPFW is delaying packets is adjusted accordingly. This process is done every second in order to maintain the target latency.

The server logs all commands received from the clients, all updates to the game state that it sends out to the clients, the settings used in the game, and measured user latencies. These logs are named base on trial number and placed in folders named after the user's identifier as assigned by the website test bed. Using these logs the entire game can be replayed exactly as it happened, except for the location of the user's mouse between clicks. This level of detail to the logs allows for many data measurements, even ones which may not have been expected when the program was created.

3.2 Bot

The bot is also implemented as a java application; it is a separate process started by the server when a new user connects. This means that unlike the server which is a single process which handles every game running on the server, there is one bot process for each game ongoing. The bot process is given a game id by the server when the process is started, which

the bot then passes back to the server when it connects, so that the server knows which game the bot is intended to join. Once the game starts, the bot issues a create target command and then issues move commands for this target at regular intervals until the target is destroyed, at which time the bot issues another create target command. This continues until the bot receives an end game command from the server, at which time the bot disconnects and the bot process ends. The regular move commands from the bot act as the impetus for the game state changing, as the server does not change anything on its own, it only responds to commands sent from the clients.

The bot system was designed in order to allow flexible behavior of the bot when creating and moving targets. This allowed for many different types of behavior to be easily made and tested so that the best system could be chosen. Three main movement types were created for the bots and tested in pilot tests to see which yielded the most useful results. All three behaviors created targets the same way, by placing them randomly on the board. They varied in how the target was moved: completely random movement, straight line movement, and approximate straight line movement. Completely random movement chooses a random direction and distance every update period, which yielded a target that moved randomly in a relatively small area. The straight line option picked a random spot on the window and moved straight towards it until it reached its destination, at which point a new destination was chosen. The approximate straight line option picked a random destination like the straight line option, but instead of moving straight towards it, it picked a random small angle and moved that amount off of the straight line to its destination. This yielded a target that moved randomly, but with a general direction that it traveled. After pilot studies, the straight line movement

option was chosen because it interacted most accurately with the latency compensation methods implemented. Clicking on a moving enemy target is a common action in many genres of video game including trying to shoot a player in a first person shooter and clicking to attack an enemy unit in a real time strategy. Also, by having the target move in a straight line the player no longer needs to predict the targets movements, making latency a larger factor in whether or not the player hits the target.

3.3 Client

The client was implemented as a java applet to be deployed onto a web server which runs the test bed, controlling settings for each trial and recording pre- and post-test questions. When the webpage with the applet is loaded the applet attempts to connect to the server, at which point the server creates the game and the bot is created and connects. The user is presented with a prompt to click when they wish to begin the game. When the client receives this click it sends a start command to the server which then sends out a command to start the game to both clients if a bot is also connected. Once the game has begun the client listens for updates to the game state and updates the graphics accordingly; when the user clicks a command is sent to the server indicating the location on the screen that was clicked and when, in game time, the command was issued. The result of the click is then sent back to the clients and appropriate graphics are drawn and the game state updated. When the end game command is received from the server, the client clears the target and other graphics from the screen and presents the user's score and a prompt to click when they are ready to move on to the post-test questions.

The head-up display of the client consists entirely of a bar at the top of the screen which contains the user's score, targets hit, and the amount of time remaining in the trial. These statistics were chosen to be shown in order to motivate the user to maximize their score by both clicking quickly and accurately. The time not only shows them how much time they have left, but also puts them under a deadline, so they know they must score all the can in their limited time. The number of targets hit is to motivate them to click quickly, and the score motivates them to be accurate in addition to quick.

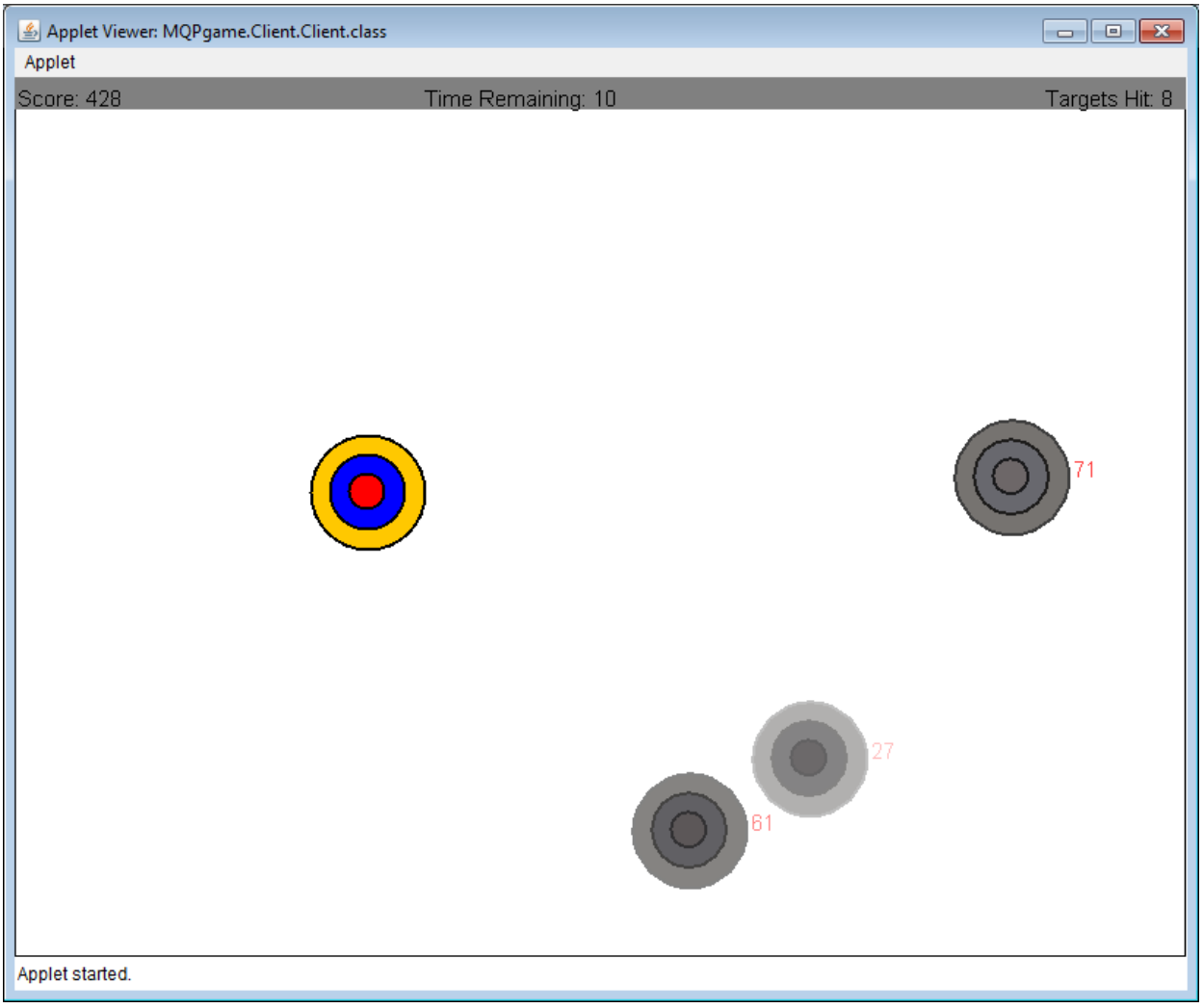

Figure 4: Screenshot of a game in progress

The graphics of the game were also designed to motivate the user to try their best when playing the game. Figure 4 shows a screenshot of the game in progress which shows the heads-up display and the target graphics. The targets are concentric circles of changing colors to suggest to the user that it is best to click near the middle and the background is white to provide good contrast with the colored targets. The game area is outlined with a black line to make it very clear what the limits of the target's position are, this is needed because the white background of the game window is the same as the background of the webpage. When the user clicks there is feedback regardless of if they hit or not. If the click misses than a miss graphic is shown in the location of the miss. The miss graphic is a small black circle which fades over a short period of time, this graphic is significantly smaller than the targets and only black so there is no confusion between the two. If a target is hit then the target becomes grayscale and fades out over a short period of time, like the miss graphic. The target becomes grayscale so that there is not confusion over action and inactive targets. In addition to this destruction graphic the score that the user got for that destruction is shown in red next to the fading target. This gives the user immediate and clear feedback as to the quality of the hit, which motivates them to attempt to hit the center of the target to get the highest possible score.

3.4 Time Warp

Time warp is implemented on the server using a tree map which holds the changes to the game state sent out by the server and the time they were sent out as the keys. Whenever the server sends out updates to the game state and time warp is enabled, it stores the changes and removes changes that were made more than five seconds earlier. This is done to keep the number of changes stored from growing indefinitely. When a shot is received from the client

the game state is changed to how it was when the shot was sent, as long as it was within five seconds. This new game state is then used to check for hit or miss and appropriate changes to the original game state are sent out.

3.5 Prediction

Prediction is implemented on the client by finding distance moved between the targets current position and the target's previous position and assuming that the target will continue to move in the same way. This was chosen over other prediction techniques, such as a moving average, which take into account more known positions of the targets because the movement pattern that was chosen for the bot changes very suddenly so a prediction method that has a longer memory would take much longer to correct itself after a change. In addition, because the target moves in a straight line between changes in direction the simple average used to predict is accurate between moves.

The ability for the client to predict the location of targets in the future would be less useful if the client only redrew the screen when it received an update from the server. Therefore the client was changed to instead redraw the screen as quickly as possible, this way the client's ability to predict is used to make the target's movements smoother and more accurate. In order to limit the amount of unneeded work the client does, a maximum number of frames rendered per second was set for the client at 150. This is well over the refresh rates of common monitors so additional frames per second would not add to the players experience and the resource use of the client is limited.

3.6 Data Rates

Data being sent between the server and the clients is in response to events. When a command is sent from a client to the server, the server then responses with an update to the game state. However, the data rate can be estimated based on the number of clicks by user and the number of times the bot moved the target. The size of commands from the bot and client to the server was 183 bytes and the size of the game state updates from server to client and bot was 173 bytes. The number of times the bot moved the target was constant, occurring every 50 milliseconds for the duration of the 20 second long trial, resulting in commands being sent from the bot to the server 400 times and 400 game state updates being sent from the server to the both the client and the bot. The number of commands sent from the client to the server is dependent on the number of times the user clicked, which could be between zero and the maximum number of clicks a human is capable of making in 20 seconds. The minimum average number of clicks for a trial was 21.6 in timewarp with 300ms of latency, the maximum average number of clicks for a trial was 31.3 for prediction with 200ms of latency, and the overall average across all trials was 26.1.

The average data rates from bot to server, client to server, and server to client were then calculated. The data rate from bot to server was 3.66 kb/s. The data rate from client to server was 0kb/s if the user never clicked, .20kb/s for timewarp at 300ms of latency, .29kb/s for prediction at 200ms of latency, and .24kb/s for the overall average. The data rate from server to client and bot would be the same and was calculated to be 3.46 kb/s if the user never clicked, 3.65 kb/s for timewarp at 300ms of latency, 3.73kb/s at prediction at 200ms of latency, and 3.69kb/s for the overall average.

4 Study Description

The test bed the study was conducted in was deployed as a website so that the effort required by subjects was kept minimal. This low barrier to entry was created to increase the number of subjects. The study was composed of four questions before the testing began, followed by ten trials in which the user played the game we created under different settings. After each trial the user was asked two post-test questions about the trial they had just completed. The study was advertised through friends, family, and WPI mailing lists. The full test was completed by 75 people.

4.1 Test bed

The test bed was implemented as an apache web server hosting a website which the user connects to and is guided through the tests. The index page of the website describes the test to the user and tells them the requirements of the test, shown in Figure 5. When the user hits "Continue", unique identifier is assigned which is used to connect the logs of each trial completed along with the answers provided to the pre- and post-test questions. The user is then asked the four pre-test questions and the website logs the answers in the folder specified by the identifier. Next, the user is presented with the first trial and the identifier is given to the applet and logs for that trial are placed in the same folder as the question answers. When the trial is complete the user is automatically moved to the next page in which they are asked that trial's post-test questions, which are logged in the same folder. The trial completion and post-test questions are repeated nine more times with different settings and then the user is presented with a completion and thank you screen.

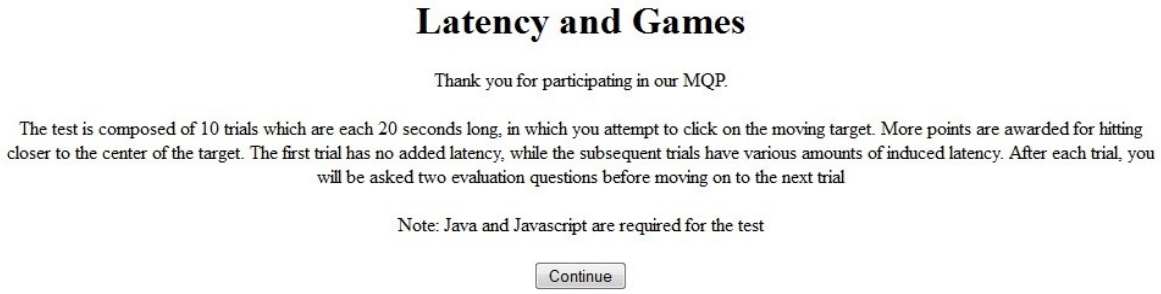

Figure 5: Starting Page of Website

4.2 Program Properties

4.2.1 Independent Variables

Two separate variables were changed in this study: the compensation technique used and the latency. There are three options for the compensation technique: prediction, time warp, and none. There was one trial with each of these settings for every level of latency that was selected. Latencies were selected in order to give enough different levels to keep the total length of the test low while showing the full range of possible latencies. The latencies which were decided upon were 100, 200, and 300. 100 was chosen to be the lowest level because the lower bound on possible latencies is the user's latency to the server, so if a very low latency was chosen it is possible that some user's would have higher base latency and having the correct latency would be impossible. In addition, 100 is a common latency and is considered to be a low latency for many applications. 300 was chosen to be the highest latency because 300 is considered a high latency and is on the upper bounds of what is commonly seen in online video games. Three different levels of latency were chosen because it is important to keep the total

test length low in order to keep the interest of the subjects, if the test length is too high then the subjects will grow bored and may leave or stop trying, which yields lower quality data.

4.2.2 Constants

Many aspects of the program needed to be made constant at values that are appropriate and help generate useful data. These variables are the size of the window, the size of the target, the speed of the target, the length of the game, and how often the bot makes a move. The window size of the game is important because it affects how large an area the target can move in and because of this how far the user will need to move their mouse. This means that a larger window size is better and more accurate to normal games which are often played in full screen. However, the window size has to be smaller than the monitor's resolution so that the entire window can be displayed at once. Taking into account these considerations, the window's size was decided to be 800 by 600 pixels. The majority of modern monitors support higher resolutions than this so it is very likely that the entire window will fit on screen at once, but this is still large for a game embedded in a webpage.

Target size and speed and their relation to the overall window size are important settings due to their impact on the difficulty of the game; a small fast target is much harder to click on than a large slow moving target. The difficulty of the game needs to be such that changes in the compensation technique make a measurable difference in the score of the user. If the difficulty is too high then scores will be so low in perfect conditions that when latency is induced there will be little difference for compensation techniques to fix, because the score could not go any lower. If difficulty is too low then latency will once again make little impact

because the score is always near max, even with latency. The target size was chosen to be 40 pixels after pilot tests, which is one twentieth of the width and one fifteenth of the height, which provides a reasonably sized target with a lot of room to move around the window. The speed was chosen to be 300 pixels per second, which means the target can move from the top to the bottom of the screen in two seconds.

The bot timer period is also an important setting that was made constant for all trials. This number controls how often the bot makes a move, increasing it leads to smoother movement of the target and more network traffic. If this setting is too high, however, then there is no time for prediction to come into play, as there will be no frames rendered on the client between updates from the server. Too low on the other hand means that without prediction enabled the game appears choppy. Through pilot tests it was chosen to set the bot timer period to 50 milliseconds, so that the bot moves the target 20 times per second. This is enough that the game still appears smooth without prediction, but the increase in smoothness from prediction still makes a noticeable difference.

The length of each trial is an extremely important variable because it strongly affects the total length of the test. If the test is too long than subjects may lose interest and give bad data or not even complete the test, however it also important to maximize the amount of data collected. It was decided that there was to be 10 trials, a control and nine sets of settings, so the total length of the test is ten times the length of a trial plus the amount of time it takes the subject to answer the pre-test questions and the post-test questions for each trial. In addition to the importance of the total test length, the length of each trial also is important because if

the trials are too long then the game can get monotonous, but the breaks of answering post-test questions and the changes in settings can mitigate this, so it is important that the individual trial lengths be kept short for this reason as well. In the end it was decided to set the individual trial lengths to 20 seconds, which was found to be a good middle ground during pilot tests between collecting as much data as possible while not tiring the user.

4.3 Pre-Test Questions

Before the test began the subjects were asked to answer four pre-test questions to measure their preexisting experience with video games and latency. These four questions are shown in Figure 6 which shows the screen presented to the subjects before the trials began. The questions are answered using a simple five point scale in order to give the subject the ability to describe their answer properly while not being so large a scale as to introduce unnecessary specificity which would cause subjects answers to become more arbitrary. For example, if the subject was asked to give on a scale from 1 to 100, 1 being never and 100 being very often, how much they play video games, there would be very little difference between 50 and 51. The cause of the choosing one or the other would largely be arbitrary difference by the subject, not related to how often they play video games.

The first question, "How often do you play video games?", is intended to determine preexisting experience and skill with the style of interaction in the program. The second question, "How often do you play online video games?", is a more specific version of the first question whose intent is not to get at their skill or experience in the base game play of the program, but to gauge their experience with dealing with latency in such an environment.

Someone who plays online video games often will be more adept at accounting for latency then someone who does not. The third question, "How would you describe the quality of your Internet connection?", follows up on this, if their Internet connection is bad then they have more experience dealing with high latency then someone who has a high quality internet connection. In addition, this helps to gauge the quality of the data collected from the subject, if they have unstable internet then it may be impossible to have the program be properly controlled due to their base latency being too high or the connection to the server being inconsistent. Finally, the fourth question, "How much of an impact do you believe latency has on game performance?", measures the subjects attitude towards latency and its effects on video games.

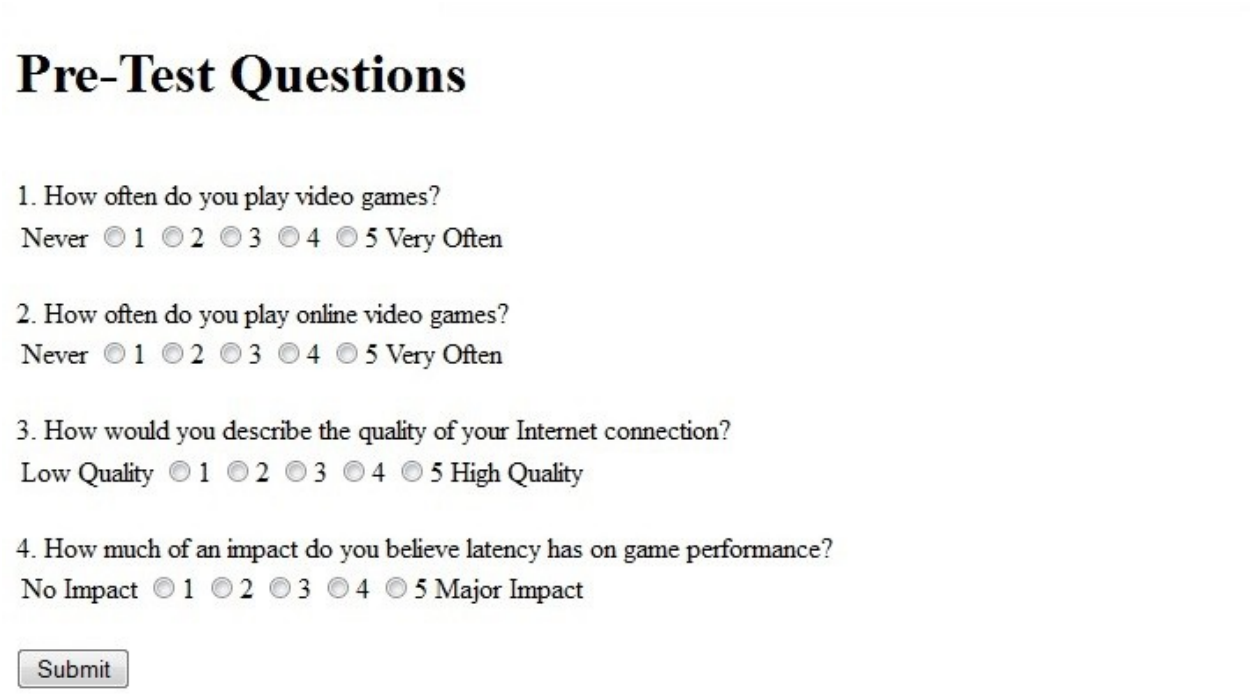

Figure 6: Webpage to collect answers to pre-test questions

4.4 Post-Test Questions

The subject was asked to answer two questions after each trial was completed. These questions used the same five point scale as the pre-test questions to measure their perception of the quality of each group of settings. The quest ions and the screen presented to the subjects are given in Figure 7. The first question, "Rate the quality of the gameplay for this trial", measures perception of the quality. Their perception of the quality is important because the goal of latency compensation is to increase the degree to which the user enjoys the video game, not just player performance. Similarly the second question, "Rate your enjoyment of the trial", ascertains experience playing the game under the circumstances in the trial, which is an important part of measuring the quality of the latency compensation.

Figure 7: Webpage to collect answers to post-test questions

5 Analysis

5.1 Pre-Test Questions

Before the test began, the subjects were asked four questions to judge their experience with video games and their attitude toward latency. The answers were given on a five point scale with one being low and five being high; the questions are given below followed by the mean, median, and mode of the responses.

Question 1: How often do you play video games?

Question 2: How often do you play online video games?

Question 3: How would you describe the quality of your Internet connection?

Question 4: How much of an impact do you believe latency has on game performance?

	Question 1	Question 2	Question 3	Question 4
Mean	3.96	3.67	3.91	4.19
Median	4	4	4	4
Mode	5	5	4	5

Table 1: Results of Pre-Test Questions

The mean and median of all the questions are clustered around four with Question 2 being the lowest at 3.67 and Question 4 being the highest at 4.19. The mode for three questions is five and the mean is lower than the median for three questions, showing that some outlying low responses dragged down the overall high responses. This means that in general the subjects who participated in our study had large amounts of experience with video games, specifically online video games, had high quality Internet, and believed that latency was an important factor in the game performance for online games.

5.2 Score

The score awarded for each target hit was based on the location of the hit relative to the center of the target, where the center of the target is worth 100 points and the very edge of the target is worth 10 points and it is scaled linearly between. This rewards both the speed and the accuracy of the subject's shots. The average score is plotted against the latency for each compensation technique setting in Figure 8.

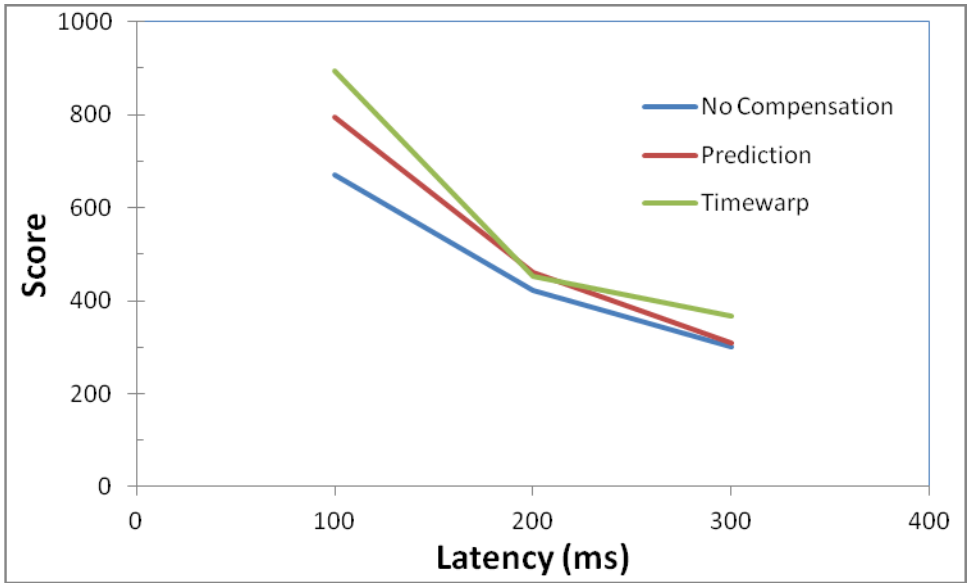

Figure 8 : Average score per trial vs. Latency for each compensation technique

A two tailed t-test was then conducted for each latency level to determine the probability that each compensation technique came from the same population as the trials with no compensation technique. The results are shown below in Table 2. A two tailed t-test is a test of statistical significance that compares two sample populations. The result is the probability that the two samples come from the same population. The two samples compared in this study are the results with no latency compensation and the results with one of the

compensation techniques. If the probability is sufficiently low, the difference is said to be statistically significant.

	100ms	200ms	300ms
Prediction	0.01	0.30	0.78
Timewarp	0.00	0.49	0.06

Table 2: t-test results for score

The graph shows that on average the compensation techniques improved performance, but the effect diminished as the latency level increased. The t-tests showed that this difference was only significant for 100ms latency with prediction, and time warp for both 100ms and 300ms.

5.3 Targets Hit

The number of targets hit during a trial was recorded in order to measure the speed with which the targets were shot separately from their accuracy. The average number of targets hit is graphed below in Figure 9 against the latency level for each compensation technique setting. The statistical significance of the measurements was then found by conducting a two tailed t-test, the results of which are given in Table 3.

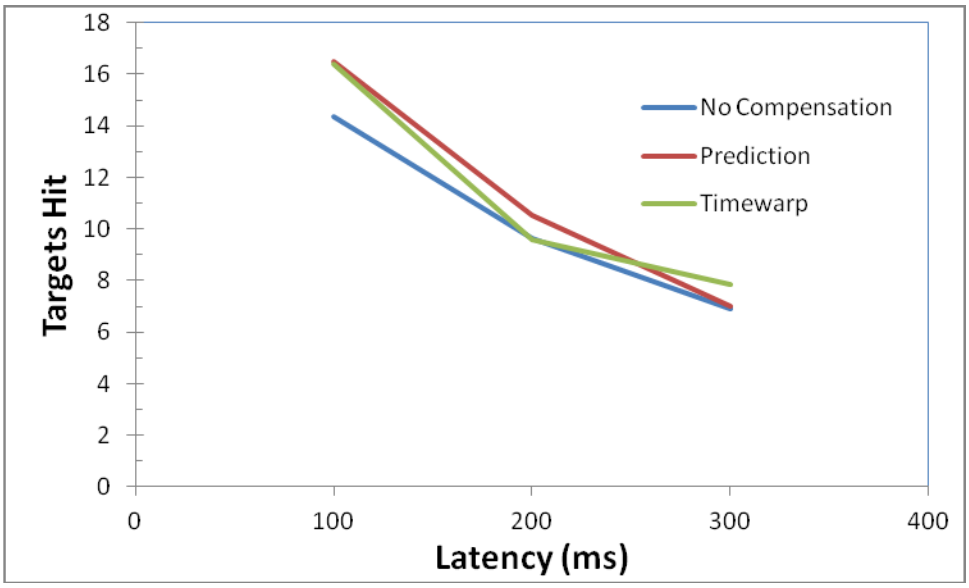

Figure 9: Average targets hit per trial vs. Latency for each compensation technique

	100ms	200ms	300ms
Prediction	0.02	0.25	0.87
Timewarp	0.02	0.95	0.16

Table 3: t-test results for targets hit

The graph shows that with compensation techniques the subjects hit more targets per trial on average than without any compensation techniques at all latency levels. However, the calculated significance shows that only a significant difference is shown for 100ms of latency.

5.4 Accuracy

The subject's accuracy was calculated by dividing the number of targets hit by the total number of shots attempted. This measures the amount of shots that did not hit a target at all and therefore both the accuracy of the target's visual position on the screen and amount of care the subject put into each shot. The average accuracy is shown below in Figure 10 plotted against the amount of latency.

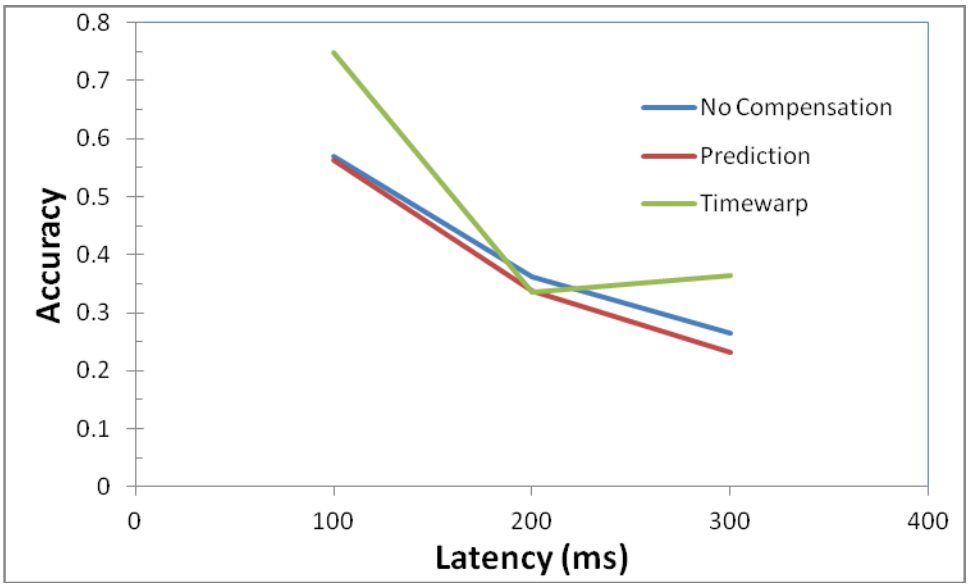

Figure 10: Average accuracy vs. Latency for each compensation technique

The statistical significance of these findings was computed by carrying out a two tailed t-test to calculate the probability that the trials with compensation came from the same population as the trials without compensation. The results of these tests are shown in Table 4.

	100ms	200ms	300ms
Prediction	0.82	0.41	0.24
Timewarp	0.00	0.45	0.01

Table 4: t-test results for Accuracy

The results show that for accuracy, prediction does not have a noticeable effect, showing no statistically significant differences at any amount of latency. Time warp shows improved accuracy at 100ms and 300ms of latency, but not at 200ms. The lack of effect of compensation techniques on accuracy may be due to the fact that the test imposed no penalties for missing the target completely, only reduced score for hitting the target on the edges.

5.5 Score Per Target

The score value of a target is determined by how close to the center of the target the user's cursor was when the target was clicked. This metric only factors shots that connect, so it provides a way of analyzing the spread of shots that were not complete misses. The results are shown in Figure 11, below, and the statistical significance as calculated by a two tailed t-test is given in Table 5.

At the lowest latency (100ms), time warp demonstrated a notably higher average score per shot than both prediction and no compensation. Prediction also had higher scores than the no-compensation trials, demonstrating that both techniques help to compensate for performance penalties caused by latency. At the intermediate latency (200ms), however, prediction's average score per target dropped below that of the no compensation trials. This demonstrates that the effectiveness of prediction drops off substantially at higher latency levels due to a larger window in which the object could change direction. Time warp performed better than the no-compensation trials at 200ms, although both groups dropped steadily.

At the highest latency (300ms), the average score per target for time warp and no-compensation actually increased, with time warp score still staying above no-compensation. This may be due to gameplay being impacted enough that players are forced to learn to manually predict where the target will actually be.

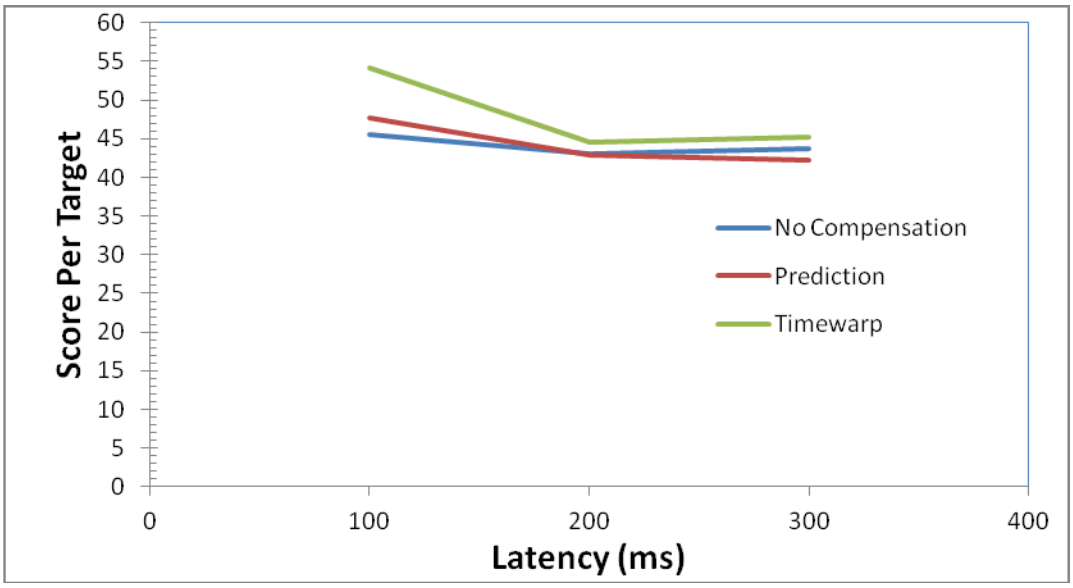

Figure 11: Average Score per target vs. Latency for each compensation technique

	100ms	200ms	300ms
Prediction	0.05	0.86	0.46
Timewarp	0.00	0.32	0.50

Table 5: t-test results for score per target

5.6 Post-Test Questions

The post-test analysis for the study consisted of a two-question questionnaire. The first question read “Rate the quality of the gameplay for this trial”, while the second read “Rate your enjoyment of the game for this trial”. Both questions were given a range of 1 to 5 as a choice of answer, with 1 being the worst and 5 being the best. This questionnaire was presented at the end of every round of gameplay other than the zero-latency control round. The mean results are shown in Table 6 and Table 7.

For all 3 groups (no compensation, prediction, and timewarp) both the quality of gameplay and enjoyment of the game went down as latency went up. At 100ms, both timewarp and prediction outperformed no compensation in both categories, with timewarp

having a slight edge over prediction. This demonstrates that both compensation techniques have a notable impact on player enjoyment at lower/moderate latencies. However, when moved up to 200ms, both the quality and enjoyment ratings for the 3 groups dropped off significantly. Prediction and timewarp were both barely above no compensation for quality of gameplay, and were actually very slightly lower for enjoyment. This demonstrates that the efficacy of prediction drops off significantly as latency increases. This observation is reinforced by noting that at the highest latency, the average quality of gameplay and enjoyment of the prediction trial is lower than that of the trial utilizing no compensation techniques.

Time warp, on the other hand, drops off less than the other two groups, retaining the highest average rating of both gameplay quality and enjoyment at the highest latency. This demonstrates that although it is still impacted by increasing latency, time warp does a notable job of maintaining a sense of playability.

	100ms	200ms	300ms
No Compensation	2.84	2.03	1.71
Prediction	3.14	2.09	1.63
Timewarp	3.38	2.15	1.85

Table 6: Mean results of Post-Test Questions on gameplay quality

	100ms	200ms	300ms
No Compensation	2.51	1.97	1.68
Prediction	2.86	1.97	1.63
Timewarp	2.93	1.97	1.89

Table 7: Mean results of Post-Test Questions on enjoyment

6 Conclusion

Latency can have a large impact on the player's experience in online computer games. The delay between messages being sent and received between server and client leads to actions being taken based on inaccurate game states. The effect of latency can, however, be mitigated using latency compensation techniques.

In this study, two latency compensation techniques were implemented and player performance and game quality were measured at multiple levels of latency. Player performance was measured by capturing gameplay statistics during the trials, such as score, accuracy, and number of targets hit. Game quality and player experience data was collected by asking the player to rate their experience after each trial. This data was used to determine the efficacy of the two implemented latency compensation techniques: prediction and time warp.

Latency compensation techniques improve the performance of players in games where these techniques are implemented. Both time warp and prediction led to higher overall scores at all tested latency levels than the no-compensation client, demonstrating that their presence reduces the impact of latency on player performance. This supports Hypothesis 1, that the latency compensation techniques would improve user performance.

The efficacy of the tested compensation techniques dropped off at higher levels of latency. Although both techniques still outperformed the no-compensation client, the overall benefit was significantly lower at higher latency levels, supporting Hypothesis 2. Time warp outperformed prediction in all categories except targets hit. This demonstrates that time warp is more useful in compensating performance than prediction. Prediction's performance

dropped off much more than the performance of time warp, eventually ending up below the no-compensation client for some trials.

Hypothesis 3 was not able to be properly tested, due to the fact that our test bed did not allow for testing with no latency, because the test was conducted over the internet. Hypothesis 4 and 5 were also not tested, because the target size and speed was constant in our trials and jitter was not implemented, however these factors could be investigated in further studies. Hypothesis 6 could not be tested due to the number of latency levels that were tested. While reduced enjoyment was measured at higher latencies, it was unclear if this decrease occurred before or after the performance decrease.

Further research can be conducted with different models of game interaction which may be influenced by latency in different ways and further explore the strengths and weaknesses of the compensation techniques. Additional research can be done on the gameplay model used in this study by adding a punishment for completely missing the in order to provide additional incentive for using care when aiming shots. In addition further research could be done with additional techniques for latency compensation.

Bibliography

Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003, In *Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, Portland, OR, USA, September 2004. Online at: http://www.cs.wpi.edu/~claypool/papers/ut2003/

Buchheit, R. F. (2004). Delay Compensation in Networked Computer Games. Cleveland: Case Western Reserve University.

Mark Claypool and Kajal Claypool. Perspectives, Frame Rates and Resolutions: It's all in the Game, In *Proceedings of the 4th ACM International Conference on the Foundations of Digital Games (FDG)*, Florida, USA, April 2009. Online at: http://www.cs.wpi.edu/~claypool/papers/perspective/

Mark Claypool and Kajal Claypool. Latency Can Kill: Precision and Deadline in Online Games, In *Proceedings of the First ACM Multimedia Systems Conference (MMSys)*, (*Invited paper*), Scottsdale, Arizona, USA, February 2010. Online at: http://www.cs.wpi.edu/~claypool/papers/precision-deadline-mmsys/

Lag Compensation. Retrieved Febuary 2, 2014 from https://developer.valvesoftware.com/w/images/c/ca/Lag_compensation.jpg

James Nichols and Mark Claypool. The Effects of Latency on Online Madden NFL Football, In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Kinsale, County Cork, Ireland, June 16-18, 2004. Online at: http://www.cs.wpi.edu/~claypool/papers/madden/