LAG COMPENSATION IN FIRST-PERSON SHOOTER GAMES

LEE WAI KIU

MPhil

The Hong Kong Polytechnic University

2019

The Hong Kong Polytechnic University
Department of Computing

Lag Compensation in First-Person Shooter Games

LEE Wai Kiu

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Philosophy

April 2018

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

LEE Wai Kiu (Name of Student)

Abstract

Multiplayer games are important constituents of video game sales every year. While all multiplayer games are affected by network latency, or lag, the extent of the effect varies across different genres. Games with a first-person perspective, such as FPS and racing games, have been found to be the most sensitive to lag [1]. Sports and RPG games are less sensitive but not as less as omnipresent games like real-time strategy and simulation games.

It is not uncommon to have players with high network latency in a public multiplayer session. In shooter games, lag compensation is used to help mitigate lag for high-latency players. Upon receiving a shot, the server rolls all other players back in time according to the lag of the shooter. We refer to this method as traditional lag compensation (TLC). The drawback of TLC is that it introduces inconsistencies we call "shot behind covers" (SBC) to whomever receiving the shot. A player suffers from an instance of SBC when he is shot shortly after taking a cover, which according to that player, should not have happened because the cover should have broken the line of sight between him and the shooter.

Recently a few games try to tackle SBC by imposing limits on lag compensation. Players who violate the restrictions, such as having a latency higher than a certain threshold, are not lag compensated. This approach, however, can be an overkill. As we will show in Chapter 7, most of the time shooters do not cause SBC even

when the limits are broken.

In this dissertation, we propose a novel lag compensation algorithm which we name advanced lag compensation (ALC). ALC ameliorates SBC while retaining the benefits of lag compensation. The fundamental idea of ALC is that the eligibility for lag compensation depends on each shot but not the shooter. A shot is lag compensated if and only if it does not cause SBC regardless of properties such as the latency of the shooter. A highly lagged player can continue to benefit from lag compensation as long as his shot does not cause SBC to the victim. The core of ALC is the algorithm for detecting SBC, which to the best of our knowledge, is the first of its kind. Not only does this algorithm form the basis for ALC, it also allows us to objectively measure the frequency of SBC in a match regardless of which lag compensation algorithm is used. In the past, the SBC frequencies could be inferred only from subjective opinions of the players. With the SBC detection algorithm, we are now able to perform objective assessment of the effectiveness of ALC in reducing SBC. Our results show that ALC significantly reduces the number of actual and perceived SBC ($p < .05$) while achieving similar hit registration accuracy ($p = .158$) and responsiveness ($p = .18$) as TLC.

We also study how SBC changes with movement speeds and the amount of lag. These two parameters directly affect lag compensation because it rolls players back in time according to the lag of the shooter. A player is rolled back further in distance if either the shooter's lag or the player's movement speed increases. It may be easier to spot an instance of SBC if it occurs when the player is further behind the cover. In this dissertation, we show how movement speeds and lag individually, and together, affect the perception of SBC, and how they in turn impact the perceived fairness of the match. Our results also suggest that the

limits used by the few recent games to tackle SBC are reasonable despite their approaches being an overkill. Other useful insights provided by this particular study include what game modes should be used in similar future experiments.

To improve lag compensation, we have to understand what influences player perception of SBC and how that in turn affects their experience. In this dissertation, we conduct an investigation on the effects of common gameplay mechanics and features on the perception of SBC. Because these elements are irrelevant to lag compensation, those that are found to significantly affect how SBC is perceived have to be adjusted before subsequent experiments on the problem of SBC are carried out. Our results show no significant difference in the perceived fairness between different settings for all tested mechanics and features. This means that in the rest of our experiments, we can configure those elements to behave as they do in commercial games.

In order to perform all the aforementioned experiments, we had spent approximately 10 person-months to build a multiplayer first-person shooter game named LAGCOM from scratch using Unreal Engine 4. This provides us the highest and necessary controls over the game that would not have been possible with commercial games. We can customize the implementation of any mechanism in the game to suit our experiments' needs. In fact, some mechanisms and features are implemented differently from commercial games for experiment purposes. For example, we have implemented ALC in LAGCOM to evaluate its performance. Building our own custom game also allows us to include elements that are designed specifically to facilitate experiments in LAGCOM. The game is open source and available on GitHub (`https://github.com/stevenlwk/LAGCOM`). Researchers working on any problem about shooter games can obtain and modify

the project to their needs.

The problem of SBC prompts the suggestion of abandoning lag compensation entirely. In this dissertation, we show that the lack of lag compensation significantly decreases the mean shooting accuracy. We argue that a better solution should therefore be improving the TLC algorithm instead of eliminating it altogether. Limiting the distance players can be rolled back is a good starting point. Eventually, lag compensation should move away from using indirect indicators such as a player's latency, and start deciding based on whether a player is/will be shot behind covers or not.

Publications Arising from the Thesis

- W. K. Lee and R. K. C. Chang. Evaluation of lag-related configurations in first-person shooter games. In *2015 International Workshop on Network and Systems Support for Games (NetGames)*, pages 1–3, 2015.

- W. K. Lee and R. K. C. Chang. On "shot around a corner" in first-person shooter games. In *2017 15th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, June 2017.

- W. K. Lee and R. K. C. Chang. Enhancing the experience of multiplayer shooter games via advanced lag compensation. In *ACM Multimedia Systems Conference*. ACM, 2018.

Acknowledgements

First and foremost I would like to thank my chief supervisor Prof. Rocky K. C. Chang for his advices and guidances on my research and beyond. I appreciate all the time and funding he has contributed to make my MPhil experience productive and fulfilling. Over the years during my research, I had come to realize that this is not to be taken for granted. I am especially grateful for Rocky's patience with me taking the time to finish building the game used in the experiments. During my study, I also had the honor of being the teaching assistant of two of Rocky's courses. I want to thank him for giving me the freedom of authoring some of the teaching materials. It had been a rewarding and fruitful teaching experience.

I am thankful to all members of the Internet Infrastructure and Security Research Laboratory whom I had shared the laboratory with: Anson Kwan, Renee Chang, Ricky Mok, Star Poon, Steven Chien, Toby Lam, Weichao Li and Yan To Lam for their advices and friendship, as well as all of the chatters we have shared.

I would also like to express my gratitude to all the players who had participated in my experiments. Special thanks to recurring participants whom I also have the pleasure to call friends for putting up with some of the bugs in my game and answering the post-game questionnaires seriously. Here I would like to share a short anecdote. At one of the play sessions, the game was broken and players

could shoot through walls. I had to fix the bug live in the computer lab while they were playing *Little Fighter 2*. In retrospect, I would have never discovered the bug were it not for the players. This is why you have QA testers for video games.

From my observation, more and more players in recent years have started caring about game networking. This probably has to do with the proliferation of eSports and the general growth of competitiveness in video games. I want to express my appreciation of all the people who have shared on the internet their meticulous analysis of the networking in games. They help educate players, raise awareness of some long-standing issues and encourage developers to push the envelope.

Last but certainly not least, I am eternally grateful and indebted to my parents, who have indulged me my passion for video games, for all their love and support. I especially want to thank my exceptional mother for taking great care of me.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Glossary

Advanced lag compensation (ALC) A novel lag compensation method that mitigates the problem of SBC while retaining the benefits of TLC.

Conditional lag compensation (CLC) TLC with limits imposed. Players exceeding those limits are not lag compensated.

First-person shooter (FPS) A video game genre that involves gunfights in a first-person perspective.

Round-trip times (RTT) The amount of time it takes for a packet to be sent from one point to another and return.

Shot behind covers (SBC) An inconsistency introduced by TLC in which a player is shot even after taking covers which should have blocked the line of sight between him and the shooter.

Team deathmatch (TDM) A common game mode in FPS games that involves players competing in teams. The goal is to have the highest number

of kills as a team before the match ends.

Traditional lag compensation (TLC) A method to mitigate the impacts of lag by rolling players back in time before deciding the outcome of a shot.

Chapter 1 Introduction

Multiplayer has been increasingly significant in video games. Most of the top-selling games in recent years include a multiplayer component. Some of them are even multiplayer-only games, a case that is especially true for free-to-play games. Bernier believes that having a robust networked multiplayer is vital to a game's success and longevity [5]. In terms of genre, first-person shooter (FPS) games has a substantial presence in the video game market. In 2016, FPS games took the top two positions of the best-selling games in the US [6]. They were joined by two other FPS games in the top 10 list [6].

All video games are affected by numerous sources of latency. Examples includes input delays arisen from devices such as mouses and monitors, as well as processing delays caused by servers handling commands from the players. In multiplayer games, however, network latency, or simply lag, usually contributes to the majority of the problems [7, 8]. Lag is inevitable because it takes time for data packets to travel between clients and servers. FPS games are very sensitive to lag [9, 1]. Studies have found that lags within 100ms are enough to impact player performance and experience [7, 1, 10, 11].

A multiplayer game usually employs various lag compensating methods together to ease the effects of lag. In shooter games, one of the most common techniques is lag compensation. Without it, when a lagged player fires at a moving opponent,

his shot would likely miss the target. This is because by the time the server knows about the player's action, the opponent could be in a different location. It means that in order to actually hit the target, players need to lead their shots. Not only does this make the gameplay unrealistic, it is also difficult for players to know how much to aim ahead.

In the traditional approach to lag compensation, the server keeps a history of all players' states such as locations and rotations. When the server receives an event about a player firing a shot, it first rolls all players back in time for a period equivalent to half of the shooter's round-trip latency. The shot is then evaluated before all player states are restored. The server essentially determines the outcome of a shot using the game world as seen by the shooter. These are all done only on the server's side so players do not actually see themselves being rolled back and forth in time. We refer this algorithm to as *traditional lag compensation* (TLC). TLC eliminates the need for players to lead their shots[1], at the expense of introducing inconsistencies in some situations. Some games are said to perform better with TLC despite of such inconsistencies [5]. Evidence supporting those claims, however, is scarce, and there lacks statistical comparisons between player performance with and without such technique. TLC favors the shooter, because the server essentially bases its verdicts on the game world as seen by the shooters when they fire the shots.

One of the inconsistencies introduced by TLC is called "shot around a corner" [5]. It describes situations in which a player is shot after he has just taken cover. A more appropriate name for this inconsistency should therefore be *"shot behind covers"* (SBC). SBC can happen to a player when he is shot by a sufficiently

[1] Aiming ahead of the opponent according to his movement

lagged player. SBC can happen quite frequently in FPS games because it is generally considered more strategically advantageous to take covers and fire at spotted opponents while peeking over, rather than running across an open field risking being seen. Imagine a sufficiently lagged (high-ping) player A firing at a less lagged (low-ping) player B who is dashing into a cover. TLC moves player B back in time and places him outside of the cover, resulting in a hit at player B. Player B experiences SBC in this case because in his view, he has already taken cover before being hit by player A. This example illustrates that TLC can be unfair to low-ping players. Despite having a batter network connection, they may still suffer from SBC if they are shot by high-ping players. Although there are legitimate reasons for some players to have a high network latency, low-ping players should not be compromised because of factors uncontrollable by them. We need an empirical study to quantify the benefits of having lag compensation in shooter games in order to justify its existence.

Although lag compensation is by no means a new invention, SBC still exists in many recent, modern games, including even some AAA[2] titles at launch [13, 14, 15]. In particular, the issues of lag compensation caused the launch fiasco of Battlefield 4 [16, 17] and the development of extra content was halted at one point [18].

Recently there are a few games (e.g. Battlefield 4, Overwatch, Call of Duty: Infinite Warfare) attempting to mitigate SBC by using what we call *conditional lag compensation* (CLC). CLC is basically the same as TLC but with conditions. Players exceeding some limit(s)–a common one being their round-trip latency– would no longer be lag compensated by the servers. The reasons of choosing

[2]Games considered AAA have "the highest development budgets" [12]

certain values as limits, however, are not well understood. CLC in some games also has its ping limit changed periodically as developers are figuring out the best way to configure the ping limit. Take Battlefield 4 for example, it uses a variable called `FrameHistoryTime` (FHT) as the limit. The way it works is explained in Section 2.2. FHT was originally set to 1.5 and later changed to 1 [19]. It then changed again to 0.125 before reverting to 1 [20]. Eventually Battlefield 4 settles on using two FHT values: 1 for vehicles and 0.125 for everything else [21]. The need to tweak the limits is one of the major reasons why CLC is not as popular as it should be. We need to investigate and possibly verify some of the limits used in order to encourage more developers to adopt CLC over TLC in their games.

Although CLC is better than TLC because it ameliorates the problem of SBC, it can produce false positives. CLC forces high-ping players to permanently fall back on leading the targets in some servers when most of their shots actually do not cause SBC. In this thesis, we will explore the possibility of coming up with an even better lag compensation algorithm.

Studies on lag compensation, especially improving it, in shooter games, and SBC are scarce. We believe this is generally because the drawbacks of TLC had not been exposed as prominently as it was in Battlefield 4. Bernier is the only one who explains the mechanisms of lag compensation and its problems in details [5]. Pfeifer suggests mitigating SBC by limiting the server to accept commands only up to a certain period in the past [22]. This is similar to CLC with ping limits. There have been, however, no studies on the appropriate values for CLC limits nor the problems of CLC.

1.1 Problem Statements

This study addresses several key issues on lag compensation in FPS games:

1. Does TLC improve player performance in FPS games?
   TLC eliminates the need for players to lead their shots, at the expense of causing SBC to less lagged players. We need to ensure that lag compensation does significantly improve the performance of sufficiently lagged players in FPS games before we can justify the need to enable and improve upon it. We use shooting accuracy as the metric for measuring player performance. It would be worth it to improve TLC in FPS games only if it significantly increases the shooting accuracy of sufficiently lagged players.

2. How do game mechanics and features affect the perception of SBC?
   We want to simulate all game mechanics and features in our experiments as closely as those in commercial products. Some of them, however, are irrelevant to lag compensation but affect player perception of SBC. We therefore have to identify them before studying SBC and control their settings during experiments.

3. How does player perception of SBC change with different round-trip times (RTTs) and movement speeds?
   The perception of SBC depends on how far the player being shot at is rolled back by TLC, i.e. the rollback distance, which in turn depends on the shooter's RTT and the victim's movement speed. CLC therefore uses either one or both of these two parameters as the limit. By understanding

the effects of RTTs and movement speeds on the perception of SBC, we can explain the values used in commercial games.

4. How do we improve lag compensation?
   Can we come up with a lag compensation algorithm better than both TLC and CLC? It should be able to mitigate SBC with adverse effects less extensive than depriving some players of lag compensation. It would be a good idea to stop lag compensating a player only when his shot(s) causes SBC.

1.2 Contributions

In this dissertation, we address several key research issues concerning lag compensation and the induced SBC in FPS games. We also propose a new approach to mitigate the problem of SBC:

1. This is the first empirical study of lag compensation and the induced SBC in FPS games. We have collected both objective and subjective measurements from human players in our experiments. The former includes shooting accuracy and numbers of SBC, which indicate player performance and SBC frequencies respectively. The latter includes questionnaires and numbers of SBC reported by players during matches. Questions such as those that ask for players' opinions on the game's hit detection accuracy and fairness reflect their perception of the game. All these measurements ultimately help us compare different lag compensation algorithms in terms of their

effectiveness and fairness.

2. We have built an open-source multiplayer FPS game for most of our experiments. To the best of our knowledge, it is the first FPS game specifically designed to study lag compensation and the induced SBC. Not only does the game include many features and mechanics commonly present in commercial FPS games, it also contains elements that aim to facilitate subjective measurements and experiments related to latency in FPS games. Researchers investigating issues on shooter games can obtain the code on GitHub and modify it to suit their needs. Details of the game are presented in Chapter 5.

3. We use our custom game to investigate the effects of 6 common FPS game mechanics and features, which are irrelevant to lag compensation, on the perception of SBC. They are the field of view, hit indicator, mini-map, ping time display, rate of fire and time-to-kill. Their details are presented in Chapter 6. Our results show that their influences are not significant. This implies that for experiments on SBC, researchers can safely configure those elements to behave as they normally do in commercial games without worrying about interfering with the results.

4. Two variables that directly affect SBC are the shooter's latency and the victim's movement speed. This is because TLC rolls players back in time according to the shooter's latency and the faster a player moves, the further he is rolled back in distance. We find out how they affect the perception of SBC. A few recent games attempt to mitigate the problem of SBC by imposing limits on these two parameters in their lag compensation. As of

this writing, none of the developers have publicly explained why certain values are chosen as the limits. This, combined with the fact that it takes even some of the most experienced developers quite a while to get the limits right, discourages other FPS games to follow suit. We study the effects of these two parameters and find that an upper lag limit of 250ms is suitable for a movement speed of 4m/s. We also confirm that the perception of SBC is significantly influenced by only the rollback distance, which is affected by both movement speeds and the amount of lag affect.

5. To address the drawback of CLC, we have proposed a novel advanced lag compensation (ALC). The idea is that the eligibility for lag compensation is determined per shot instead of per player. A shot is lag compensated if it would not cause SBC. ALC can significantly reduce the number of both actual and perceived SBC while retaining the benefits of TLC. ALC is better than CLC in that it decides whether to perform lag compensation or not based on each shot but not the player. Each shot fired from a player is lag compensated as long as it does not cause low-ping players to be shot behind covers. We have implemented ALC in our game to evaluate its performance. Our results show that ALC significantly reduces the total number of SBC by 94.1%. Both the actual and perceived numbers of SBC also see a significant drop ($p < .05$). As a lag compensation algorithm, ALC is as effective as TLC in mitigating the impacts of lag. The actual and perceived difference in hit accuracy under both algorithms are insignificant ($p = .158$ and $p = .18$).

6. The fundamental part of ALC is the algorithm for detecting SBC. As far as we know, it is the first of its kind. Apart from using the algorithm in

ALC to determine if a shot should be lag compensated or not, we can also use it to measure the number of SBC instances in a match. This allows us to objectively evaluate the effectiveness of ALC in reducing SBC. We can directly compare the number of SBC instances in a match under TLC and ALC instead of having to rely on the number reported by players and questionnaires.

Chapter 2 Lag Compensation

In this chapter, we first look at the algorithm for hit detection in FPS games without lag compensation along with its problems. Then we go into the mechanisms for traditional and conditional lag compensation, along with their drawbacks.

Traditional lag compensation has been a *de facto* standard as it is used in most multiplayer shooter games. It, however, introduces inconsistencies. A few recent games try to address the problem using what we classify as conditional lag compensation. This approach, however, can be an overkill.

Let us establish the criteria for a good algorithm before dwelling on the details. As a lag compensation mechanism, obviously it should significantly improve the shooting accuracy of the players. Lag, however, cannot be eliminated but only traded for another type of inconsistency [9, 5]. Such inconsistencies should have a low, preferably no, perceptual impact on the players.

Figure 2.1 illustrates the problem of lag in multiplayer FPS games. Consider two players, Alice and Bob, playing on a server with a tick rate of 60 Hz. We assume the client send rate is the same as the server tick rate. This means that every player sends updates, which includes information such as his location and rotation, to the server every $1/60 = 16.7$ms. Alice is a high-ping player with an

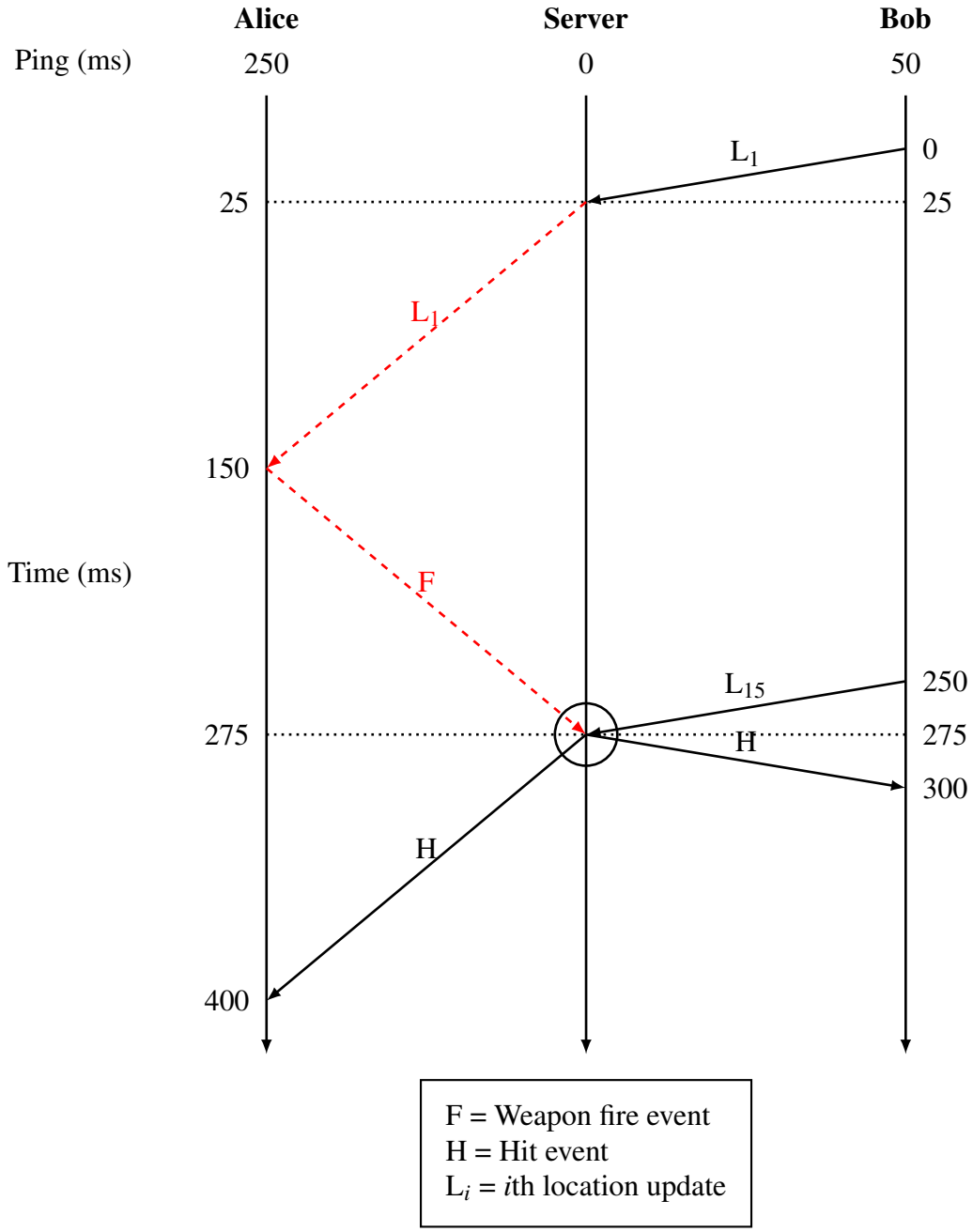


Figure 2.1: A timeline diagram illustrating the problem of lag.

RTT of 250ms and Bob is a low-ping player with an RTT of just 50ms.

In this example, Bob sends his first location update to the server at 0ms. Upon receiving it at $0 + 50/2 = 25$ms, the server relays the update to Alice, who in turn receives it at $25 + 250/2 = 150$ms. Alice then fires at Bob basing on the information carried by the update. The fire event takes another 125ms to arrive at the server at 275ms (circled point). The server now has to decide whether Alice's shot hits or not.

Without lag compensation, the server just uses the latest information it has on all the players to make the decision. For Bob, this is location update L_{15} he sent at 250ms. The problem of this approach is that if Bob has moved during the 250ms period (spanned by red dashed messages), the server would count Alice's shot as a miss. This outcome is inaccurate to Alice because from her perspective, her aim was on target.

This is known as the client-server architecture which is used by most multiplayer games. All critical decisions are performed by the server. Clients can estimate or approximate the expected values but will always respect the server's decisions.

One of these critical decisions in shooter games is hit detection, the process of determining if a shot hit any player. It is performed whenever the server receives a fire event. The pseudocode for hit detection without lag compensation is shown in Algorithm 1. Here we assume the game uses hitscan weapons, meaning that bullets travel with infinite speed along a straight line and thus any hit is instant [23]. For clarity's sake, names of functions that run on the server's side have a prefix of "server", and "client" for client-side functions. We simplify the calculation of *end* in line 3 by assuming that bullets always travel wherever the

Algorithm 1 Hit detection without lag compensation

```
1: procedure ServerReceiveFire(shooter)
2:     start ← shooter's camera location
3:     end ← start + shooter's aim direction × trace distance
4:     perform a collision trace from start to end
5:     if hit detected and hit actor is a player then
6:         target ← hit actor
7:         if shooter can damage target then
8:             send result to shooter and target
```

shooter aims. In practice this is usually not the case due to recoil and spread. In line 7, the evaluation should include checking if *shooter* is still alive. This is necessary to prevent kill trading[1] from happening due to lag.

2.1 Traditional Lag Compensation

Figure 2.2 illustrates how traditional lag compensation (TLC) works. Everything is the same up until when Alice's fire event reaches the server at 275ms. Instead of using the latest states of players it possesses at that moment, the server now decides the outcome of Alice's shot with those at 25ms (circled point). The server essentially rolls back all players behind the scene (players are not actually teleported back to their previous locations) by 250ms (Alice's RTT) before making the decision.

This outcome is now consistent with Alice's perspective and she no longer needs to lead her shots in order to actually hit the targets. The issue, however, is passed from Alice to Bob. Note that Bob's location which the server uses to

[1]Two players firing at each other both die even though one of them should have already been dead [24]

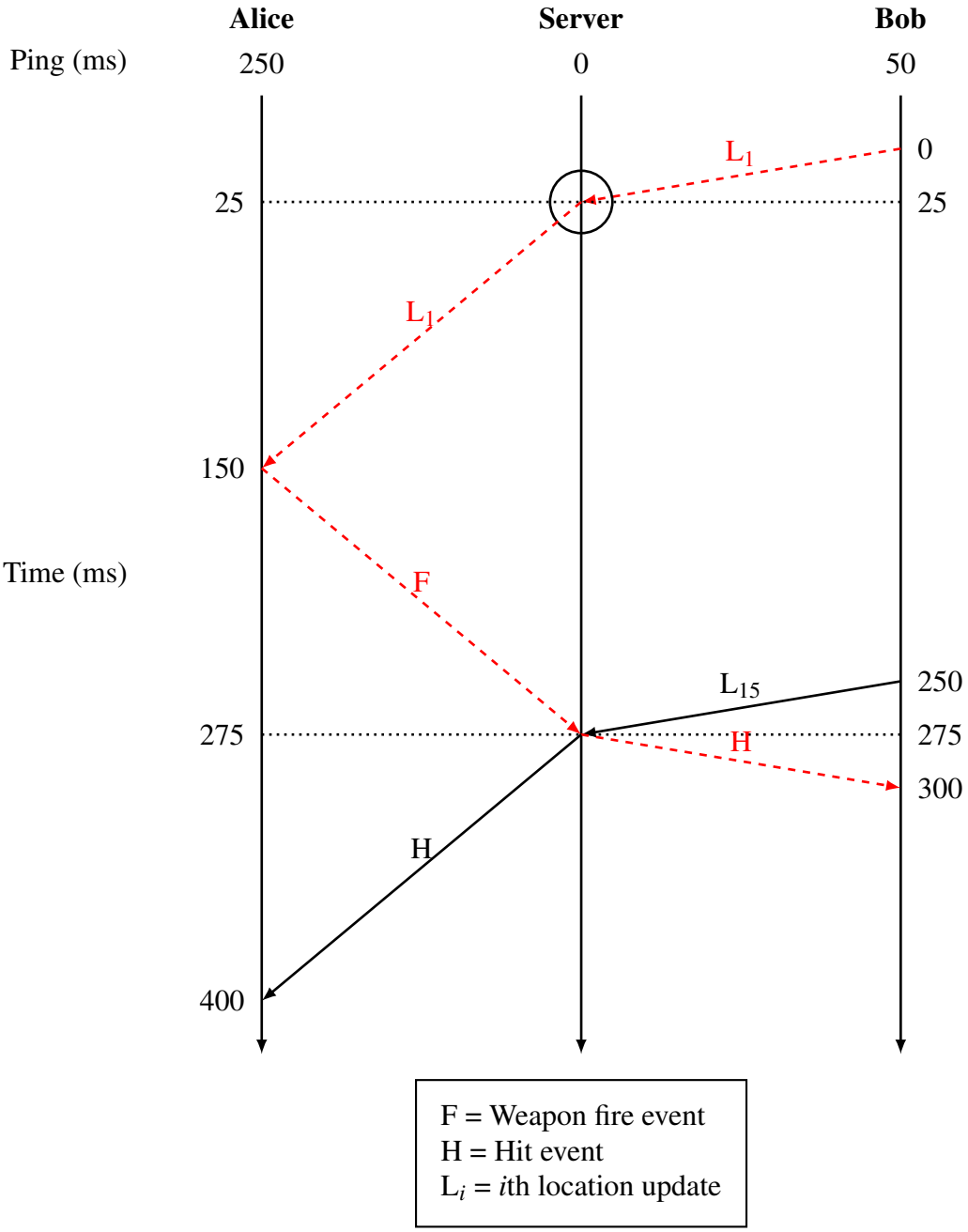


Figure 2.2: A timeline diagram illustrating TLC and its problem.

make the decision originates at 0ms. From Bob's perspective, if he has moved into any cover since then (period spanned by red dashed messages), he would have received unreasonable damages at 300ms. This is the problem we refer to

Algorithm 2 Traditional lag compensation

```
1: procedure ServerTick
2:     for all players do
3:         if length of player's frames = maxFrameSize then
4:             remove the first element of player's frames
5:         push player's states into player's frames
6:
7: procedure ServerReceiveFire(shooter)
8:     frameIndex = maxFrameSize−round(shooter's ping/(1000/tick rate)−
   1
9:     for all players do
10:        rollback player's states to player's frames[frameIndex]
11:    start ← shooter's camera location
12:    end ← start + shooter's aim direction × trace distance
13:    perform a collision trace from start to end
14:    if hit detected and hit actor is a player then
15:        target ← hit actor
16:        if shooter can damage target then
17:            send result to shooter and target
18:    for all players do
19:        restore player's states
```

as "shot behind covers" (SBC).

Algorithm 2 shows the pseudocode for TLC. ServerTick is called every frame (or tick). In our example, this means that ServerTick runs 60 times per second. We only save each player's states for a fixed period of time in order to save memory. If states should be saved for n seconds, $maxFrameSize$ should be $max(n/(1/\text{tick rate}))$. ServerReceiveFire is called whenever the server receives a fire event. The first loop rolls back the states of all players and the second one restores them after the hit detection is completed.

2.1.1 Shot behind Covers

Although under TLC, players are rolled back behind the scene by the server upon receiving each shot, not every roll back results in "shot behind covers" (SBC). In fact, as we will show in Chapter 7, only a few percentages of the total number of shots lead to SBC. This, however, in no way downplays the problem of SBC.

SBC does not occur if the victim has not moved into any cover by the time of receiving the shot. This includes if the victim has not moved at all, or has moved only along the shooter's aim. In all these cases, the line of sight between the shooter and the victim is not broken. SBC is more likely to occur if the victim is moving perpendicularly to the shooter.

It is unlikely for the victim to perceive SBC if the whereabouts of the shooter is unknown to him. This can be affected by quite a number of factors. These include user interface (UI) elements that help players locate the shooters. The effects of such UI elements will be investigated in Section 6.1. Some shooter games switch players from a first-person camera to a "deathcam" or "killcam" briefly upon their demises. These often reveal the shooters' locations and increase the possibility of players realizing being retrospectively shot behind covers. SBC is also more likely to be noticeable in third-person shooter games because the camera is placed behind the player's character.

While this study focuses on mitigating SBC for low-ping players, it should be noted that high-ping players can also experience SBC for a different reason. Figure 2.3 illustrates this using Alice and Bob again. This time, however, Bob,

a low-ping player, fires at Alice, a high-ping player, instead of the other way around. Bob fires the shot at 150ms using Alice's location that originates at 0ms. When the server receives Bob's shot at 175ms, it determines that Alice is shot. The information arrives at Alice at 300ms. If Alice has moved into any cover during the last 300ms (period spanned by red messages), she would be shot behind covers at 300ms. The cause, however, is mainly Alice's own 250ms lag, which accounts for five sixth of the 300ms RTT. To summarize, low-ping players are shot behind covers mainly due to the shooters' lag, while for high-ping players it is mainly due to their own lag.

2.2 Conditional Lag Compensation

The launch fiasco of Battlefield 4 prompted the developers to come up with what we refer to as conditional lag compensation (CLC). As its name suggests, the server no longer performs lag compensation unconditionally anymore. A player is eligible for lag compensation only if, for example, his amount of lag is within a certain limit. In the case of Battlefield 4, it is named `FrameHistoryTime` (FHT) [25]. According to the official document, an FHT value of 0.125 means that the server will roll players back for 0.125s at most, thus a player with an RTT higher than 250ms will not be lag compensated [21]. One of the developers reveals further that it also takes a player's movement speed into accounts [26]. This means that with an FHT value of 0.125, a player with an RTT of lower than 250ms may still be ineligible for lag compensation if he is moving fast enough. In other words, FHT is really a limit on how far in distance players can be rolled back by lag compensation.

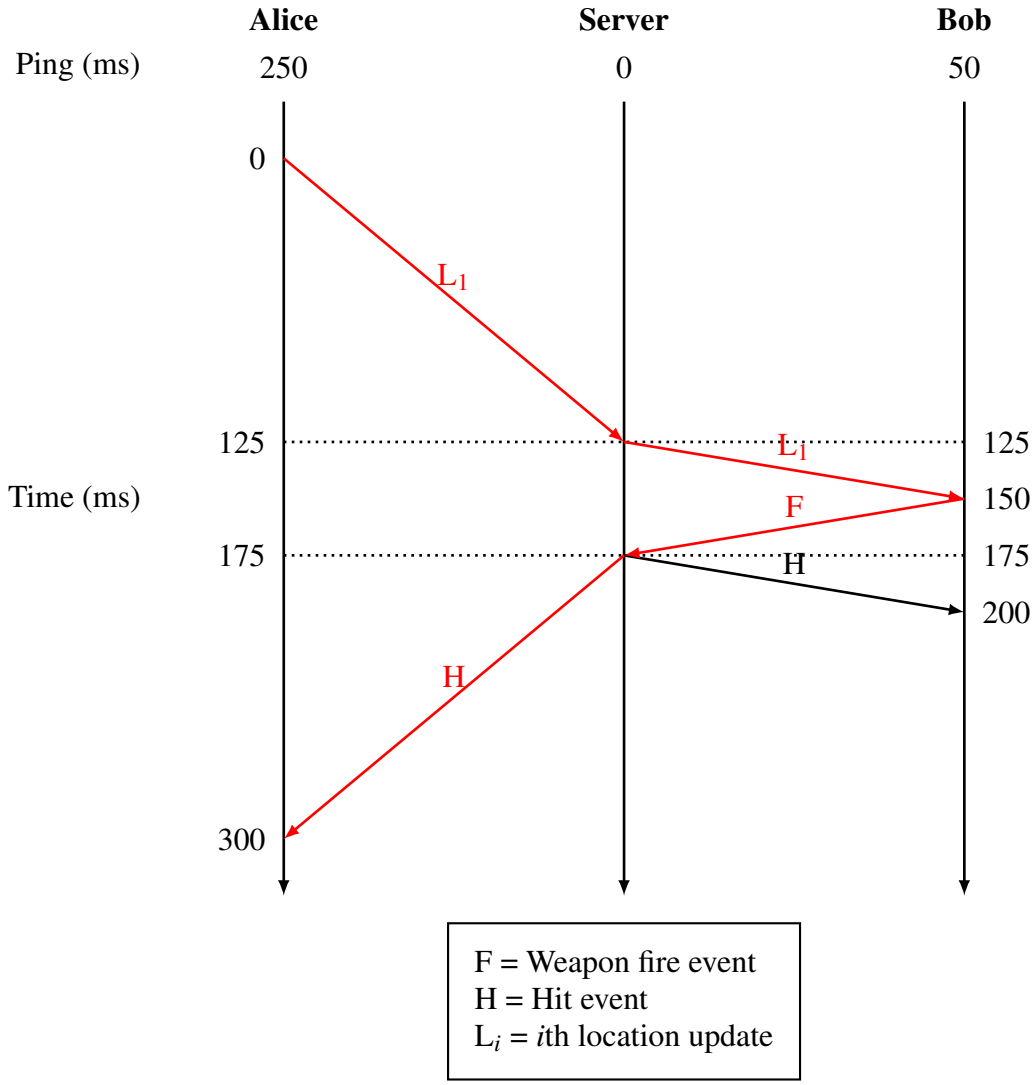


Figure 2.3: A timeline diagram illustrating SBC for high-ping players.

CLC should not be particularly difficult to implement. As shown in Algorithm 3, CLC only needs slight modifications to TLC. The challenge to this approach though is to strike a balance between fairness to low- and high-ping players. When the FHT value is set too high, SBC happens more frequently; when the FHT value is set too low, dusting happens more frequently. Dusting describes situations where the shooter sees client-side hit effects (e.g. blood splashes) but the shots are not actually registered by the server. The earlier FHT values in

Algorithm 3 Conditional lag compensation

```
1:  procedure ServerTick
2:      for all players do
3:          if length of player's frames = maxFrameSize then
4:              remove the first element of player's frames
5:          push player's states into player's frames
6:
7:  procedure ServerReceiveFire(shooter)
8:      shouldCompensate = shooter is within the designated limit
9:      if shouldCompensate then
10:         frameIndex = maxFrameSize−round(shooter's ping/(1000/tick rate)−
    1
11:         for all players do
12:             rollback player's states to player's frames[frameIndex]
13:     start ← shooter's camera location
14:     end ← start + shooter's aim direction × trace distance
15:     perform a collision trace from start to end
16:     if hit detected and hit actor is a player then
17:         target ← hit actor
18:         if shooter can damage target then
19:             send result to shooter and target
20:     if shouldCompensate then
21:         for all players do
22:             restore player's states
```

Battlefield 4 were nowhere near as low as the current ones. The developers could not set the value any lower without causing players on fast-moving vehicles to experience dusting everywhere. Their solution to this issue was to introduce dual FHT values in October 2015, two years after the game launched [21]. Since then, vehicles in Battlefield 4 use an FHT value of 1 and 0.125 for everything else. This makes it possible for players in fast-moving vehicles to still hit the targets within the same RTT limit.

A few games have since followed in Battlefield 4's footsteps by employing CLC. This includes Battlefield 1, the latest installment in the franchise. Released

in October 2016, the game is still having its FHT values tuned by the same developers. Initially players in Battlefield 1 could be hit by opponents with an RTT of up to 475ms [27]. The limit is much looser than that in Battlefield 4. An update then brought the limit drastically down to 100ms, which is more than half of what is used in Battlefield 4 [28]. But one month later the RTT limit was updated to 160ms for the United States and Europe; 200ms elsewhere [29]. This ongoing tweaking demonstrates that the difficulty in CLC lies on determining the limit. A limit that works in one game may not work in another, even for those from the same franchise.

As of this writing two more FPS games have employed CLC. In Overwatch, the server will not lag compensate players with an RTT of higher than 250ms [30]. This is simpler than the FHT approach because players in Overwatch move roughly at the same speed most of the time [31]. Call of Duty: Infinite Warfare also uses a simple RTT limit of 500ms for matches on dedicated servers [13].

Although CLC does improve the problem of SBC, it can be an overkill. Recall that when a player exceeds the designated limit, he will not be lag compensated by the server regardless of whether his shot causes SBC or not. However, our experiment, which will be presented in Chapter 7, shows that in a typical match, only a few percentages of the total number of shots cause SBC. This means that said players have to lead their shots, which is difficult and unrealistic, when it is not necessary most of the time.

Chapter 3 Related Work

In this chapter, we review academic works related to lag in multiplayer networked games. We also look at the current state of lag in the game industry. They demonstrate the necessity of having and improving lag compensation in FPS games. It is not uncommon to have sufficiently lagged players in a match. Such high delays exceed the tolerance of most players and therefore needed to be mitigated by lag compensation. Moreover, game servers are expected, if not mandated, to accommodate such players. We further examine studies with subjective measurements in FPS games to gain insights into setting up our own approaches.

3.1 Lag in Multiplayer Networked Games

3.1.1 Academia

There have been a rich collection of academic literatures on lag in multiplayer networked games. They generally agree that the FPS genre is one of, if not the least tolerant to lag. Claypool and Claypool indicate that the overall lag for Internet connections can vary from hundreds of milliseconds to one second [1]. They point out that not all player actions are equally sensitive to lag. It depends on the genre and the game itself. The authors categorize different player actions

according to the precision and time required to complete the action. They conclude that games with a first person perspective, such as FPS and racing games, are the most sensitive to lag. Sports and RPG games have medium lag sensitivity, while that of omnipresent games like real-time strategy and simulation games is the lowest. They also show that a lag of 100ms significantly decreases the hit accuracy. Brun et al. agree that the FPS genre has the highest responsiveness and consistency requirements [9]. Pantel and Wolf believe lag causes "a major problem of network-based multiplayer games" [8]. They recruited 12 candidates to participate in experiments that determine the acceptable presentation delay for racing games. Their results show that such games can tolerate up to 50ms of delay but should avoid a delay of more than 100ms. They speculate that a presentation delay of 100ms or more may be acceptable for FPS games.

Among these studies exist many that focus on FPS games. Armitage hosts public servers for Quake III and records latency information of each player [32]. The latency tolerance of the 9522 players is estimated to be between 150ms to 180ms. Henderson uses a similar approach by hosting a public server for Half-Life and infer player sensitivities from the observed user behavior [33]. For the estimated 16969 players, there is a tendency to stop playing on the server when their latency exceeds 225ms to 250ms.

Armitage and Zander, however, point out that it is challenging to use public servers to access the effects of lag on the perceived quality because it is hard to gather opinions from the players [34]. The authors later revisit Quake III with the addition of Halo and conduct a usability trial. They have 6 Quake III players and 8 Halo players participate in 4 different trials, each has a randomly chosen delay and packet loss. Players are asked to rate the perceived quality and state

whether they would stay or leave the game under that conditions. The perceived quality drops from being good after a 300ms delay for Quake III and 200ms for Halo. Beigbeder et al. run over 200 experiments, each with 4 players, using Unreal Tournament 2003 [11]. Results show that the hit accuracy, as well as the number of kills and deaths, decrease significantly after a latency of 100ms. Quax et al. has 12 players participate in different scenarios in the same game [10]. Each scenario simulates a certain combination of delay and jitter. Besides recording the number of kills and deaths of each players, the authors also ask players to rate the network quality and its effect on their performance after each scenario. Results show that a delay as low as 60ms is enough to impact player experience.

Not all studies involve human players. Kaiser et al. try to correlate lag and jitter with player experience using bots [7]. Four bots that "mimic a real-user behavior" are connected to a dedicated server and play a series of matches in Quake III. The authors use the time between kills as the performance metric. They find that a lag of 50ms is enough to degrade player performance.

3.1.2 Industry

The Technical Certification Requirement (TCR) is a list of criteria that Microsoft requires a game to meet before allowing it to release on Xbox [35]. Other console manufacturers have similar documents under different names. According to a leaked TCR for Xbox 360, requirement #97 states that servers should cater for players with latency of up to 300ms [36]. Although the document applies only to Xbox 360 which belongs to a previous generation of consoles, it gives an approximate idea of the maximum delay a server is expected to cope with.

It is not uncommon for games to be sold in countries where no servers are present. This is especially true for independent developers. Take Battlefield 4 as an example. The game is available in both South Africa and South America, yet there were no console servers in the two regions at launch. This forces players from the two regions to play on servers located outside of their continents, making them "out-of-region (OOR)" players. A search on the game's forum reveals OOR players frustrated by the fact that they cannot play on any server without having a latency beyond their tolerance [37]. Feng and Feng show that players do not necessarily connect to servers that are geographically close [38]. One possible reason is that all servers in the same region are either full or not populated with enough players.

3.2 Lag Compensation

Bernier details some common lag compensating techniques used in multiplayer games with authoritative servers to hide the effects of lag. These include client-side prediction, extrapolation, interpolation and lag compensation [5]. The author mentions SBC, or "shot around a corner" as he calls it during the discussion about lag compensation. He explains that SBC can happen when the shooter is sufficiently lagged. The inconsistency can be more or less pronounced depending on the combat situations. As an employee at Valve, Bernier shares that lag compensation is implemented in the company's games because its benefits "easily outweighed the inconsistencies." Pfeifer echoes the problem of SBC in lag compensation [22]. The author suggests fixing it by having servers accept commands only within a certain amount of lag. This should prevent a few severely lagged players from impacting the experience of the less lagged majority. Pfeifer also

cautions that when dealing with latency issues, measures must be put in place to prevent cheaters from manipulating the system.

Brun et al. identify multiple types of inconsistencies that can present in multiplayer networked games. They suggest trading those with higher perceptual impacts with others. For instance, client-side prediction can be used to increase the perceived responsiveness but may cause local rollbacks if server's authoritative decisions disagree with local predictions. Aside from these lag compensation techniques, game fairness can also be improved by equalizing inconsistencies among players by artificially delaying some of the information [9]. This is precisely what Zander et al. attempt to achieve. The authors have developed an application that tries to improve fairness by equalizing lag across players [39]. They evaluate the effectiveness of their approach using Quake II with 4 client-side bots. The authors acknowledge that bots have "a number of limitations" but argue that they are advantageous in that bots are predictable, truly balanced in skills and never get tired. Their results show that the application evens the mean kill rates across all players to a value lower than a highly lagged player would have otherwise. Le and Liu also explore the possibility of balancing fairness in FPS games by equalizing lags among players [40]. The authors define a game to be fair if the average one-way lag between each client and the server is the same. Basing on this definition, they evaluate the performance of this strategy using a simulated network consisting of 11 nodes, one being the server. Simulation results show that game fairness is significantly improved.

To have an idea of players' opinions on lag compensation and its problem, we can refer back to the aforementioned search on Battlefield 4's forum. It reveals not only frustrated OOR players, but also others who are irritated by them. To

be specific, many players are annoyed by the fact that lag compensation favors the shooters. Such kind of complaints is not uncommon in popular FPS games. Some players believe an appropriate solution to be abandoning lag compensation. Although such suggestions are often quickly dismissed by those who understand game networking, the academia has not provided any empirical evidence to falsify their conjectures.

3.3 Subjective Measurements in FPS Games

Besides network latency, many works on other issues in FPS games involve subjective measurements as well. Ivkovic et al. explore the effects of input lags arisen from input and display devices [41]. They attempt to mitigate the impacts using lag compensation techniques based on aim assistance. The authors recruited 18 players to participate in two sessions of an experiment implemented as a C++ program. Each player is instructed to aim and/or shoot at targets under different amounts of input lag. Participants fill out a 7-point Likert scale questionnaire after testing each lag. They are asked to rate their own performance, the effect of lag on it, frustration of the round, and the level of lag for the controls. Results show that their compensation technique significantly reduces the impacts of input lags on targeting and completely eliminates the problem for tracking.

Vicencio-Moreira et al. study the efficacy of two aiming assistance techniques [42]. The authors evaluate the two methods by having 10 pairs of players competing in a custom game created with Unreal Development Kit. 4 rounds of Deathmatch are played for each pair. Three types of weapons are available in the game: an assault rifle, a pistol and a sniper rifle, with the former two equipped by

default. The assault rifle has the highest rate of fire but the sniper rifle is the most powerful. Player performance is evaluated by looking at 4 types of data: the score difference within each pair, the hit accuracy, the number of kills and answers given to the questionnaire. After each round, players are asked if they think the round is fair and to identify the stronger player. Their results indicate that aiming assistance increases the performance of novices but not the scores. The authors thus conclude that additional methods should be employed along with aiming assistance to balance the gameplay.

Claypool et al. investigate the effects of frame rate and resolution on player performance and enjoyment in FPS games [43]. They recruited 64 participants to play Quake III with different combinations of frame rates and resolutions. A custom map is created to minimize uncontrollable and distracting factors, as well as maximizing the number of kills, which is used to measure player performance. Each player is pitted against a bot and is allowed to use only one weapon. After each game, a client program collects players' ratings on the game playability, the picture quality and the amount of efforts they spent. The authors find that frame rates and resolutions have a similar effect on the perceived quality but the former has a more pronounced influence on player performance.

Chapter 4 Performance of TLC

After looking at how TLC works in Chapter 2, now we evaluate the performance of TLC. Because TLC mitigates the effects of lag by favoring the shooters, low-ping players may suffer from SBC when the shooter is sufficiently lagged. This understandably makes some players feel being unfairly treated and question the inclusion of TLC in shooter games. The purpose of this chapter is to address the debates within the gaming community on the existence of TLC. There has been a lack of empirical evidence needed to settle the dispute. A justification of TLC would also benefit any future effort at improving lag compensation in FPS games.

We set up an experiment based on [10] to study the effectiveness of TLC. We have 4 players participating in Counter-Strike: Global Offensive (CS:GO), a popular FPS game with hundreds of thousands of active players daily. All players are males in their 20s, two of them are beginners with experience in the predecessors of CS:GO, one being an intermediate gamer, and one being an experienced gamer. To minimize potential skill differences, there is only one human player in each match. Each player teams up with 4 bots and plays against 5 others. We also allow only the P250 pistol to be used in the game in order to further control the experiment. We use Dust II (`de_dust2`) as the map due to its popularity among players throughout the Counter-Strike series and the fact that some participants are already familiar with the map.

Table 4.1: Scenarios for studying the effectiveness of TLC.

Scenario #	1	2
Ping (ms)	150	
Lag compensation	On	Off
Interpolation period (s)	0.015625	
Tick rate (Hz)	64	
Vertical synchronization	Off	

All rounds are played on a PC with a CPU of Intel Core i7-3770 @ 3.40GHz, a GPU of Intel HD Graphics 4000, 8GB RAM and a monitor with a 60Hz refresh rate (Samsung HP L2314). In order to minimize latency, both the client and server are run on the same PC. Table 4.1 shows the two sets of configurations, or "scenarios", used. Each scenario is played for 5 minutes.

As stated in [44], the visual lag introduced by interpolation has to be compensated by lag compensation. It would therefore be ideal to turn off interpolation completely in the experiment. We, however, have to set the interpolation period to the minimum value instead as interpolation is always enabled in CS:GO irrespective to the configurations.

We originally measure each player's performance using his kill-death difference (K-D), which is defined by the difference between the number of kills and deaths scored in each round. This performance metric, however, causes fluctuations in some of the results. The K-Ds across different trials for the same scenario could vary significantly (up to around 85%). The reason is the non-deterministic nature of player distribution. In some matches a player can wander around the map for half a minute without encountering any enemy. Our solution is to repeat the experiment using shooting accuracy as the performance metric because it is much less sensitive to player distribution given the abundant amount of shots

Table 4.2: Shooting accuracies with and without TLC.

Scenario #	Player 1	Player 2	Player 3	Player 4	Mean
1	31.8%	27.8%	15.6%	30.1%	26.3%
2	28.6%	23.5%	14.1%	20.3%	21.6%

fired in each match.

Without lag compensation, all players, regardless of their lag, would have to resort to leading their targets, which is unrealistic and most importantly, difficult. Our hypothesis is therefore that TLC increases the shooting accuracy.

4.1 Results and Analysis

Table 4.2 shows the shooting accuracies of all the players for the two scenarios. The mean accuracy in scenario 1, in which lag compensation is enabled, is around 5% higher than that in scenario 2. A paired t-test shows that the mean accuracy is increased significantly in scenario 1 from scenario 2 ($p < 0.05$), meaning that TLC is effective in reducing the impact of lag. We therefore reject the null hypothesis. TLC does increase player performance in terms of shooting accuracies.

A higher mean accuracy with TLC enabled is contributed by how the server determines the outcome of a shot. With TLC, the server refers to the game world as seen by the shooter when he fires the shot, instead of that when the server receives the event. This improves hit registration.

As mentioned in Section 1.1, TLC may seem to transfer inconsistencies from

lagged players to the others. This result, however, rejects the conjecture of some players that lag compensation should be removed entirely in order to fix SBC. TLC fulfills the first criterion we have established in Chapter 2 for a good lag compensation algorithm, i.e. significantly improving the shooter accuracy. Abandoning TLC means that all players, including less lagged players, would need to lead their shots by some amount, impacting their performance. We believe the appropriate solution to SBC is to impose limits on TLC, instead of lag compensating all players unconditionally.

Chapter 5 LAGCOM

LAGCOM is a multiplayer FPS game we have built to study lag compensation, the induced SBC and ultimately improve the algorithm. In this chapter, we discuss various mechanisms presented in the game. Some of these are unique while others are common FPS game mechanics and features that behave differently than they do in commercial games.

Let us first explain our motivations for building a custom game. Our first experiment, present in the previous chapter, evaluates the performance of TLC using the popular FPS game Counter-Strike: Global Offensive (CS:GO). In the experiment, we want to turn off interpolation, which should be possible according to the official documentation. In practice, however, some form of interpolation always exists irrespective of our configurations. We have also tried to mess around with different related parameters but to no avail. This makes us start realizing the deficiencies of using commercial games to achieve our research objectives. CS:GO already offers one of the highest number of server configurations among popular FPS games. But even that is inadequate for our subsequent experiments. Without access to the source codes, it is very difficult, if not impossible, to modify the implementations of many fundamental game mechanisms. These range from small UI tweaks like disabling hit indicators, to game changers such as employing a new lag compensation algorithm. With our own custom game, we can also augment it with features that facilitate smoother and less error-prone

experiments.

We have built LAGCOM from scratch using Unreal Engine 4 (UE4), an established and industry proven game engine with a robust network implementation. The development effort for the game is approximately 10 person-months. Other researchers are free to obtain the project on GitHub (`https://github.com/stevenlwk/LAGCOM`) and modify the game to suit their needs. We believe LAGCOM can be utilized to study many shooter game-related issues outside lag compensation, or game networking for that matter.

5.1 Unique Features

5.1.1 Lag Compensation

LAGCOM supports three types of lag compensation: none, TLC and the new ALC. We do not implement CLC in the game because we have controls over the latency of each player in experiments. If CLC is required, it can be substituted for using TLC on low-ping players and no lag compensation on high-ping players. To the best of our knowledge, LAGCOM is the first game that is able to detect SBC, which can be done both locally and on the servers. The detection algorithm is detailed in Section 7.3. Besides being the core of ALC, the ability to detect SBC also expands our capability for studying the impacts of TLC on low-ping players.

5.1.2 Server-Side UI and Visual Settings

LAGCOM supports mandating some UI and visual settings from the server. These include the field of view (FOV), the presence of hit indicators, mini-map and each player's ping time on the scoreboard. Details of these are presented in Section 6.1.1. Usually players can customize such settings in commercial games. For example, some players may prefer a narrower FOV due to limitations of their hardwares, while some may find a wider value necessary to prevent from getting headaches or nausea. The ability to mandate these settings allow us to investigate their effects. It is possible to hand over the controls back to the players when experiment settings allow.

5.1.3 Built-in Questionnaire Support

Real players (in contrast to bots) are often involved when LAGCOM is used in experiments. To facilitate subjective measurements, the game supports built-in questionnaires. Researchers can designate any number of questions to be asked in the configuration file. A question can be a binary one (two options) or one with answers as a Likert-type scale (5 levels). At the end of any match specified in the configuration file, players are prompted to answer the predefined questions (see Figure 5.1). The next match can start only after all players have finished the questionnaire. If the questions are unchanged throughout the experiment, they need to be set only once at the first match. Alternatively, each match can be configured to have its own set of questions.

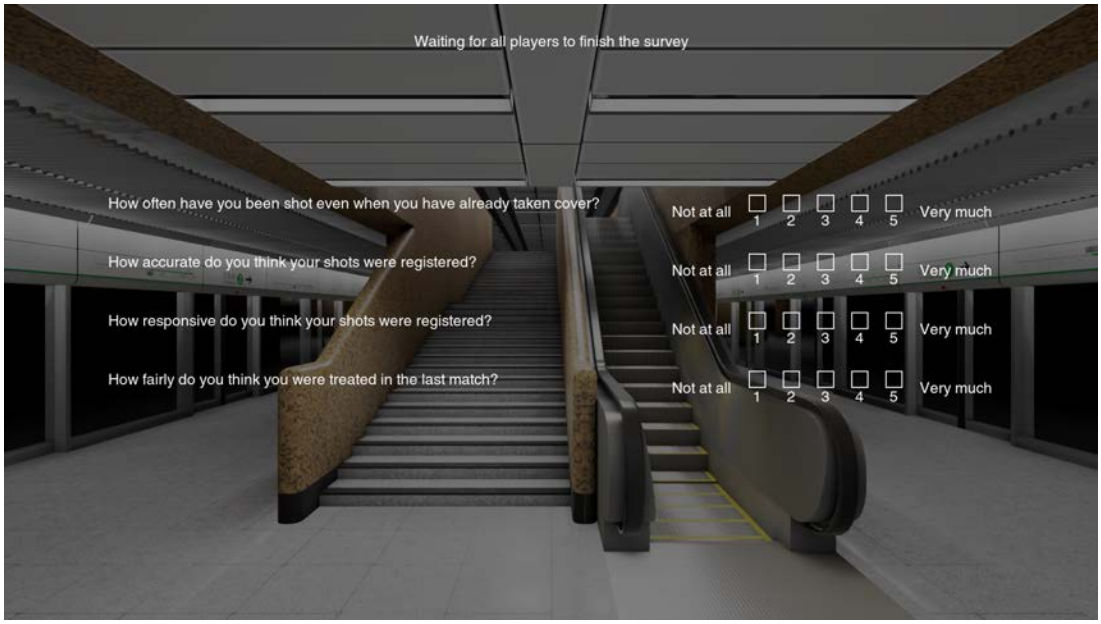


Figure 5.1: Questionnaires in LAGCOM.

5.1.4 Integration with Network Simulation Utility

Although UE4 contains a native command for simulating network latency, it only applies to outbound packets. This can sometimes be inadequate for empirical studies. In our experiments, we use an external program named `clumsy` [45] to perform the task instead. While `clumsy` supports a graphical user interface, it can also be launched from the command line. LAGCOM supports generating a batch file for `clumsy` for each match. In the configuration file, we can specify which players should have lag injected in each match.

5.2 Experiment-Oriented Features

5.2.1 Configuration File

Before the game server starts, it reads the configuration file `MatchSettings.ini`. It supports a wide range of game settings. The main purpose of the configuration file is to specify the settings of each match and the order in which they are played. Each match requires a map name, a game mode name, a duration, an interval before the next match, and a lag compensation algorithm (if any). The duration and interval can range from one second to 5999 seconds (99:99). Optionally a match can be named. Events and statistics of each match are logged in a separate file with the match's name. Having a meaningful name for a match can help during result analysis.

5.2.2 Administrator Mode

As the experiment is progressing, we may need to adjust some of the configurations. For example, before the real experiment begins, players may be able to get familiarized with the game in a warm up match. Because it is hard to estimate the time it would require to prepare the players for the experiment, a warm up match is usually set to last the maximum duration. When all players are ready, experiment moderators can join the game as an administrator and manually start the first formal match. Administrators can also spectate the game with a free camera. This helps discover and sort out problems as soon as possible.

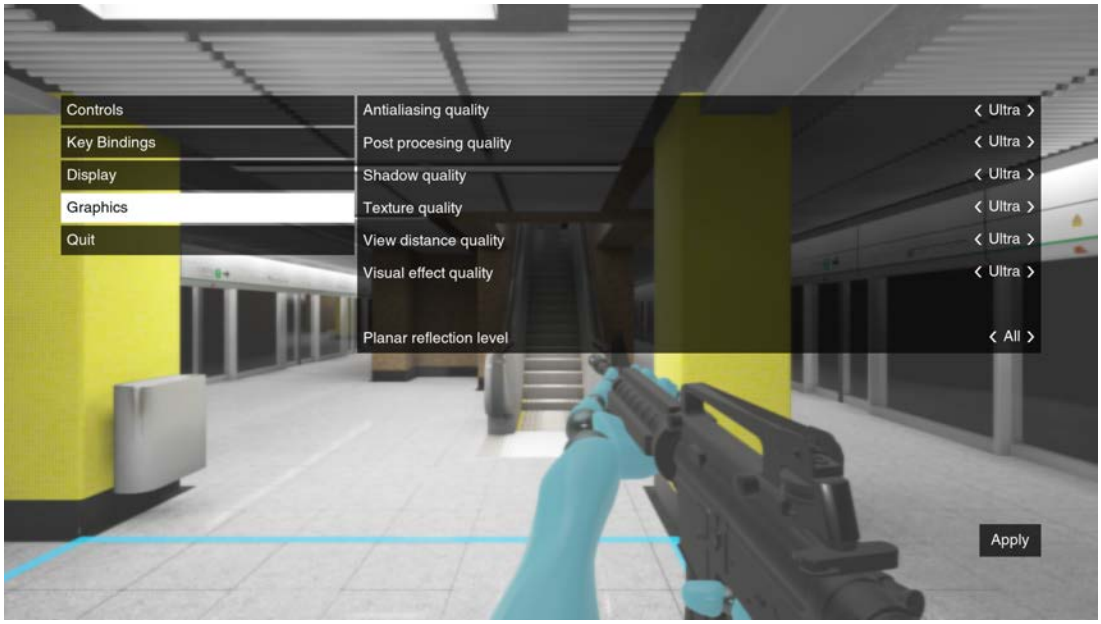


Figure 5.2: Graphic settings in LAGCOM.

5.2.3 Graphic Settings

All FPS games provide some form of graphic settings. This is no different in LAGCOM (see Figure 5.2). While allowing players to customize how the game is rendered may seem trivial, it is significant in our experiments. As Bernier mentions, interpolation must be taken into account in lag compensation [5]. Being able to run the game at 60 frames per second on a 60Hz monitor eliminates the need for interpolation. The ability to tweak graphic settings in LAGCOM allows it to run smoothly even on non-gaming PCs. In all of our experiments, LAGCOM runs on computers equipped with an Intel Core i5-4210U @ 1.7GHz, a GTX 860M and 16GB of RAM. To verify that the game indeed runs at 60 frames per second, we use the native command `stat fps` provided by UE4.

Figure 5.3: Invincibility in first person.

Figure 5.4: Invincibility in third person.

5.2.4 Spawn Protection

In some experiments, especially ours, researchers may want to discourage one-sided engagements. LAGCOM can help to achieve this with spawn protection.

Our experiments are the most productive when victims are aware of the whereabouts of the shooters. We therefore prefer players not to carry out "surprising attacks" such as flanking or even camping the enemy base. The former is hindered by the absence of flanking routes on the map. To thwart base camping, we enable spawn protection in the game.

Spawn protection is usually achieved in shooter games with either prohibited areas or spawn invincibility. For the former, the opponent base and their surroundings are either impassable or hostile to trespassers. Players who enter such areas are killed after a short period of time. This, however, is inadequate because the maps used in our experiments are relatively small, players can easily camp all exit routes from the opponent base instead and directly fire at anyone coming out of it. We therefore choose the latter measure over the former.

Upon respawning after being killed by an opponent, a player is invincible (immune to any damage) for a short period of time (5 seconds in our experiments). This removes the incentives for hiding behind enemy spawn points as newly respawned players can easily eliminate campers without suffering any damage. Invincibility is visible to both the beneficiary and other players through a glowing character (see Figure 5.3 and Figure 5.4).

5.3 Differences in Common Game Mechanisms

Most common FPS game mechanics and features in LAGCOM are implemented and configured the ways they are in commercial games. For the benefit of the

experiments, however, some are (or were once) different than how they typically work.

5.3.1 Running

Like in most FPS games, running is implemented in LAGCOM. In all of our experiments, however, this feature is disabled. The reason is that our study focuses on lag compensation and the induced SBC. Distances which players are rolled back by lag compensation play an important role. Because rollback distances are affected by movement speeds, it is most appropriate for all players to move at an identical speed throughout each experiment. For the same reason, we also set the walking speed when crouched to be the same as the normal walking speed.

5.3.2 Spotting

The mechanism for spotting enemies varies among shooting games. In Battlefield 4, for instance, enemies can be spotted by pressing a dedicated key while aiming at them. Spotted enemies appear on the player's mini-map and screen with a marker over their head for up to 7 seconds. Enemies are spotted also for teammates that can see them. In Counter-Strike: Global Offensive, however, having a direct line of sight to the enemies is enough to spot them. In LAGCOM, we choose to display all players, friendly and hostile, on the mini-maps of all players at all time. This removes the need for spotting which may distract some of the players. Always showing all players on the mini-map can also encourage more engagements which is beneficial to our experiments.

5.3.3 Aiming Down Sights

Aiming down sights (ADS) is a gameplay mechanic that assists players in aiming by allow the sight of the weapon to be brought to the center of the screen (usually by pressing the right mouse button). ADS is implemented in LAGCOM but is disabled in all of our experiments. Although ADS is present in many shooter games, it is notably absent in popular competitive FPS games like the Counter-Strike series, Overwatch and the Unreal Tournament series. Even for games that include ADS, its implementation can vary quite drastically. Activation of ADS is usually accompanied by zooming in of the player's camera. The level of magnification, however, varies across games, weapons and sights. When a player fires the weapon while aiming down the sight, some games treat the bullets as coming from the sight, i.e. the center of the screen; others from the muzzle of the weapon. The absence of ADS should also not matter much in our experiments because the maps we use are relatively small. ADS is particularly useful only for long-range target acquisitions.

5.3.4 Weapon Recoils and Spreads

Initially weapon recoils and spreads do not exist in the game. The reason for their absences is that we want to minimize randomness in experiments. To save times for engagements in a short 10-minute match, we also configured the weapon to have an infinite amount of ammunition, effectively disabling reloading. During some of the preliminary experiments, however, these were abused by some players. Offending players would keep pressing the fire button for an extended period of

time, sometimes even throughout the whole match, regardless of whether an enemy is spotted or not. Without recoils and spreads, there is no penalty for doing that. The mentality of the instigating players is that "perpetual firing" would put them at an advantage by having the first shot(s) over the enemy. Some players noticed and followed this behavior due to either the fear of missing out or that of being put at an disadvantage. Eventually we introduce recoils, spreads and reloading to the weapon in LAGCOM. We have not observed such behaviors since. Values used for the amount of recoils and spreads, as well as the reloading time are chosen in reference to those in Battlefield 4.

5.3.5 Deathcam

A deathcam is a perspective that activates upon death until the player is respawned. The exact implementation varies drastically across different shooter games. In some games the deathcam follows the player character. That means if the body slides away or launches into air after death, the camera would move accordingly. Other games have the deathcam follow the killer instead. On top of these, players are able to switch to the perspectives of other living teammates (and enemies). With all these different implementations, we choose the simplest one. The deathcam in LAGCOM is fixed to the last perspective seen from the player before his demise. Because our experiments involve studying the effects of SBC, having the deathcam follow the dead body or even the killer may influence the ability of the player to retrospectively identify instances of SBC. We believe this is the fairest approach.

5.4 Challenges and Difficulties

5.4.1 Lag Compensation Accuracy

Having an accurate lag compensation in LAGCOM is critical to this study and it took us some time to get the implementation right. When the server receives a shot, it rolls back all players on the server's side and perform collision tests between the shot and the skeleton of each rolled back player character. A character in LAGCOM has 67 bones in its skeleton and each requires a separate collision test. For an experiment that involves 6 players, it means that each shot would require at most 67×5 (excluding the shooter) = 335 collision tests. Our initial approach is to save the bounding volume [1] for each bone on the server on every tick because collision tests against a bounding volume is relatively cheap. In UE4, we can easily get the bounding *box* for a bone but it makes collision tests quite inaccurate. For instance, a player can shoot around the head of an opponent and still scores a hit. This is unacceptable. The cause is that some body parts are quite irregular in shape and it is more appropriate to use capsules as the bounding volume for skeletons.

Eventually we decide on a more complicated approach. Instead of saving each bounding volume, we save the transformation of each bone. We also add to each player character another skeleton that is visible only to the server. When the server rolls back a player, it transforms each bone of the skeleton. Although this approach introduces some overheads due to bone transformations, the resulting collision tests are perfect.

[1] A simple geometry, usually a box, sphere or capsule, that encapsulates the entire object

5.4.2 Match Rejoining

Due to problems with the display driver and overheating of the computers, some game clients crash during some of our experiments. Because by default all data associated with a player is erased upon leaving the match, this results in losing all statistics collected prior to the player's departure and the match has to be restarted. We therefore decide to add supports for players to rejoin a match along with their last statistics. This feature is not uncommon in card games but we are not aware of its presence in any major shooter game.

We collect many types of statistics during experiments. Some of the data such as the number of SBC can only be collected client-side. In order to reduce the workload of the server, we initially wait until the end of the match to send such telemetry data from each client back to the server. In order to support match rejoining, the server must always possesses the latest statistics of all players so that they can be restored upon rejoining. We therefore modify the game client to send each telemetry data back to the server as soon as it is collected.

5.4.3 Aesthetic Elements

From the beginning, LAGCOM is designed for experiments. It does not aim to be fun, beautiful or realistic, although many players do find the game enjoyable and pretty. Our priority is to finish the game and start the experiments as soon as possible. We therefore try to skip including as many aesthetic elements in LAGCOM we can. Some of such items, however, turn out to be necessary. The

lack of those elements may lead to suboptimal player performance or even a worse perceived fairness.

Initially player characters in LAGCOM do not have any animation other than movements such as walking, sprinting and crouching. In our preliminary experiments, this confuses a few players when changes in their states are not reflected through animations. For instance, due to the absence of a reloading animation, some players are baffled why they cannot fire their weapons all of a sudden.

The map used in our early experiments is not textured. All of the objects on the map are simply white. Although lights and shadows are present, a few players indicate after the experiments that it is difficult for them to determine the positions of some objects and to a extent the locations of some players. We therefore fully texture our second map and use it in the rest of our experiments.

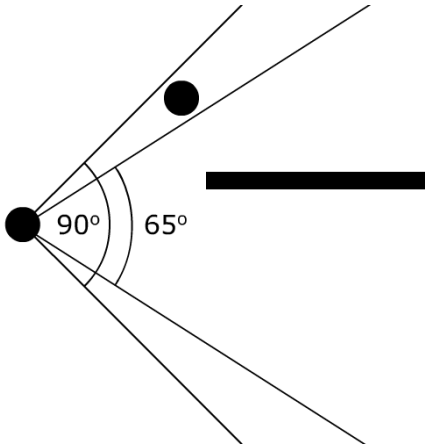
Chapter 6 Perception of SBC

A solution to SBC requires an understanding of not just the problem itself but also how it is perceived by the victims. In this chapter, we find out how the perception of SBC is affected by various factors. First we look at the effects of game mechanics and features that can usually vary across games, matches or players. Such mechanisms do not belong to lag compensation or even game networking at all. If found to influence the perception of SBC, however, they would need to be controlled accordingly in the following experiments. We then investigate the effects of RTTs and movement speeds, which directly concern lag compensation. Limits used in CLC have so far been based on these two variables. Studying their effects can help evaluate the current limits and establish appropriate ones.

6.1 Effects of Game Mechanics and Features

As discussed in Section 2.1.1, SBC is caused by the way TLC works. Many common mechanics and features in FPS games, while having nothing to do with lag compensation itself, may affect how SBC is perceived. It is essential to investigate the effects (or lack thereof) of them on the perception of SBC. Depending on the results, we may want to restrict settings for certain features in future experiments. This experiment addresses the second problem stated in Section 1.1.

Figure 6.1: Difference in player's view of the opponent with different FOV. The opponent disappears sooner with a FOV of 65°.



We first introduce the game mechanics and features being tested before going into the experiment settings.

6.1.1 Game Mechanics and Features

6.1.1.1 Field of View (FOV)

In video games, the field of view (FOV) influences the extent of the game world seen by the players. The observable game world expands with FOV, which is usually given in angles. In FPS games, FOV usually decreases when aiming down sights[1](ADS) given such feature is present.

As illustrated in Figure 6.1, when a player (the left circle) is moving into a

[1]A gameplay mechanic that assists players in aiming by bringing the sight of the weapon to the center of the screen.

Figure 6.2: Hit indicator (the red annulus sector).

cover (the rectangle) perpendicularly to the opponent (the right circle), FOV influences the last seen location of the opponent. Specifically with a smaller FOV, the opponent disappears sooner before the player is in a cover. Because of this, players having a smaller FOV may be less likely to be aware of the problem of SBC.

The default FOV in LAGCOM is 90°. It is changed to 65° in the experiment. Both are common values in FPS games.

6.1.1.2 Hit Indicator

When a player is shot, some FPS games display a hit indicator to show the direction in which the shot came from (see Figure 6.2). This is to substitute

Figure 6.3: Mini-map.

the lack of physical feedback in video games. Sometimes players can infer the opponents' locations from this. The presence of hit indicators may increase the likelihood of players identifying inconsistencies. Hit indicators are enabled by default.

6.1.1.3 Mini-map

As its name suggests, mini-map is a miniaturized map that shows the top-down view of the level. Depending on the spotting mechanic, conditions for revealing an opponent on the mini-map may vary from game to game. In this experiment, only the player himself is shown on the mini-map (see Figure 6.3). Players may be able to identify more inconsistencies with the inclusion of a mini-map, as it provides them with a better awareness of the surroundings. Mini-map is enabled by default.

6.1.1.4 Ping Time Display

In most FPS games, players can toggle the scoreboard (usually by pressing the Tab key) and check the statistics of all players, often including their ping times, or round-trip latency (RTT). After a player is shot or killed, discovering that the opponent has a high ping time may increase one's tendency to blame the outcome on it. The ping times of all players are shown on the scoreboard by default.

6.1.1.5 Rate of Fire

A weapon's rate of fire is measured in rounds per minute (RPM). A higher rate of fire shortens the interval between each shot, and may make it more difficult for a player to isolate each inconsistency. For example, it may be easier to identify one unfair sniper shot, than a few out of many shots from an assault rifle, which has a higher rate of fire.

The default firing rate of the only weapon in the game is 600 RPM, a typical value for assault rifles. In the experiment it is changed to 40 RPM, a typical value for sniper rifles.

6.1.1.6 Time-to-kill (TTK)

Time-to-kill (TTK) is the minimum time needed to kill a player. It is usually manipulated by adjusting weapon damages and the maximum player health.

Though changing weapons' rates of fire can also affect the TTK, games usually adhere to the values of their real-life counterparts. Another reason for us to not manipulate the TTK by changing the rate of fire, is to avoid confusion over the cause of the potential difference in the results.

A higher TTK means that a player can endure more number of shots before dying. Thus players may be able to tolerate more instances of SBC. By default, the TTK in the game is 0.3s (maximum health: 100, weapon damage: 25). In the experiment we change each player's maximum health to 500, effectively increasing the TTK to 1.9s.

6.1.2 Experiment Settings

We invite 5 experienced FPS players to play LAGCOM for 8 rounds. The first round lets the players get familiarized with the game. The following two rounds are used to verify that the amount of lag injected (150ms) is enough to influence their perceptions of fairness. Starting from the forth round, each has a different game feature altered or disabled. To minimize the recency effect while allowing enough engagements to happen, each round lasts for 10 minutes. Players are asked the following questions after every round except the first:

1. Do you think you were treated fairly in the last round? (1(=not at all)-2-3-4-5(=very much))

2. Did you get a chance to look at your opponents' pings? (Yes-No) (This question was not asked when the ping time was not displayed)

Table 6.1: p-values for each game mechanic and feature.

Game feature	p-value
FOV	0.31065
Hit indicator	0.5
Mini-map	0.18695
Ping time display	0.5
Rate of fire	0.18695
TTK	0.41378

We invite only experienced players because it is suggested that more experienced players are more sensitive to QoS degradation than less experienced players [34]. Although SBC does not belong to QoS degradations, they are similar in the sense that both impair gameplay experience.

6.1.3 Results and Analysis

In the first round, no latency is injected and all game features are set to their default configurations. The same is true for the second round, except we start asking players to rate each round's fairness at the end. In the third round, we inject 150ms to all players and the perceived fairness is significantly worse than the second round ($p < 0.05$). Since TLC is present during the whole experiment, this implies that instances of SBC have taken places and are noticed by the players. We therefore conclude that injecting 150ms of RTT is enough for the purpose of this experiment. For the following results (see Table 6.1), the third round is used as the base for comparison.

We find that for experienced players, their perceptions of fairness are not affected by different game feature settings. For the FOV, the default value is 90°.

A one-tailed paired t-test shows that the perception of fairness does not improve when the FOV is changed to a smaller value of 65° ($p > 0.31$). Similarly, disabling hit indicators does not improve the perception of fairness either ($p = 0.5$). The same p-value is also obtained when players' ping times are hidden from the scoreboard.

Without the presence of a mini-map, 3 out of 5 players perceived the round to be fairer, but the difference is not significant according to the paired t-test ($p > 0.18$). The same p-value is also obtained when the rate of fire is changed from the default 600 RPM to 40 RPM, indicating that a lower rate of fire does not significantly improve the perception of fairness. As predicted, a longer TTK of 1.9s does not significantly improve the perceived fairness from the default 0.3s ($p > 0.41$).

These results suggest that mechanics and features in LAGCOM can be implemented the way they are in typical commercial games. Latter experiments on lag compensation and SBC can also allow players to customize relevant settings, without affecting their perceived fairness.

The key takeaway is that for experienced FPS players, their perceptions of fairness do not change significantly with different game feature configurations. One may conjecture that this is because the players were not aware of the different settings. Although this is possible, it is not the case in this experiment. From the answers given to the end-of-round questions, all players have looked at their opponents' ping times at least once during the experiment. Some players also asked if the absences of hit indicators and the mini-map were normal during the rounds in which the items were tested. For other features like FOV, rate of fire and TTK,

the changes are simply too obvious to miss. A plausible explanation, therefore, is that experienced players can correctly spot most, if not all inconsistencies, even when game features are altered in ways to impair their abilities to do so. Their perceptions of fairness thus do not change significantly when "easier" settings are used.

Besides the game features investigated, we have also gained some insights into how a few other mechanisms should be configured in latter experiments on lag compensation and SBC. In the experiment, reloading is not needed as each magazine is set to have an infinite number of bullets. The reason for this is to promote more engagements in each 10-minute round by saving the time spent by reloads, each of which usually renders a player unable to fire the weapon for a few seconds. Typically a player reloads a dozen times in a 10-minute round, which can amount to a total of almost one minute. The absence of the need to reload, however, has a negative side effect on the hit accuracy. In fact, it renders the hit accuracy useless to be analyzed in any meaningful way. The reason is that some players abused this setting by holding the trigger the whole time. Normally this would not happen because weapons have bullet spreads and recoils, which introduce randomnesses and deviations that magnify with the time the weapon is continuously fired. Bullet spreads and recoils, however, are also disabled in the experiment as we want to avoid any kind of randomness. Although the hit accuracy is not our primary metric, it could still provide some additional insights should it be usable. It is therefore preferable to enable reloading, bullet spreads and/or recoils in similar experiments.

The results show that we can augment the game by allowing players to set the FOV and customize the look of the mini-map. The game can also follow various

common practices of commercial FPS games, such as allowing custom weapon choice, enabling ADS, displaying hit indicators and showing players' ping times on the scoreboard. TTK being irrelevant also means that it is not necessary for all players to have the same total health, which allows armors, health pickups and healing.

6.2 Effects of RTTs and Movement Speeds

SBC occurs because a low-ping victim is shot by a high-ping shooter. Although it can also be caused by the victim's own latency given that he is sufficiently lagged, we focus only on the former case as explained in Section 2.1.1. If the victim has moved into any cover when he receives the damage, it would appear to him that he has been shot behind covers.

The further in distance the victim is into the cover, the more obvious the problem of SBC should be. We know that distance is the product of time and speed. In the domain of lag compensation, we can say that the rollback distance is the product of RTT and movement speed. Theoretically a victim should be rolled back further if the shooter's RTT increases and/or the victim is moving faster. In practice, however, players do not always move in the same speed (or direction). Rolling a player back also does not always lead to SBC because the victim may not be in any cover when shot. On top of these, player behavior can change drastically depending on the match settings.

This experiment attempts to investigate how RTTs and movement speeds affect

Table 6.2: Combinations of RTTs and movement speeds used in the experiment.

RTTs (ms)	Movement speeds (m/s)	Maximum rollback distance (m)
0	4	0
50	4	0.2
100	4	0.4
150	4	0.6
200	4	0.8
250	4	1
225	4.5	∼1 (1.0125)
200	5	1
180	5.5	∼1 (0.99)
150	6.5	∼1 (0.975)

the perception of SBC in real situations. We look at their effects on the distances which players are rolled back by TLC. Also examined are the correlations among rollback distances, perceived SBC frequencies and fairness. This addresses the last question asked in Section 1.1.

6.2.1 Experiment Settings

In this experiment, all common FPS game mechanics and features are enabled, as we have previously verified that they do not significantly affect the perception of SBC. Because the movement speed is an independent variable, we equalize the crouched movement speed and the walking speed. Sprinting is also disabled.

We invited 4 experienced FPS players (each has 350+ to 1450+ hours of playtime in FPS games) who are all males in their 20s to participate in a series of matches, each lasting for 10 minutes. The game mode is Team Deathmatch, in which players are divided into two teams, and the purpose is to get the highest number

of kills before the end of the match. Each match is configured with one of the combinations of RTTs and movement speeds from Table 6.2. To avoid players getting used to the gradual increase of each independent variable, we randomly selected the order of the combinations. At the end of each match, we asked each player the following questions:

1. Do you think you were treated fairly in the last round? (1 (not at all)-2-3-4-5 (very much))

2. How often do you think you were shot around a corner/behind a cover in the last round? (1 (not at all)-2-3-4-5 (very often))

The RTTs are selected based on some limits found in the modern FPS games and the delay sensitivities measured in previous work as reviewed in Section 3.1. As mentioned in Section 3.2, 250ms is the RTT cap on lag compensations in Battlefield 4 and Overwatch. In Counter-Strike: Global Offensive (CS:GO), by default, a player would only be assigned to a match with less than 150ms.

Similarly, the movement speeds are chosen based on the common values found in games. For example, 4 m/s is the walking speed in Battlefield 4 [46], and in CS:GO for some weapons [47, 48]; 5.5 m/s the base movement speed in Overwatch [30]; and 6.5 m/s the sprinting speed in Battlefield 4 [46].

Table 6.2 also shows the theoretical maximum distance that a player can be rolled back by lag compensation. It is the product of the RTT and movement speed. The top 6 combinations are used to investigate the effects of different rollback

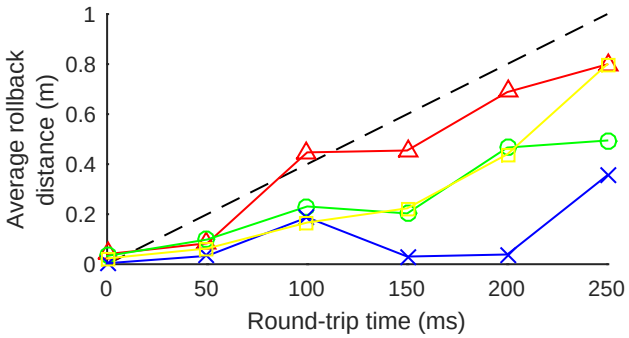


Figure 6.4: Average rollback distances for different RTTs with a movement speed of 4m/s.

distances. The bottom 5 combinations are used to test our hypothesis that the perceived SBC frequencies and fairness are influenced by rollback distances only.

6.2.2 Results and analysis

We first analyze the rollback distances for different RTTs and movement speeds. Here the average rollback distance is the average value for a player, and the grand average rollback distance is that for all players.

6.2.2.1 Rollback distances

Figure 6.4 shows the average rollback distance for each player over different RTTs with a movement speed of 4m/s. Figure 6.5 shows the grand averages. The black dashed line denotes the theoretical maximum rollback distance. Some values exceed the theoretical maximums because lag compensation is susceptible to "small

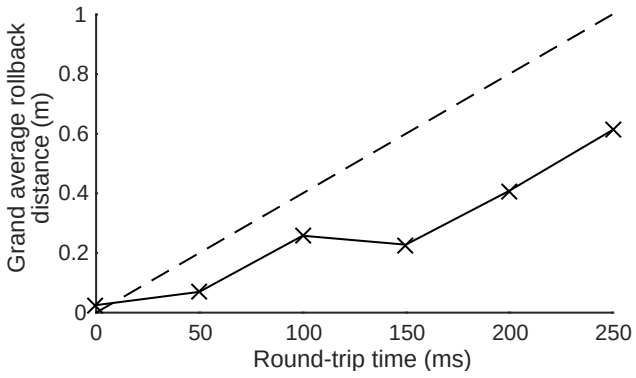


Figure 6.5: Grand average rollback distances for different RTTs.

precision errors in time measurements" [44]. The natural network latency between the server and each player is around 8ms. During the experiment, it could fluctuate up to a few more milliseconds. This translates to a few centimeters of negligible discrepancies shown in the figure.

Some average rollback distances are only around half of, or even much less than, the maximum values. To explain this, we have to first understand the three cases of SBC.

1. The victim moves perpendicular to the shooter. This maximizes the rollback distance.

2. The victim does not move, or moves only directly towards or away from the shooter. The victim in this case is much less likely to notice the problem of SBC [5].

3. The victim peeks and crouches behind a cover. SBC is perceivable in this case, because it is possible for the victim to be rolled back to a previous

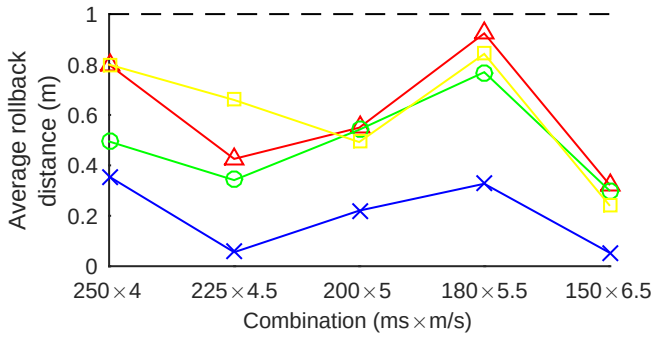


Figure 6.6: Average rollback distances for the last 5 combinations.

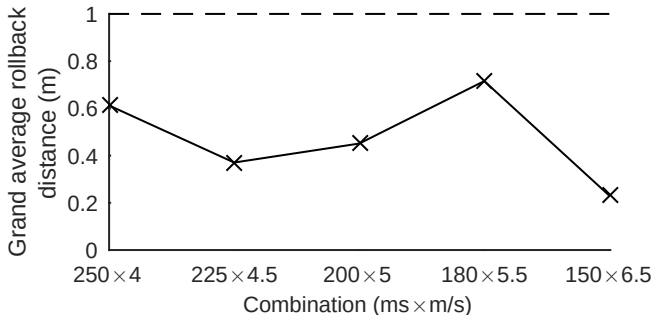


Figure 6.7: Grand average rollback distances for the last 5 combinations.

state in which he is still standing behind the cover, exposing him to the line of sight of the enemy.

For experienced players, based on our observation during the experiment, the second case is rare. It is usually the last case that causes the low average rollback distances. Another reason is that players are not moving all the time, or not in the same direction, during which the shooter's action travels to the server.

Figure 6.6 shows the average rollback distances for the last 5 combinations, each having a different RTT and movement speed but the same maximum rollback distance of around 1m. Figure 6.7 shows the grand averages. We compute the Pearson product-moment correlation coefficient r to identify the correlation be-

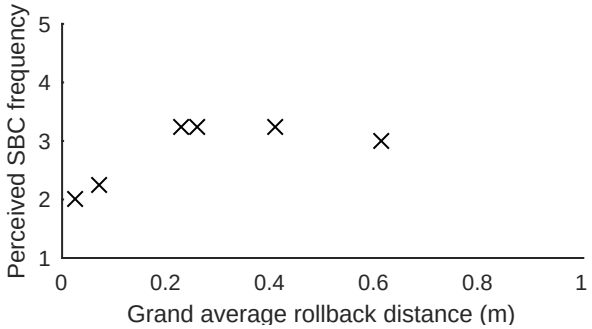


Figure 6.8: Perceived SBC frequencies for different grand average rollback distances.

tween grand average rollback distances and the last 5 combinations. The value of r can range from +1 to -1, where +1 indicates a perfect positive correlation, 0 for no correlation at all, and -1 for a perfect negative correlation. There exists a moderate negative correlation between the grand average rollback distances and movement speeds ($r = -0.45$).

6.2.2.2 Perceived SBC frequencies

We now study the perceived SBC frequencies for different rollback distances and movement speeds. Figure 6.8 shows the perceived SBC frequency for different grand average rollback distances, which exhibits a relatively strong positive correlation ($r = 0.67$). While the maximum rollback distance is 1m, the longest actual grand average rollback distance for the first 6 combinations is around 0.61m, which can also be observed in Figure 6.5. The chance of a player being shot behind covers increases with his average rollback distance. The positive correlation suggests that players, or at least experienced players, do notice how often they are shot behind covers.

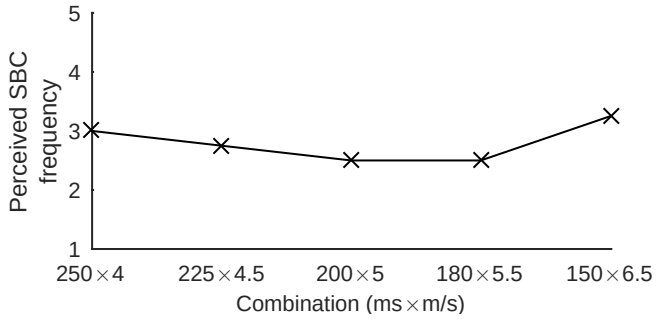


Figure 6.9: Perceived SBC frequencies for the last 5 combinations.

The perceived frequency plateaus when the grand average rollback distance reaches around 0.23m. The value does not go all the way up to 5 because not every rollback causes an instance of SBC. A player can be rolled back without being shot behind covers if he is moving along the shooter's line of sight, not moving at all (the second case discussed in Section 6.2.2.1), or not even behind a cover before being rolled back.

Figure 6.9 shows the perceived SBC frequencies for the last 5 combinations, which do not exhibit a significant correlation ($r = 0.28$). This is expected, because each combination has the same theoretical maximum rollback distance of around 1m.

6.2.2.3 Perceived fairness

Here we examine how player perception of fairness changes with different average rollback distance and perceived SBC frequency. We also conduct repeated measures analysis of variance (rANOVA) to find out if the differences in the perceived fairness are statistically significant.

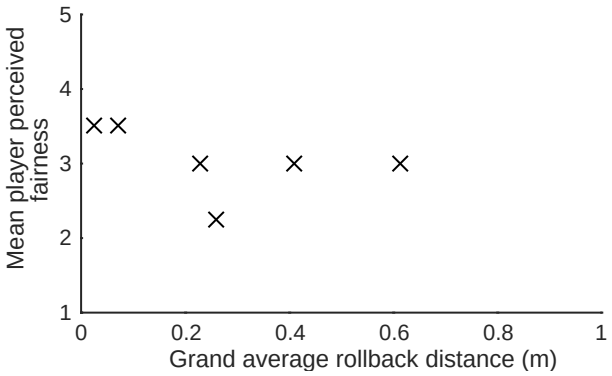


Figure 6.10: Mean player perception of fairness for different grand average rollback distances.

Figure 6.10 shows the mean player perceived fairness for different grand average rollback distances, which exhibits a moderate negative correlation ($r = -0.43$). The mean perceived fairness reaches its minimum when the grand average rollback distance is 0.26m, similar to the value that maximizes the perceived SBC frequency (0.23m). This may implicate that at least for experience FPS players and a movement speed of 4m/s, SBC impacts gameplay experience the most when the rollback distance is around 0.23m to 0.26m. This distance range may be similar to how far players usually duck into covers, so that their perceptions do not get worse with higher rollback distances. Rolling players back by 0.23m to 0.26m may already guarantee to "move" them out of their covers.

The weakness of this comparison is that again not every rollback results in an instance of SBC. To better evaluate how the player perceived fairness is affected by SBC, it is more appropriate to compare the mean perceived fairness with different perceived SBC frequencies. We still show the previous comparison because it is useful for determining the limits imposed on lag compensation.

The player perceived SBC frequency is difficult to be used as a limit on lag

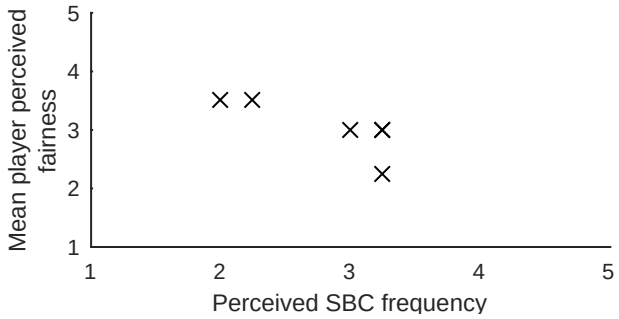


Figure 6.11: Mean player perceived fairness for different perceived SBC frequencies.

compensation. Doing so would mean that the limits are only effective once a player has been shot behind covers for a certain frequency. Figure 6.11 shows that the negative correlation between the mean player perceived fairness and the perceived SBC frequencies is much more pronounced ($r = -0.79$).

The rANOVA results show that the effect of rollback distances up to 0.61m on player perceived fairness is insignificant ($F(5, 15) = 1.91, p = 0.247$). This suggests that a limit of 250ms on lag compensation is reasonable, at least for a movement speed of 4m/s.

As previously mentioned, however, some players were rolled back by TLC for only around half of, or even much less, than the theoretical maximum values. Our conclusion about the 250ms limit can change when the two values are closer, as rollback distances do affect player perceived fairness. This can also pose a problem for investigations on rollback distances, because manipulating network latency and movement speeds can not guarantee rollback distances nearly as much as the theoretical maximum values in reality. To better reconcile the two values, the game has to encourage players to move around the level more. It may be achieved by spawning health and ammo pickups around the level from time to

time, or even by using a different game mode. As mentioned in Section 6.2.1, the game mode used in this experiment is Team Deathmatch (TDM). The objective is to acquire the most kills as a team at the end of the match. Because of how TDM works, members of a team that is in the lead may adopt a more passive strategy by camping around covers. For similar experiments, game modes such as Domination and Capture the Flag (CTF) may be better candidates. In Domination, teams score by gaining and maintaining controls over capture points around the level. In CTF, teams score by capturing randomly spawned flags and returning them to their bases. Both game modes are excellent for encouraging movements because players themselves are no longer the objectives. To win the match, players have to actively compete for external objectives. This removes the incentives for them to hide.

Chapter 7 Advanced Lag Compensation

After establishing the need for improving TLC as well as understanding the drawback of CLC, the perception of SBC and its effects, we come up with a novel lag compensation algorithm. Based on TLC, advanced lag compensation (ALC) attempts to mitigate the problem of SBC while retaining the benefits of lag compensation for *all* players most of the time. The first part of the chapter discusses the algorithm of ALC in details. This is followed by an evaluation of ALC. ALC fulfills both the criteria we have established in Chapter 2 for a good lag compensation algorithm. It achieves a similar improvement in shooting accuracy as TLC does, while the inconsistency introduced has no significant perceptual impacts on the players.

Illustrated in Figure 7.1, at 275ms, the server still uses what Alice sees at 150ms to determine the outcome of her shot. If Alice's shot is on target, Bob will still receive a hit at 300ms. However, this time the hit is only provisional. Bob is given a chance to appeal the decision of the server through a confirmation message. Bob's local game client either confirms or denies the hit in its confirmation message, depending on whether it detects an SBC or not. Besides the confirmation message, Bob also updates the server on his latest location, just as what he does at every tick.

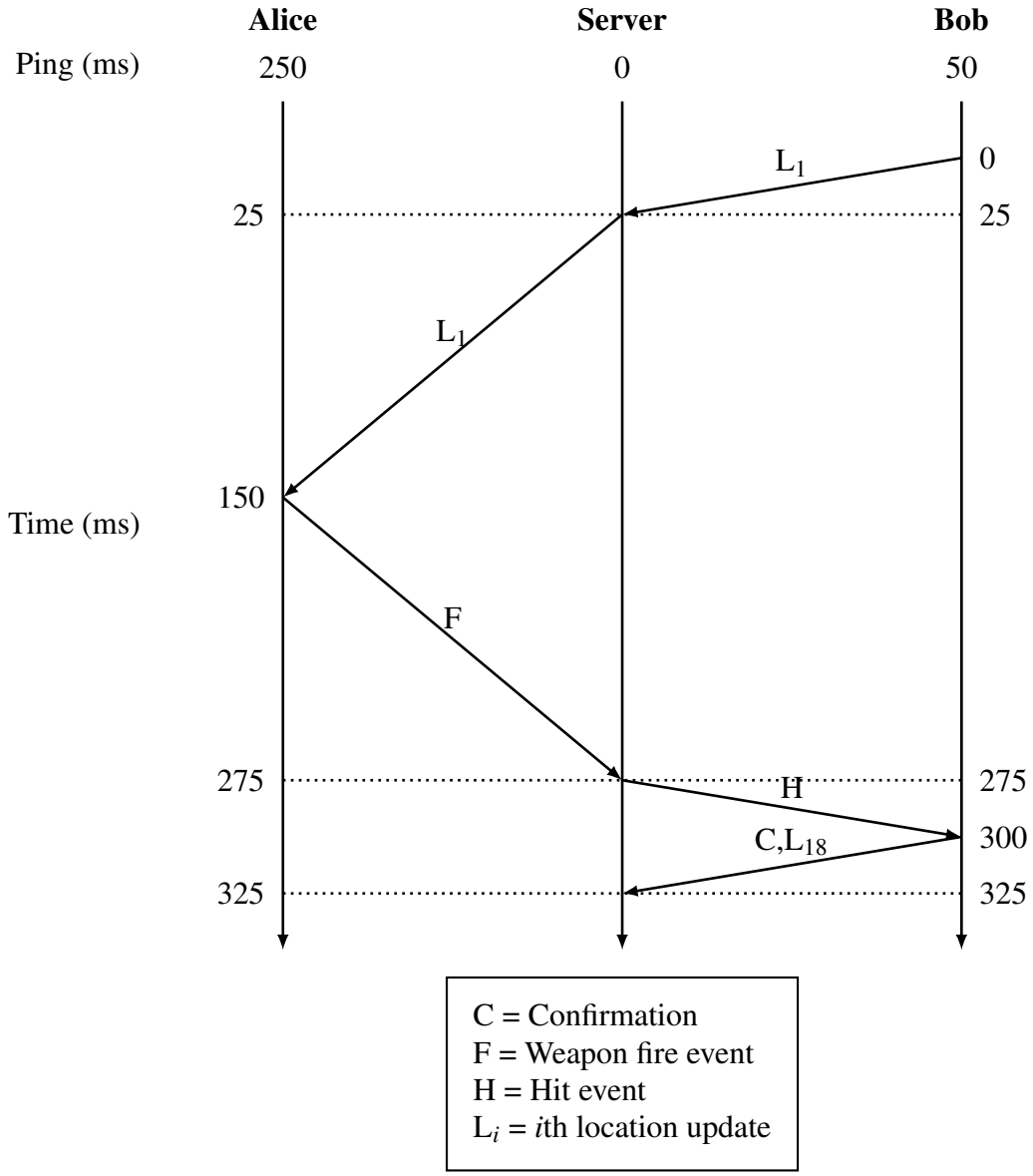


Figure 7.1: Timeline diagram for ALC before the server responds.

When the server receives the confirmation message, it has to decide whether to trust Bob's claim or not. Three scenarios can happen here.

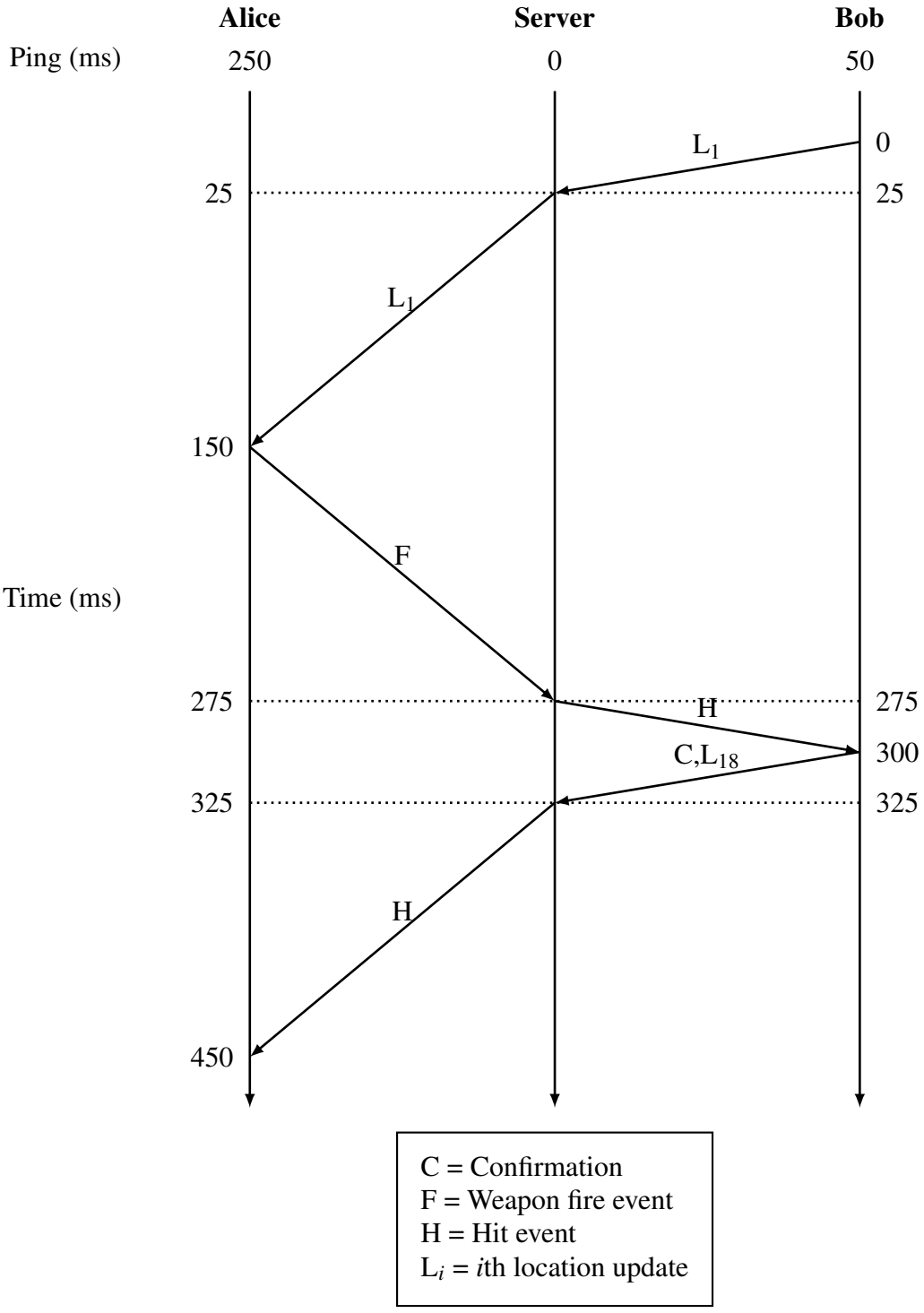


Figure 7.2: Timeline diagram for ALC in which the victim confirms a hit.

Hit confirmed If Bob confirms the hit, the game will show the damages without any further delay, just like the normal TLC. The server can then send the hit event also to Alice, as shown in Figure 7.2. Notice that Alice's hit event is delayed by 50ms, the ping of Bob. Therefore, if the shooter eventually gets a hit, the message is delayed additionally by the ping of the victim. This delay, however, can be limited to a certain value. We will give further details on why this is necessary after presenting all three scenarios.

Hit denied correctly If Bob denies the hit, the server has to verify the claim. It does so using the location update that is sent along the confirmation message. The location derived by the server from that update is also what Bob's denial bases on. The server then runs the same SBC detection as Bob's client did to verify his claim. If the server also detects SBC, Bob's denial is verified. The diagram for this case is identical to Figure 7.1, since no extra operation is needed.

Hit denied incorrectly If the server does not detect any SBC, Bob's denial is refuted. In this case, the server sends the hit event also to Alice and to Bob again, as illustrated in Figure 7.3. This time the hit is irrefutable. Notice that the hit event to both players is delayed by 50ms, the ping of Bob. This scenario, however, should be the least common. The reason is that an incorrect denial happens only if there are "small precision errors" in hit detection [44] or the victim is cheating.

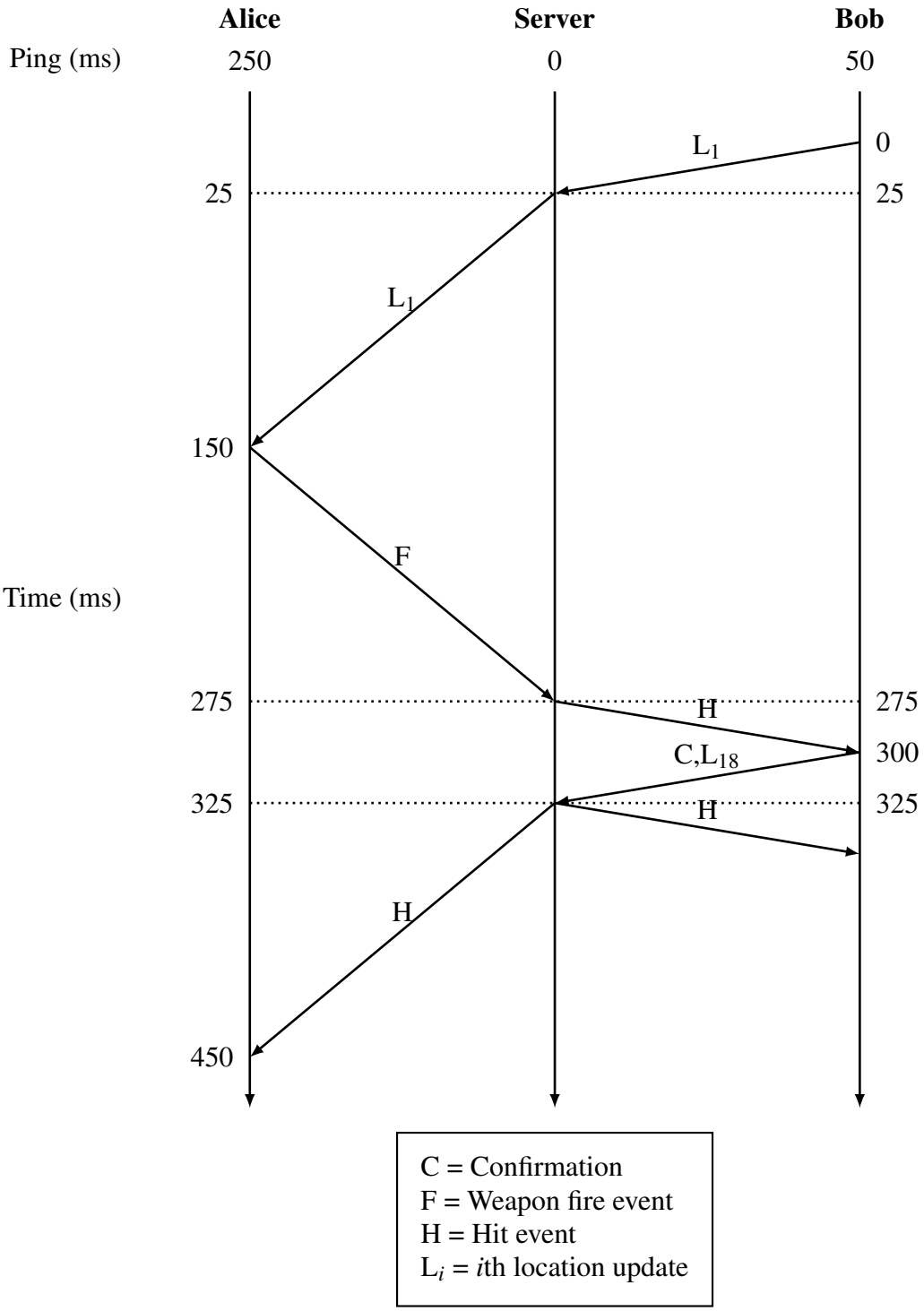


Figure 7.3: Timeline diagram for ALC in which the victim denies a hit incorrectly.

Algorithm 4 Advanced lag compensation

```
1: procedure ServerTick
2:     for all players do
3:         if length of player's frames = maxFrameSize then
4:             remove the first element of player's frames
5:         push player's states into player's frames
6:
7: procedure ServerReceiveFire(shooter)
8:     frameIndex = maxFrameSize−round(shooter's ping/(1000/tick rate)−
   1
9:     for all players do
10:        rollback player's states to player's frames[frameIndex]
11:    start ← shooter's camera location
12:    end ← start + shooter's aim direction × trace distance
13:    perform a collision trace from start to end
14:    if hit detected and hit actor is a player then
15:        target ← hit actor
16:        if shooter can damage target then
17:            canAppeal = target's ping is within lowPingValue
18:            Call ClientReceiveFire(canAppeal) on target
19:            if not canAppeal then
20:                send result to shooter
21:    for all players do
22:        restore player's states
23:
24: procedure ServerReceiveConfirmation(confirm)
25:    if confirm then
26:        send result to shooter
27:    else
28:        if not target is shot behind covers then
29:            Call ClientReceiveFire(false) on target
30:            send result to shooter
31:
32: procedure ClientReceiveFire(canAppeal)
33:    if canAppeal then
34:        if client is shot behind covers then
35:            ServerReceiveConfirmation(false)
36:        else
37:            ServerReceiveConfirmation(true)
```

7.1 Appeal Eligibility

With ALC, if the shooter eventually gets a hit, the corresponding event(s) is delayed by the ping of the victim. This is the case for scenario 1 and 3. In our example, the victim is only slightly lagged so the hit event(s) is delayed only by a relatively small amount of time. If it is Bob who shoots at Alice, however, his hit event will be delayed by 250ms, even though his ping is only 50ms.

Illustrated in Figure 2.3, imagine Bob fires at Alice this time. When the server notices the event at 175ms, it determines the outcome using the game world at 125ms (the first circle). If the hit event were to arrive at Alice instantaneously, Alice may not suffer from SBC at that moment (the second circle). However, due to Alice's high ping, by the time the hit event arrives at Alice at 300ms, she may see herself being shot behind covers. The cause of Alice's SBC is therefore different from that of Bob. Bob's SBC is caused by the shooter's high ping; Alice's SBC, however, by her own high ping.

The main purpose of ALC is to protect less lagged players, like Bob, from being shot behind covers. ALC does not protect highly lagged players like Alice because of the following reasons:

- It would be unfair to the less lagged shooters.

- Less lagged shooters are less likely to cause their victims to be shot behind covers.

- Highly lagged victims may suffer from SBC due to their high pings anyway.

Because of the above reasons, ALC allows developers to specify a low-ping threshold. A hit event can be appealed only if the victim's ping is within the threshold. In our example, the value is set to 50ms. This means that only Bob in our example could appeal a hit should his client detect an SBC.

Player sensitivities concluded from some of the previous works we have reviewed in Section 3.1 can help choose a low-ping value in ALC. ALC protects players with an RTT not higher than the low-ping value from SBC. The value also determines how long at most hit registrations could be delayed for the shooters if their targets are protected. A good low-ping value should not be higher than most of the tolerance values found in previous studies. We want to cover players with a reasonably low ping while keeping the added delay minimum. Having a relatively high low-ping value is often pointless as well because high-ping players may suffer from SBC due to their own lag anyway (as discussed in Section 2.1.1).

7.2 Advantages over CLC

As mentioned in Section 2.2, a few recent games attempt to mitigate SBC by setting a ping limit on lag compensation. Players with a ping higher than the limit are not lag compensated. They would therefore need to aim ahead of (or lead) their targets in order to actually hit them. This approach is inadequate because pings vary among players and over time, thus making it very difficult to estimate how much a shot should be led. That is why some long-range combat

games, such as War Thunder, in which target leading is expected, include a lead indicator to assist players in hitting the targets. In fact, it is found that the mean shooting accuracy is reduced significantly by 5% as compared to the one without lag compensation [2].

Leading the targets is not needed in ALC. Even players with what is normally considered a high ping are still lag compensated, as long as their low-ping victims are not shot behind covers. Moreover, SBC does not occur if the line of sight between the shooter and the victim remains when the victim receives the hit. This includes the following situations:

- The victim has not moved at all.
- The victim has not moved around a corner or into a cover.
- The victim moves only directly towards or away from the shooter.

In practice, victims are not shot behind covers most of the time, as we will show in the evaluation later.

7.3 SBC Detection

The benefit of detecting SBC on the client's side first is that no extra operation is required from the victim if he confirms or correctly denies the hit. As previously

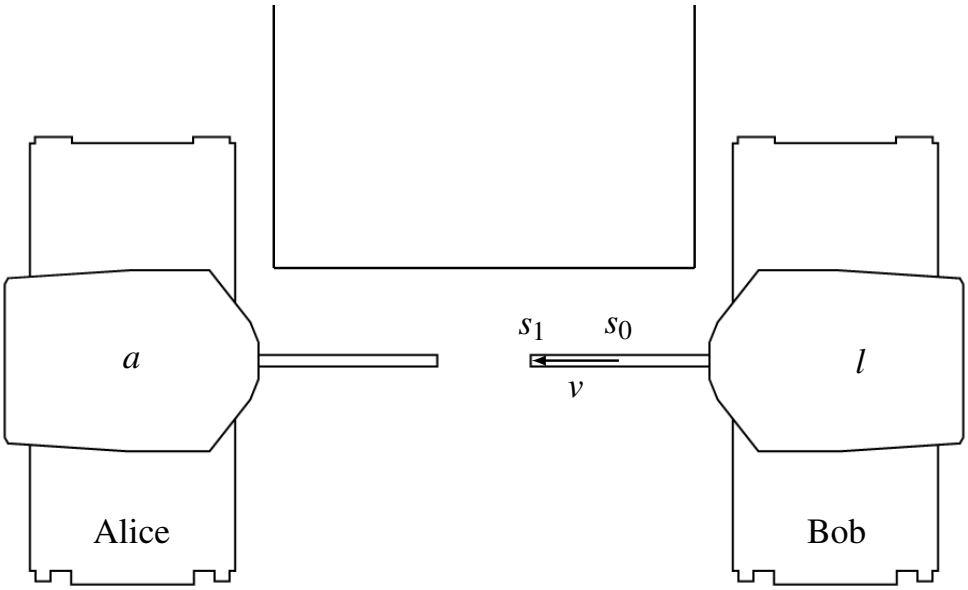


Figure 7.4: SBC detection before Bob receives a hit event from Alice.

explained, these should be the most frequent cases. In these cases, having the victim detect SBC locally first can save an RTT which is his own ping, because he does not need to wait for the server to answer the confirmation message.

Now we explain how SBC can be detected. The same detection algorithm is run on both the victim and the server. We use the scenario illustrated in Figure 7.1 again as an example. For the benefit of the explanation, assume both Alice and Bob control their own tanks depicted in Figure 7.4.

At 150ms, Alice fires at Bob's cannon at s_1 (Figure 7.4). At 275ms, the server rolls back all players behind the scenes to 25ms and detects a hit from Alice to Bob. The hit event sent to Bob identifies the shooter and which body part is hit. In this case Alice is the shooter and Bob's cannon is the body part. It also contains a vector v that represents the offset of the impact location s_1 from the

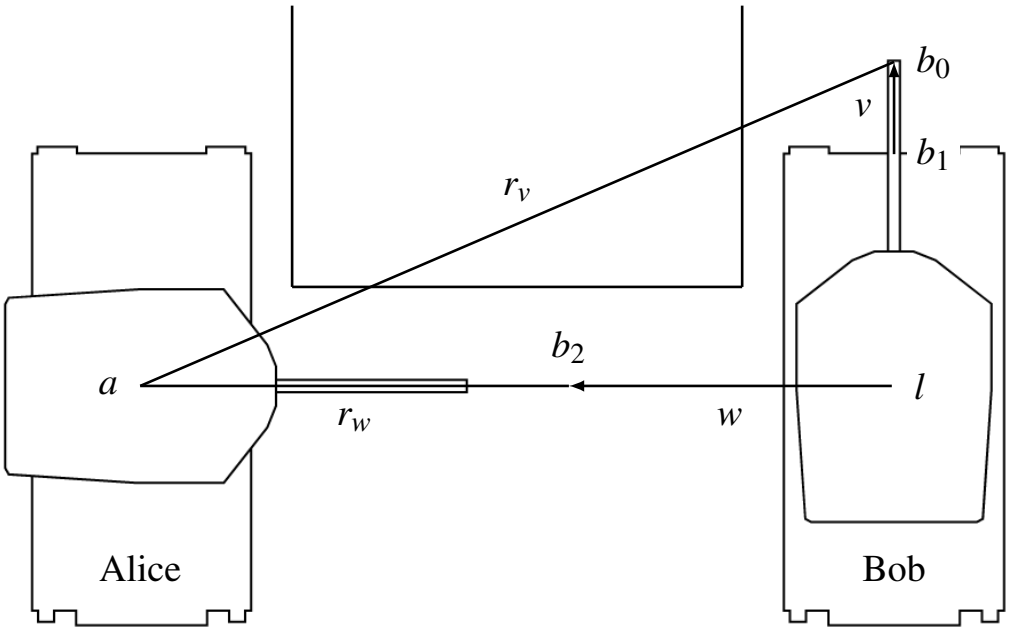


Figure 7.5: SBC detection after Bob receives a hit event from Alice.

center of that body part s_0. In other words, $v = s_1 - s_0$.

Before the hit event arrives at Bob, he has already turned his cannon to a position that is out of Alice's line of sight (Figure 7.5). When Bob receives the hit event at 300ms, he rotates v by the rotation of his cannon before adding it to the cannon's current location b_0 to get b_1. A ray r_v is then traced between b_1 and Alice's camera location a, which is replicated from the server through location updates. If the ray intersects any entity other than Alice and Bob, an SBC is detected. Bob's game client then denies the hit through the confirmation message.

When the server receives the appeal from Bob, it runs the same algorithm using the values it originally sends to Bob. Since the server now has all the locations of Bob's body parts at the time he denies the shot, he can verify Bob's claim.

One may question the choice of using the center of the hit body part instead of just the victim's location as the reference point. Assume instead of v, we now use w, which equals to the offset of the impact location s_1 from Bob's location l. When Bob receives the hit event at 300ms, he adds w to his current location l to get b_2. A ray is then traced between b_2 and Alice's camera location a. In this case, the algorithm fails to detect an SBC. Bob's game client then confirms the hit and his cannon is now ruined. The detection fails because Bob has moved only his cannon but not the tank itself.

In more technical terms, we want to include the displacement between the impact location and the corresponding rigid body in the hit event. A rigid body is a solid body that does not undergo deformation [49]. Examples are the left forearm of a player (but not the whole arm) and the cannon of a tank. This approach can minimize the number of false positives in SBC detections. It also makes sense in video games since damages usually vary depending on which body part is hit.

7.4 Cheating

The design of ALC takes into account the possibility of cheating. A cheater may try to avoid being damaged by denying every shot that he receives. This is not a problem because when the server receives a deny message from the victim, it runs the SBC detection algorithm using his location sent along with the message. The server should then be able to discover that the cheater is not actually shot behind covers. An irrefutable hit event is then sent to the shooter and the victim. ALC should therefore be resilient to this type of cheating assuming that the game can

prevent players from cheating their locations.

Instead of outright denying each received shot, a cheater may attempt to stop the game client from responding to the server with a confirmation message. Because a player can appeal a shot only if his lag is within the low-ping value, a confirmation message should arrive at the server also approximately the same amount of time after the player is notified of the hit. In this case, a player can be assumed to confirm the hit if a response is not received after a time equals to double the low-ping value has elapsed after the hit event is dispatched.

7.5 Performance

The way we implement ALC in our game, the only extra non-trivial task generated by the algorithm is an extra line trace when a shot is denied by the victim. This is because the server has to run hit detection again with the latest victim's location to verify the claim. In our experiments where each match consists of 6 players, an average of 68.4 ± 3.11 ticks elapsed between each shot. According to the developers of Unreal Engine 4, at least hundreds of line traces can be performed every tick on modern hardware without perceivable performance hits [50]. The extra processing caused by ALC should therefore be negligible even for matches with around 100 players or if every shot is denied due to cheating.

As for the network traffic, the effects of ALC can be different depending on which of the three previously discussed scenarios dominates during a match. If a

shot is confirmed by the victim, which should usually be the case, only an extra confirmation message is generated in the process. In our game, a confirmation message is similar in size to a hit event. If a shot is incorrectly denied, the server has to also resend the hit event to the victim. If a shot is correctly denied, there is actually less traffic since the server does not have to send the hit event to the shooter.

ALC can be modified to save network traffic at the expense of accuracy. After detecting a hit, the server can predict where the victim will be when he receives the event. A hit is discarded if it is expected to result in an SBC. This eliminates the need for confirmation messages.

7.6 Evaluation

7.6.1 Experiment Design and Settings

We design the experiments to answer the following questions:

1. How much can ALC reduce the frequency of SBC?

   The main goal of ALC is to protect less lagged players from being shot behind covers. We therefore want to find out how effective ALC can reduce both the actual and perceived SBC frequency for low-ping players.

2. Is ALC as effective as TLC in mitigating the effects of latency?

   In ALC, high-ping players will have their hits rejected if they cause SBC to low-ping players. We believe this accounts for only a small number of all the shots, and should not have a significant impact on high-ping players. This question attempts to verify this belief.

3. Are hit registrations in ALC as responsive as in TLC?

   In ALC, hit registrations for high-ping players may be delayed by the RTT of the low-ping targets. We use this question to find out if the additional delays introduced by ALC have a significant impact on high-ping players.

4. How does ALC affect the overall fairness of the game?

   Since ALC protects low-ping players from being shot behind covers, obviously we want to find out whether ALC helps increase their perception of fairness of the game. On the other hand, we also want to make sure that the misses and delays introduced to high-ping players do not impact their perception of fairness significantly. We answer this question based on each player's rating on the fairness of each match.

To answer these questions, we design the experiment settings as follows. CTF is used as the game mode, because we believe it best encourages player movement. In fact, it is found that players in TDM generally do not move around very much [3]. This may reduce the frequency of occurrence of SBC, because the line

of sight between the shooter and the victim is less likely to be broken. A total of five matches, each lasting for 10 minutes, are played:

1. 10-minute warm-up.

2. 10 minutes under TLC, a random half of each team's players are injected with a latency of 250ms.

   A 10-minute match is long enough for each player to fire a sufficient number of shots (a few hundreds). It is also short enough to avoid the boredom effect and impacting players' ability to recall inconsistencies closer to the start of the match.

   We inject latency only to a subset of players so that there are always low-ping players and high-ping players in the same match. We also do not inject latency to only one team at any given time in order to minimize potential differences caused by team compositions. We choose to inject 250ms, because, as mentioned in Section 2.2, this value is considered to be the threshold for a high ping by both Battlefield and Overwatch. Moreover, 250ms is higher than the latency tolerance of FPS players found in most of the previous studies (see Section 3.1). This suggests that the extra delay caused by ALC should not significantly reduce the responsiveness to hit registrations.

3. 10 minutes under ALC. Latency is injected to the same set of players in the previous round (round 2).

The low-ping value of ALC is set to 50ms. This choice is based on the fact that Battlefield considers players with a ping of 50ms or below to have a good connection [21]. This value is also lower than all latency tolerance of FPS players found in previous studies (see Section 3.1).

4. Another 10 minutes under ALC, but with latency injected to the opposite players.

5. 10 minutes back under TLC. Latency is injected to the same set of players in the previous round (round 4).

In these experiments, players are classified into two groups: those injected with latency and those are not. The first group of players plays under TLC and then ALC. The order for the second group is reversed, i.e., ALC followed by TLC. This design is known as counterbalancing [51], which helps minimize the learning and boredom effects.

At the end of each match except the warm-up, we ask each player the following questions:

1. How often have you been shot even when you have already taken cover? (1 (not at all)-2-3-4-5 (very often))

2. How accurate do you think your shots were registered? (1 (not at all)-2-3-4-5 (very much))

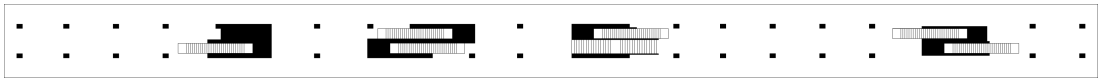


Figure 7.6: Layout of the map. Obstacles are colored in black.

3. How responsive do you think your shots were registered? (1 (not at all)-2-3-4-5 (very much))

4. How fairly do you think you were treated in the last match? (1 (not at all)-2-3-4-5 (very much))

The use of a post-hoc questionnaire requires players to remember their gaming experience over a 10-minute match, which may not be easy to some players. We therefore also include in the game a key which players can press to report an instance of SBC as soon as it occurs. The number of reported SBC instances, along with the scores given to the first question, should paint a more accurate picture of the SBC frequency.

7.6.1.1 Map

The map used in the experiments is a remake of a real metro station in the city where the authors reside. Figure 7.6 shows the layout of the map. Walls and pillars are colored in black and striped rectangles represent escalators. Since all players in the experiments are already familiar with the layout of the station, the time needed for them to warm up is minimal.

The map also has benefits for investigating SBC in TLC and evaluating the performance of ALC. SBC is most noticeable when players confront each other directly. The map is therefore designed to contain only the underground level of the station. This removes the verticality and flanking opportunities from the map, thus encouraging direct confrontations. The map also promotes heavy uses of covers with all the pillars and various structures populated around the station.

In terms of size, the map is relatively small. The close-quarter nature of the map is particularly suitable for experimentation purposes. Players wandering around the map without being involved in any engagement is not really useful to our evaluation.

The symmetrical nature of a metro station provides a good team balance for various game modes. The spawn of each team situates at either end of the station. In Domination, for example, three control points can be easily placed on the map with a good balance: one outside each team's spawn and the remaining one at the center of the map.

7.6.1.2 Players

We invited 12 players, who are all males in their 20s, to participate in the experiments (we have difficulty finding female players). Two sets of experiments were conducted, each consisting of 6 players and lasting for 50 minutes.

	TLC			ALC		
Player #	Hits	SBC	%	Hits	SBC	%
1	95	11	11.6%	113	0	0%
2	79	9	11.4%	66	1	1.52%
3	95	4	4.21%	126	2	1.59%
4	100	13	13%	96	0	0%
5	101	16	15.8%	104	1	0.96%
6	82	4	4.88%	92	0	0%
7	93	7	7.53%	97	0	0%
8	89	8	8.99%	88	0	0%
9	101	6	5.94%	92	0	0%
10	76	2	2.63%	77	0	0%
11	125	2	1.6%	103	0	0%
12	85	3	3.53%	85	1	1.18%
Total	1121	85	7.58%	1139	5	0.44%

Table 7.1: Number of hits and SBCs received by low-ping players.

7.6.2 Results and Analysis

7.6.2.1 SBC frequency

Here we find out the answer to our first question: how much does ALC reduce the frequency of SBC? To begin with, we count the numbers of SBC for low-ping players while using TLC and ALC. To calculate the number of SBC for a player in ALC, we use the number of times the server overrules the player's denial of a hit. Detecting SBC on the player's game client locally is the most accurate method, since it uses the same data that renders what the player sees. When the server rejects a player's appeal, the player eventually has to accept the SBC hit.

As shown in Table 7.1, we record 85 SBC for low-ping players under TLC, which

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	2	4	4	2	4	2	1	2	3	3	4	4
ALC	2	1	3	2	2	2	1	1	4	2	4	2

Table 7.2: Perceived SBC frequency of low-ping players.

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	3	4	1	2	11	0	2	2	1	3	0	0
ALC	1	3	0	2	5	1	0	0	0	1	0	0

Table 7.3: Number of SBC reported by low-ping players.

accounts for 7.58% of the total number of hits they receive. This confirms our previous belief that the majority of hits is not SBC, and the most frequent case of ALC is indeed the one in which the victims simply confirm the hits. ALC therefore should not delay victims' hit notifications in most cases. The result also reinforces what we have argued when explaining the advantages of ALC over CLC. It is better to lag compensate even high-ping players and reject their hits only if SBC occurs.

ALC reduces the total number of SBC by 94.1% to just 5, which accounts for 0.44% of the total number of hits received by low-ping players. A Shapiro-Wilk Test confirms that the differences in the number of SBC between TLC and ALC are normally distributed ($p = .14$). A dependent t-test then shows that ALC significantly reduces the number of SBC from 7.08 ± 4.50 to $.42 \pm .67$ ($p < .0005$), a reduction of 6.67 ± 4.54.

To find out how much ALC reduces the perceived SBC frequency, we first look at the number of SBC reported by low-ping players during matches (Table 7.2). A Wilcoxon signed-rank test shows that the number is significantly reduced when ALC is used ($Z = -2.412, p < .05$). We then examine the answers to the first

Player #	TLC	ALC
1	10.7%	8.48%
2	6.94%	7.42%
3	6.98%	6.57%
4	11.2%	10.2%
5	9.71%	8.29%
6	10.5%	10.2%
7	9.94%	7.64%
8	6.78%	7.21%
9	9.94%	7.93%
10	6.52%	6.41%
11	5.55%	4.55%
12	9.66%	12.7%

Table 7.4: Hit accuracy of high-ping players.

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	5	2	2	4	4	3	4	3	3	1	3	2
ALC	4	2	1	4	4	3	3	2	3	2	3	2

Table 7.5: Perceived hit accuracy of high-ping players.

QoE question given by low-ping players (Table 7.3). It shows that ALC also significantly reduces their perceived SBC frequency ($Z = -1.983. p < .05$).

7.6.2.2 Hit registration accuracy

In our second question, we want to find out whether ALC mitigates the effects of latency as effectively as TLC. To answer that, we look at the hit accuracy of all players. The goal of lag compensation is to eliminate the need for players to aim ahead of their targets. If ALC results in a similar level of hit accuracy as TLC does, we therefore conclude that both of them are effective.

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	4	2	2	4	4	4	4	3	3	1	3	3
ALC	4	2	1	3	5	3	2	3	3	1	4	3

Table 7.6: Perceived hit registration responsiveness of high-ping players.

A Wilcoxon signed-rank test shows that the difference in hit accuracies under ALC and TLC (Table 7.4) does not demonstrate a statistical significance ($Z = -1.412, p = .158$). We then look at the difference perceived by the players with their answers to the second QoE question (Table 7.5). A Wilcoxon signed-rank test again shows no statistical significance ($Z = -1.342, p = .18$). We can conclude that ALC is as effective as TLC in mitigating the effects of lag.

7.6.2.3 Hit registration responsiveness

As we have discussed, if a shooter eventually hits in ALC, the hit registration is delayed by the RTT of the victim. A low-ping value of 50ms means that hit registrations can be delayed by up to 50ms. In the experiments, however, most hit registrations should be delayed by 20ms at most, since the existing lag between each client and the server is around 16ms to 20ms.

Here we answer our third question by looking at the hit registration responsiveness perceived by high-ping players (Table 7.6). A Wilcoxon signed-rank test shows that the difference under TLC and ALC does not demonstrate a statistical significance ($Z = -1, p = .317$). This means that high-ping players do not experience significantly slower hit registrations under ALC. The result is expected. As we have mentioned before, 250ms is higher than the latency tolerance of FPS players found in most previous studies. A further 20ms increase in hit

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	4	1	4	5	2	2	1	4	2	2	3	3
ALC	3	3	4	5	3	2	4	4	1	3	1	4

Table 7.7: Perceived fairness of low-ping players.

Player #	1	2	3	4	5	6	7	8	9	10	11	12
TLC	4	2	2	3	4	3	5	3	4	1	3	3
ALC	4	2	1	4	3	2	4	3	2	3	4	3

Table 7.8: Perceived fairness of high-ping players.

registrations should not make a significant difference.

7.6.2.4 Fairness

Our final question aims to discover how ALC affects the overall fairness of the game. To answer that, we look at the scores given by both low-ping (Table 7.7) and high-ping (Table 7.8) players in the final QoE question.

For low-ping players, we are interested in finding out whether ALC will significantly increase the fairness, because it eliminates a majority of SBC. A Wilcoxon signed-rank test shows no statistically significant change in fairness for low-ping players when ALC is used ($Z = -.791, p = .429$). This result is unexpected but not surprising. The perceived fairness of a match can be affected by many factors besides lag compensation. For this question, it is more important to ensure that ALC does not significantly impact the overall fairness perceived by high-ping players.

For high-ping players, we look for a similar level of fairness for TLC and ALC. Since ALC does not benefit high-ping players in any way, we just need to make sure that it does not worsen their perceived fairness. A Wilcoxon signed-rank test shows an insignificant difference in fairness between the use of TLC and ALC for high-ping players ($Z = -.513, p = .608$).

Chapter 8 Conclusion and Future Works

In this dissertation, we present a novel advanced lag compensation (ALC) algorithm. ALC mitigates the problem of "shot behind covers" (SBC), inconsistencies stemmed from traditional lag compensation (TLC). We have built a multiplayer first-person shooter (FPS) game named LAGCOM from scratch and implemented ALC in it. The core of ALC includes an algorithm we have proposed for detecting SBC instances. Comparing with TLC, ALC reduces the total number of SBC by 94.1%. The number of SBC reported and the perceived SBC frequency also see a significant drop. Although ALC may introduce additional delays to hit registrations in some cases, the delays do not impact player QoE significantly. Hit registration accuracy and responsiveness under ALC are similar to that under TLC as well.

Our evaluation reveals that TLC significantly improves the shooting accuracy. This refutes the suggestion by some players to eliminate SBC by abandoning lag compensation. Using LAGCOM, we discover that the perception of SBC is not significantly affected by many FPS game features, such as the field of view, the presence of mini-map and ping time display. These results help decide the experiment settings for future studies on SBC. In our investigation, we confirm that there exists a strong positive correlation between the perceived SBC frequencies and the distances which players are rolled back by TLC. The former in turn has a

significant impact on the perceived fairness of the entire match. We also find that 250ms is a reasonable limit for TLC for a movement speed of 4m/s. This should be helpful to developers who want to incorporate conditional lag compensation in their games.

8.1 Future Extension

ALC mitigates the problem of SBC by allowing low-ping players to appeal shots that they receive. The side effect is an extra delay of up to the low-ping value being introduced to the hit registration process. There is an alternative approach that can eliminate this overhead at the expense of potential SBC detection inaccuracy. Upon receiving a shot, instead of relaying it to the player and waits for a response, the server uses extrapolation to predict where the player will be when he receives the shot. It can then run the SBC detection algorithm with the extrapolated location to guess if the player will be shot behind covers or not. Whether the server registers or discards the shot depends on the prediction but the decision is final. This approach removes the extra delay but extrapolation can often be inaccurate especially in FPS games. Player movements in FPS games can be very random and subject to high changes in acceleration [5]. This drawback may be further exaggerated in games like Overwatch in which players can move in many non-traditional ways such as teleporting, launching into air and charging. Although we have shown that the extra delay does not impact the perceived hit responsiveness significantly, this alternative approach is worth studying nonetheless. It is still better if ALC can achieve a similar reduction in SBC without the extra overhead.

In Battlefield 4 and Overwatch, players with a round-trip time (RTT) beyond 250ms are not lag compensated. Our study suggests this limit to be reasonable for a movement speed of 4m/s. It would be useful to find out if a 250ms limit is still appropriate for movement speeds beyond 4m/s. Future studies can also investigate RTT limits above 250ms. A higher RTT limit means that each server can accommodate more players without them needing to lead the targets. Game developers and publishers can therefore save resources by deploying less servers for the same area.

All experiments in our study use hitscan weapons. The effect of a hitscan weapon is instant because hit detections are performed immediately upon firing. The opposites of hitscan weapons are called projectile weapons. This work can be extended to investigate the perception of SBC when such weapons are used. Because projectile weapons usually have damage falloff instead of a definite hit or miss, the problem of SBC may be less obvious to players damaged by projectiles. Furthermore, ALC can be expanded to compensate projectile weapons as well. We do not consider them in our study because only few games extensively use projectile weapons. Part of the reason is that lag compensation for these weapons can be more complicated as projectiles are now real entities that exist in the game world. Weapons like rockets and grenades are, however, intrinsically projectile. Some games such as Team Fortress 2 that include such weapons choose not to lag compensate them [52].

Besides SBC, kill trading is another problem caused by TLC. It describes situations in which two players kill each other at the same time, even though one of the players should have already been dead. For games that use hitscan weapons, this should rarely happen because the chance of two players firing at the exact

same moment is very slim. Because of TLC, however, kill trading can happen more often. Although kill trading is generally less complained than SBC, it is nonetheless an inconsistency. It would be useful to investigate the problem of kill trading in order to further improve lag compensation.

Lag compensation is not exclusive to FPS games. It would be interesting to look at how SBC is perceived in other genres like third-person shooters (TPS) games. In TPS games, the camera is usually placed behind the player's character. This configuration allows players to observe more of the surroundings than they can in FPS games. As a result, players may be less tolerant of SBC because it is easier to locate the shooters. TPS games using ALC may want to adjust the low-ping value due to this property.

References

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

[2] W. K. Lee and R. K. C. Chang. Evaluation of lag-related configurations in first-person shooter games. In *2015 International Workshop on Network and Systems Support for Games (NetGames)*, pages 1–3, 2015.

[3] W. K. Lee and R. K. C. Chang. On "shot around a corner" in first-person shooter games. In *2017 15th Annual Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, June 2017.

[4] W. K. Lee and R. K. C. Chang. Enhancing the experience of multiplayer shooter games via advanced lag compensation. In *ACM Multimedia Systems Conference*. ACM, 2018.

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

[6] Chris Pereira. 2016's best-selling games in the US revealed. `https://www.gamespot.com/articles/2016s-best-selling-games-in-the-us-revealed/1100-6447090/`, 2017.

[7] Arnaud Kaiser, Dario Maggiorini, Nadjib Achir, and Khaled Boussetta. On the objective evaluation of real-time networked games. In *Global*

Telecommunications Conference, 2009. GLOBECOM 2009. IEEE, pages 1–5. IEEE, 2009.

[8] Lothar Pantel and Lars C Wolf. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 23–29. ACM, 2002.

[9] Jeremy Brun, Farzad Safaei, and Paul Boustead. Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51, 2006.

[10] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 152–156. ACM, 2004.

[11] 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 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151. ACM, 2004.

[12] AAA (video game industry). `https://en.wikipedia.org/wiki/AAA_(video_game_industry)`.

[13] CoD Infinite Warfare & Modern Warfare netcode analysis. `https://www.youtube.com/watch?v=oKE_eaTb1TU`.

[14] Counter Strike Global Offensive netcode analysis. `https://www.youtube.com/watch?v=pHi2DfSFFpk`.

[15] Titanfall 2 netcode analysis. https://www.youtube.com/watch?v=-DfqxpNrXFw.

[16] Wesley Yin-Poole. EA addresses "unacceptable" Battlefield 4 launch. http://www.eurogamer.net/articles/2014-06-19-ea-addresses-unacceptable-battlefield-4-launch, 2014.

[17] Eddie Makuch. Battlefield 4's rocky launch "absolutely" damaged player trust, producer says. http://www.gamespot.com/articles/battlefield-4s-rocky-launch-absolutely-damaged-pla/1100-6422805/, 2014.

[18] Dave Tach. EA halts Battlefield 4 expansions to "sort out all the issues". http://www.polygon.com/2013/12/4/5175588/ea-halts-battlefield-4-expansions, 2013.

[19] Community test environment.

[20] R52 server update sep 14. http://battlelog.battlefield.com/bf4/forum/threadview/2955065245255263541/, .

[21] Battlefield 4 fall update patch notes. https://eaassets-a.akamaihd.net/dice-commerce/battlefield4/assets/patch_notes/Patchnotes_FallPatch_EN.pdf, .

[22] Sean A Pfeifer. Real-time multiplayer gaming: Keeping everyone on the same page.

[23] Hitscan. https://en.wikipedia.org/wiki/Hitscan.

[24] PC game update Dec 16. http://battlelog.battlefield.com/bf4/forum/threadview/2955064770534426141/.

[25] Initiative #2: FramehistoryTime testing. http://cte.battlelog.com/bf4/news/view/initiative-2-framehistorytime-testing/.

[26] https://twitter.com/leewaikiu/status/727257446918447104.

[27] Battlefield 1 netcode analysis. https://www.youtube.com/watch?v=Sa_AsRYlBOA, .

[28] Battlefield 1 update notes âĂŞ spring update. https://www.battlefield.com/news/update-notes/spring-update, .

[29] Battlefield 1 update notes âĂŞ may update. https://www.battlefield.com/news/update-notes/may-update, .

[30] PlayOverwatch. Developer update | let's talk netcode | Overwatch. https://www.youtube.com/watch?v=vTH2ZPgYujQ.

[31] Hero movement speeds (plus a few more). https://www.reddit.com/r/Overwatch/comments/4ajmq5/hero_movement_speeds_plus_a_few_more/.

[32] Grenville Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 137–141. IEEE, 2003.

[33] Tristan Henderson. Latency and user behaviour on a multiplayer game server. *Networked Group Communication*, pages 1–13, 2001.

[34] G Armitage and Sebastian Zander. Empirically measuring the QoS sensitivity of interactive online game players. 2004.

[35] Benjamin Kenwright. Technical requirement certification (TRC) game development. http://games.soc.napier.ac.uk, 2014.

[36] Technical certification requirements - March 2009 (v1.5i). `http://blog.csdn.net/baozi3026/article/details/4272761`, 2009.

[37] `https://www.google.com/#q=oor+site:battlelog.battlefield.com`.

[38] Wu-chang Feng and Wu-chi Feng. On the geographic distribution of on-line game servers and players. In *Proceedings of the 2nd workshop on Network and system support for games*, pages 173–179. ACM, 2003.

[39] Sebastian Zander, Ian Leeder, and Grenville Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 117–124. ACM, 2005.

[40] Anh Le and Yanni Ellen Liu. Fairness in multi-player online games on deadline-based networks. *HonorâĂŹs Project Report, University of Manitoba*, 2006.

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

[42] Rodrigo Vicencio-Moreira, Regan L Mandryk, and Carl Gutwin. Balancing multiplayer first-person shooter games using aiming assistance. In *Games Media Entertainment (GEM), 2014 IEEE*, pages 1–8. IEEE, 2014.

[43] Mark Claypool, Kajal Claypool, and Feissal Damaa. The effects of frame rate and resolution on users playing first person shooter games. In *Proceedings of SPIE*, volume 6071, pages 1–11, 2006.

[44] Source multiplayer networking. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

[45] clumsy, an utility for simulating broken network for Windows Vista / Windows 7 and above. https://jagt.github.io/clumsy/.

[46] YXxTRUTHxXY. Ever wonder how fast your soldier runs/walks ? http://battlelog.battlefield.com/bf4/forum/threadview/2955064779058213894/.

[47] BlackRetina. CSGO weapon spreadsheet. https://docs.google.com/spreadsheets/d/11tDzUNBq9zIX6_9Rel__fdAUezAQzSnh5AVYzCP060c/edit#gid=0.

[48] Dimensions - Valve developer community. https://developer.valvesoftware.com/wiki/Dimensions#Speed.

[49] Rigid body. https://en.wikipedia.org/wiki/Rigid_body.

[50] Ian Shadden and Alexander Paschall. Blueprint line & shape traces — live training — unreal engine. https://www.youtube.com/watch?v=2LP5shWCnhc.

[51] Usability First. counterbalancing. http://www.usabilityfirst.com/glossary/counterbalancing/, 2018.

[52] Lag compensation. https://wiki.teamfortress.com/wiki/Lag_compensation#Exceptions.