# Internet For All - Fairness in multiplayer network games

Felix Kosian
felix.kosian@tum.de
Technical University Munich
Munich, Germany

## Abstract

Fairness is one of the key elements for competitive first person multiplayer games. Not only the absolute fairness but especially the perceived fairness is important for a good player experience and therefore a necessity for a successful game. Achieving an accepted level of fairness is not always given, so in this paper we will first go over the key elements of fairness, the basics of networking and why only a client-server model will be able to prevent cheating. Then we will create the basic server-client interaction guidelines used in many currently popular games, look at common problems like "Shot around a Corner" and finally we will look at various other factors that will help to improve the perceived fairness regarding latency problems.

## 1 Introduction

Video games are one of the most used forms of entertainment, especially for younger generations[18]. The most played form of them are multiplayer games with first-person-shooter (FPS) leading in popularity[6]. To provide the most fun experience for all, fairness is a key cornerstone in a social online environment.
In this paper we will focus on the perceived fairness of players which is more important for a better experience than the absolute fairness. Over the last two decades there have been many concepts which address methods to improve problems in network games. This paper will collect and summarize the most important factors and modern solutions regarding latency.
We will not discuss network congestion, packet loss compensation, update rates and complete game state update vs partial game state update or other elements like matchmaking and cross-play balancing which are extensive research areas on their own.

## 2 Fairness

While the absolute fairness can be defined as "the quality of treating people equally or in a way that is right or reasonable"[1], the perceived fairness can strongly deviate. Especially the fairness during a process that led to a certain outcome are important in competitive video games.
Following characteristics where identified to have an important impact on fairness[14]:

1. Accuracy. Procedures are based on accurate and valid information;
2. Bias suppression. Procedures are not affected by personal bias, preconception or self-interest;
3. Consistency. Procedures are consistently applied across people and time;
4. Correctability. Procedures provide opportunities to modify or reverse decisions such as allowing appeals and grievances to be considered;
5. Ethicality. Procedures are congruent with the moral and ethical values held by the people affected;
6. Representativeness. Procedures are representative in reflecting the basic concerns of the people affected.

### 2.1 Perceived unfairness in FPS

For the perceived fairness in FPS games we will focus on examples which show unfairness in Accuracy, Consistency, Correctability and Representativeness.

**2.1.1 Control loss.** If an input of a player has no effect, the player has no control over the game anymore. This can be the case if the input is not recognized or there is no feedback for this action.
An example is if the button click for shooting has no effect.

**2.1.2 Control delay.** If an input of a player has the expected connected result but the execution is delayed and the expected overall result is different.
One examples is a player who wants to jump over a gap while moving but falls in because the jump is executed after the player walked over the edge.
Another example is the case where a player is shooting another player but the shot is not hitting because the shot was executed later and the other player is not at the same spot anymore or the position of the other player is displayed delayed where in fact the other player moved away already.

**2.1.3 Implausible outcome.** This includes the cases above but has some more specific situations that are by now quite infamous in FPS games.
The first example is the so called Rubberbanding. This occurs if a player is not moving normally in one direction but the character is teleporting back and forth along its path which makes it impossible for other players to interact with it.
The second example is called "Shot around a Corner"[20, 23]. This is the case when Player 1 is shooting at Player 2 who

is running towards the cover. For Player 1, the shot is hitting because Player 2 is still in the open. Player 2 is in his view already behind cover but still gets hit even though this should not be possible in their view.

**2.1.4 Cheating.** While cheating is not the main topic of this paper, it is still important to consider for choosing a network architecture. Cheating in the context of network games includes all variants of a player manipulating the data which is sent over the network to gain an unfair advantage over other players. This makes it impossible to have a fair game and for that reason we have to consider the received data from other players as untrustworthy.
We can therefore only consider data that lies inside the allowed inputs. E.g. the absolute position of another player could be manipulated so we should only allow their button presses for movement to be considered.

### 2.2 False case of unfairness

Point 2, Bias suppression, in particular can contradict the perception of fairness and can be a cause of false cases of unfairness. This case is often when players complain that that the game is unfair whereas only their ability of competing with other players is lacking[2].

## 3 Background Networking

### 3.1 Network Connection

For a network game to work, information has to be exchanged between multiple machines. This results in physical limitations which greatly impact player experience and fairness and are often the root of the unfair examples given before.

**3.1.1 Network Latency.** Network latency or in the context of gaming often called lag, ping, round-trip-time (RTT) or network delay[16] is for this paper defined as the time it takes for an input on one machine to reach another machine and send information back to the original or another machine. In case of an FPS this can be the time for a player input and the resulting movement of the ingame character. Another case can be the firing of a weapon and the hit of an opponent or the Shot around a Corner example.
As a result, players with low latency will send and receive changes faster and can react faster than players with high latency and have an unfair advantage. Therefore lower latency results in a better player experience[22].
Latency increases with physical distance, connection medium, intermediate steps and processing speeds.

**3.1.2 Packet loss.** Packet loss is the occurrence of data being lost on the way to its destination. As a result information of commands like movement or firing of a weapon don't reach it's destination and have to be compensated, ignored or sent again. This can also be the cause of the Rubberbanding example.
Packet loss increases with bad physical connection, message routing issues such as congestion and processing errors.
Dealing with packet loss is important for a good player experience as well but we will not focus on this topic in this paper.

### 3.2 Network architecture

Typical networks for FPS games are Peer-To-Peer and Client-Server architectures. In a Peer-To-Peer system, every machine has a connection to every other machine. Often all machines are then called clients. In a Client-Server system, every machine of one player, called client, has a connection to one other machine, called dedicated server.
The special case where one machine is a server and a client, called host, will be handled like a dedicated server.

**3.2.1 Validation.** Validation describes which instance is allowed to call an action valid. In other words which machine is allowed to change the game state for all participants. E.g. a shot is registered and is allowed to deal damage.

**3.2.2 Client side.** Client side validation brings the benefit of having no delay for the client. When the player aims and shoots directly at a target, the client verifies the hit and sends the information to the server or other players. This is extremely important to have a responsive game to prevent frustration for the players.
The downside is that now this client can send any information, including incorrect and manipulated information the other participants.
In a Peer-To-Peer system, every client can only validate themselves because there is no instance which solely controls the state of the game.

**3.2.3 Server side.** With a Server side validation, all information has first to be sent to the server. There the information about e.g. positions and shots come together and can be evaluated without manipulation. The results then get sent back to all clients.
While this ensures correct information, the information will be delayed. Players have to shoot in front of other players so when the information reaches the server, the shot and the other player are at the same position[3]. This leading of the shot can differ with the different latency of the players and as a result leads to frustration and the feeling of an unfair and not correctly working game.

**3.2.4 Trusted information.** For a network game with unknown participants, the question is which information can be trusted. Because it is not possible to control all personal machines, the only trusted machine is the dedicated server which can be controlled by an entity trusted by all players.
The dedicated server architecture is the only way to prevent a majority of cheats[4]. Therefore we will only consider this architecture in following examples.

## 4 Basic Client-Server interaction

In this section we will go through a few examples of two players interacting with each other. Step by step we will solve upcoming problems to reach a basic functioning client-server guideline concept[3–5, 7] which is used by many currently popular games like Valorant[9], CS:GO[17], Fortnite[12], Overwatch[15], Apex Legends[13] or Battlefield V[11].

### 4.1 Player Movement

In this example Player 1 is starting to move in one direction and then stops after a few meters again. Player 2 is only watching this action.

**4.1.1 Movement: client side.** Player 1 is locally moving and sending the new state of it's position to the server where this information is forwarded towards Player 2 where the position of Player 1 is displayed delayed.

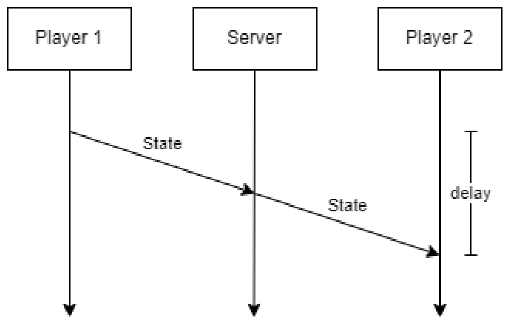

**Figure 1.** Player movement client-side

**4.1.2 Movement: server side.** Now to prevent cheating of Player 1 we only send the action to the server to validate. It seems like the state after the movement action is simultaneously received at both players.
Not only is that not true if the RTT for both players is different, but Player 1 has a delay of it's actions and the resulting state as well.

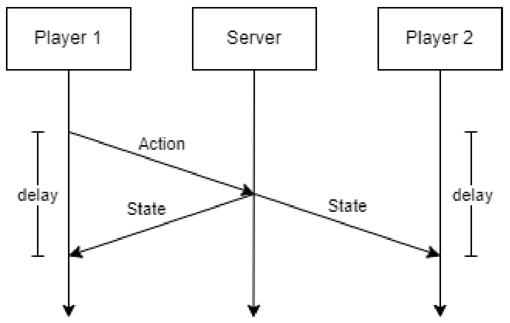

**Figure 2.** Player movement server-side

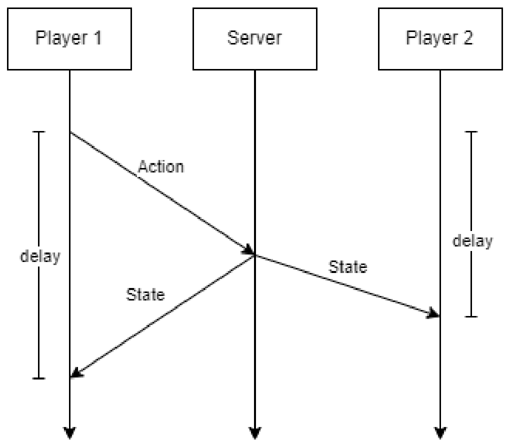

**Figure 3.** Player movement server-side, different RTT

**4.1.3 Movement: server side with local client side prediction.** To prevent this delay, the client can predict the same action locally parallel to sending the action to the server.
This seems fine but what happens if multiple actions are sent before the state of the ones before come back? We will have a mismatch between the prediction and the validated state. Therefore, we have to move the Player 1 back but then when the second state come back we have to move Player 1 forwards again. This can be one reason for Rubberbanding.

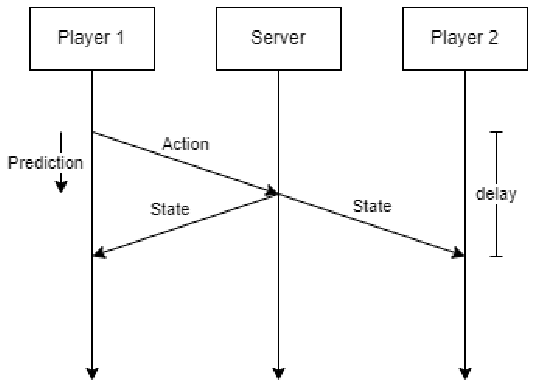

**Figure 4.** Player movement server-side, local prediction

**4.1.4 Movement: local client side prediction + server reconciliation.** Instead of resetting the Player 1 to the state of the server we can add an identifier to the actions and keep a record of the actions. Then if Player 1 gets a response we can predict with those action again.

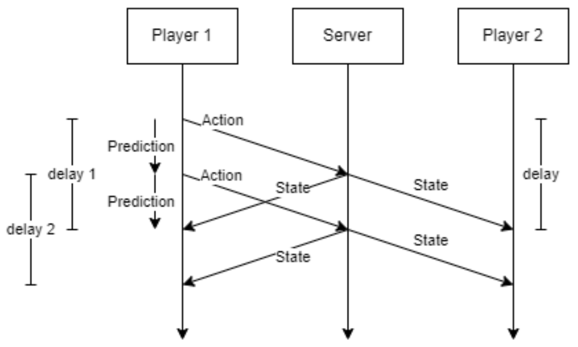

**Figure 5.** Player movement server-side, local prediction mismatch

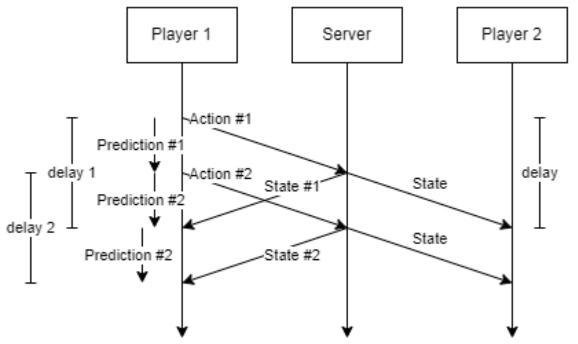

**Figure 6.** Player movement server-side, local prediction with reconciliation

#### 4.1.5 Movement: external prediction.

Until now Player 2 only observed the state of Player 1. In a real-time game the gap between those observations has to be continuously simulated. This can be done either by 1) dead reckoning, also called extrapolation, which is the prediction from a position and a direction with the assumption that the movement will be linear.

Or method 2) waiting for the next state and then interpolating the movement in between two states.

While method 1) is accurate most of the time and does not introduce more delay, it can lead to wrong predictions and Rubberbanding as well.

Method 2) is always accurate but introduces a larger delay.

### 4.2 Player 1 shooting and Player 2 moving

Let's look at a more complex scenario. Player 1 sees Player 2 moving perpendicular in front of it, aims at the player and shoots.

#### 4.2.1 Shot not hitting.

Now this information is sent to the server. The server checks if the shot hit anything but because of the delay, Player 2 already moved forwards, so

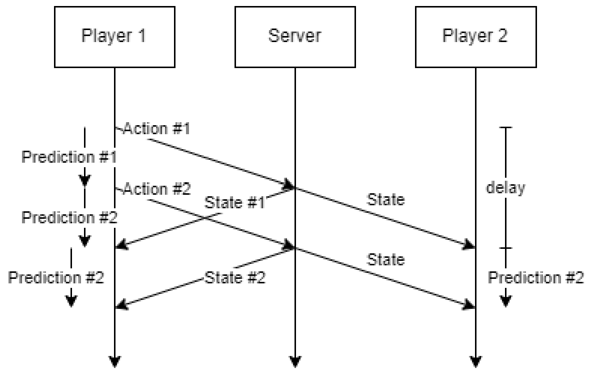

**Figure 7.** Player movement server-side, external reckoning

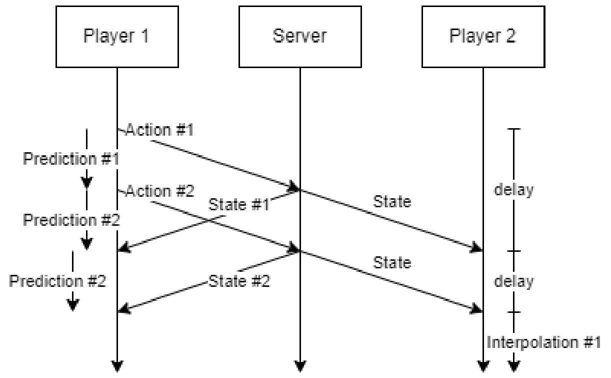

**Figure 8.** Player movement server-side, external interpolation

no hit is registered. In other words, Player 1 only sees where Player 2 has been before and would have to lead their shot. This is frustrating because it is impossible to know how far to lead the shot because this is dependent on the delays.

#### 4.2.2 Server side reconstruction.

We can solve this problem by reconstructing the state of the game on the server for the view of Player 1 during the shot. This is possible if we have timestamped all actions which were sent and received to and from the server. The server can check to which outcome those actions lead for Player 1. Of course this increases the advantage of Player 1 because independent of where Player 2 is at this point of time, the shot will be counted as valid.

#### 4.2.3 Shot around a Corner.

This leads to the situation where Player 2 moved around a corner into cover to prevent being shot but because of the reconstruction, gets hit anyways.

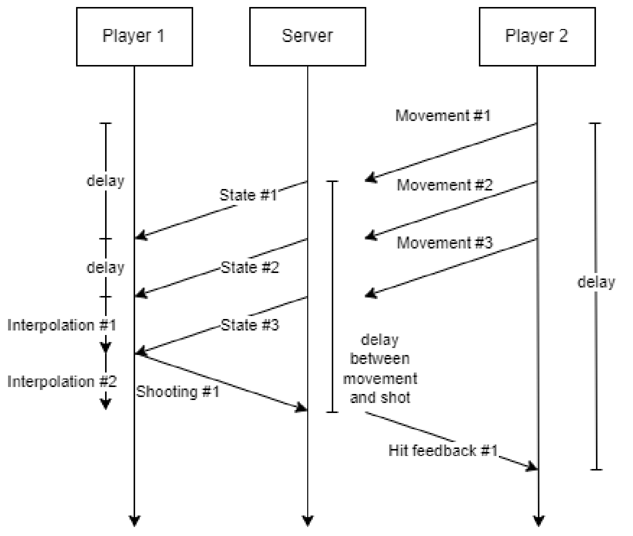

**Figure 9.** Player movement and shooting

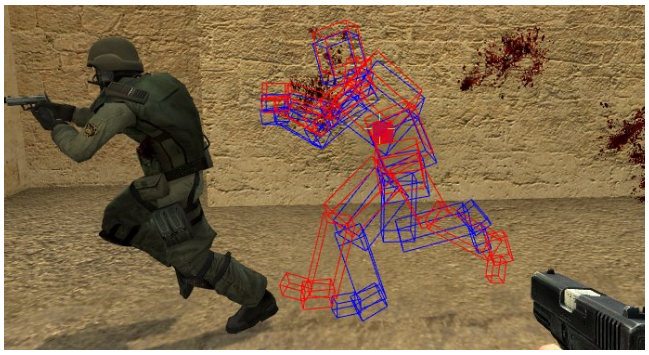

**Figure 10.** View of Player 1 on the server

### 4.3 Remaining Problems

This server-client interaction is the basis of most modern competitive FPS games. To recap, we now have a server which validates all actions and controls the game state. Furthermore, it reconstructs the game state in favor of the shooter. The clients predict locally the own interactions and interpolate the actions of other players.
As a result we have a much better player experience but still a few more issues to solve:

- high RTT: players with high ping have an unfair advantage because of the reconstruction lag compensation
- Shot around a Corner problem
- the game feels sometimes unresponsive because of having only simple local prediction

## 5 Solutions

There are numerous ways to reduce the network problems or mitigate their effects. In the following section we will go through different approaches. These can also be combined for the best results.

### 5.1 Reducing Latency

The most effective approach is to reduce the problem at the origin, the network delay between client and server.

**5.1.1 Local Competition.** The simplest method to reduce latency is to reduce the distance which the data has to travel. This is one reason why official competitions where teams compete against each other are held in a specific location with clients directly connected to a local server.

**5.1.2 Accelerator Network.** Nowadays some cloud service providers like Amazon Web Services (AWS) provide a parallel network to the standard internet. Games can forward their data to the nearest router which then leads the data around the world with fewer hops and less traffic[8].

**5.1.3 Excluding players.** Often games have a maximum RTT. If players have a higher RTT to the server than allowed, they will be disconnected from the game[20]. While this ensures a low ping for all players in the match, the players who can't participate have no solution. Of course in general if the RTT is too high, it will be impossible to have a real time game playable. In those cases other game concepts are needed that do not require fast interactions between players.

**5.1.4 Grouping players with similar RTT.** To increase the absolute fairness, one match could be created only with players with similar RTT. In this case no player would have an advantage over other players. The downsides are that this is maybe not possible due to a small amount of players, multiple people with different pings wanting to play in the same match or RTT changes during the match.

**5.1.5 Multi-server.** The idea behind a multi-server approach is to connect all clients to the closest server. Then only the servers have to exchange information between each other. As a result the RTT of an interaction between two players can be much faster and in the worst case still as fast as a single server approach.

### 5.2 Compensation methods

If the RTT can't be reduce anymore, there are other methods to compensate this delay.

**5.2.1 Lag compensation.** We already introduced the basic version of lag compensation, the server side reconstruction. Other modern approaches utilize deep learning methods to reduce effects of latency[19] or advanced lag compensation which uses additional checks on the client side as well[21].

**5.2.2 Client side initiation, Server-validation.** To reduce the load on the server, it is also possible to calculate all states on the client and only have the server validate critical moments. This is used in Battlefield 1[10] when the shooter has a ping under 150ms. Above this threshold, the shooter has to lead their shot.

#### 5.2.3 Specific Shot around a Corner prevention.

Because the Shot around a Corner problem is one of the only critical points of perceived unfairness, some approaches target this specifically with additional checks[21].

### 5.3 Design and art improvements

To increase the perceived fairness, it often helps to design the game around the given limitations.

#### 5.3.1 No compensation.

Depending on the situation in the game, the latency might not even be realized. For example if two players walk towards each other and one player shoots, it does not matter if the shots are delayed. Another example is when a player is shot but does not look in the direction where the shot came from or did not get behind cover. The player would not perceive this as unfair as well[5].

#### 5.3.2 Delayed actions.

One approach to cover up the latency is to delay the actions. Often this is done by having an animation that is played at the start of an action. During that time the action can already be sent to the server but the player does not realize the delay from the server feedback.

#### 5.3.3 Local feedback.

To increase the responsiveness, feedback of an action can already be provided at the client side. This includes for firing a gun the muzzle flash or the hit indication. Of course this can lead to misleading situations when the server does not validate this action but is less recognizable than a delayed feedback[9].

#### 5.3.4 Movement speed.

The experienced effect of latency is often directly connected to the movement speed of in-game characters. So to reduce the effect, the movement speed can be reduced as well. E.g. the distance where a player around a corner is shot[20].

## 6 Conclusion

We established how to achieve a network concept which is used as a basis in many current competitive multiplayer games. By using a server-client architecture with validation on the server which uses basic lag compensation, the player experience is increased immensely in terms of fairness. With this basis we can use other techniques which decrease the effect of latency and increase the perceived fairness even more. As a result we have a good structures which hides the fact, that a real time game is processed delayed in the background.

## Acknowledgments

Special thanks go to my advisor Ljubica Kärkkäinen for giving me the freedom of choosing this topic and her motivating feedback.

## References

[1] fairness - cambridge dictionary, . URL https://dictionary.cambridge.org/dictionary/english/fairness.

[2] GDC Session Summary: Halo networking - Wolfire Games Blog, . URL http://blog.wolfire.com/2011/03/GDC-Session-Summary-Halo-networking.

[3] Lag Compensation - Valve Developer Community, . URL https://developer.valvesoftware.com/wiki/Lag_Compensation.

[4] Lag Compensation - Gabriel Gambetta, . URL https://www.gabrielgambetta.com/lag-compensation.html.

[5] Source Multiplayer Networking - Valve Developer Community, . URL https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.

[6] Unity report states 77% of gamers play multiplayer games, . URL https://mobidictum.biz/unity-report-states-77-percent-gamers-play-multiplayer/.

[7] What Every Programmer Needs To Know About Game Networking, February 2010. URL https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/.

[8] Improving the Player Experience by Leveraging AWS Global Accelerator and Amazon GameLift FleetIQ | AWS for Games Blog, March 2021. URL https://aws.amazon.com/blogs/gametech/improving-the-player-experience-by-leveraging-aws-global-accelerator-and-amazon-gamelift-fleetiq/. Section: Amazon Cognito.

[9] The State of Hit Registration, December 2022. URL https://playvalorant.com/en-us/news/dev/the-state-of-hit-registration/.

[10] Battle(non)sense. Netcode 101 - What You Need To Know, August 2017. URL https://www.youtube.com/watch?v=hiHP0N-jMx8.

[11] Battle(non)sense. Battlefield V Netcode Issues Tested & Explained, December 2018. URL https://www.youtube.com/watch?v=8Kvj5TZNNJ4.

[12] Battle(non)sense. Fortnite 4.2's Netcode Beats CS:GO And BF1!?, May 2018. URL https://www.youtube.com/watch?v=W5lUCeAu_2k.

[13] Battle(non)sense. Apex Legends Netcode Changes, June 2019. URL https://www.youtube.com/watch?v=xRj3KZJCDiM.

[14] David Chan. Perceptions of fairness.

[15] Daposto. Game Networking (9) — Bonus, Overwatch Model, July 2020. URL https://daposto.medium.com/game-networking-9-bonus-overwatch-model-4faba078cf05.

[16] Daposto. Game Networking (3) - RTT, PING, latency, lag, July 2020. URL https://daposto.medium.com/game-networking-3-rtt-ping-latency-lag-679b73b274ae.

[17] DevinDTV. How It Works: Lag compensation and Interp in CS:GO, November 2015. URL https://www.youtube.com/watch?v=6EwaW2iz4iA.

[18] Andrej Hadji-Vasilev. 23 Online Gaming Statistics, Facts & Trends for 2023, March 2022. URL https://www.cloudwards.net/online-gaming-statistics/.

[19] David Halbhuber. To Lag or Not to Lag: Understanding and Compensating Latency in Video Games. In *Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play*, pages 370–373, Bremen Germany, November 2022. ACM. ISBN 978-1-4503-9211-2. doi: 10.1145/3505270.3558364. URL https://dl.acm.org/doi/10.1145/3505270.3558364.

[20] Steven W. K. Lee and Rocky 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, Taipei, Taiwan, June 2017. IEEE. ISBN 978-1-5090-5038-3. doi: 10.1109/NetGames.2017.7991545. URL http://ieeexplore.ieee.org/document/7991545/.

[21] Steven W. K. Lee and Rocky K. C. Chang. Enhancing the experience of multiplayer shooter games via advanced lag compensation. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 284–293, Amsterdam Netherlands, June 2018. ACM. ISBN 978-1-4503-5192-8. doi: 10.1145/3204949.3204971. URL https://dl.acm.org/doi/10.1145/3204949.3204971.

[22] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, James Scovell, and Jamie Sherman. The Effects of Network Latency on Competitive First-Person Shooter Game Players. In *2021 13th International Conference on Quality of Multimedia Experience (QoMEX)*, pages 151–156, Montreal, QC, Canada, June 2021. IEEE. ISBN 978-1-66543-589-5. doi: 10.1109/QoMEX51781.2021.9465419. URL https://ieeexplore.ieee.org/document/9465419/.

[23] Jose Saldana and Mirko Suznjevic. QoE and Latency Issues in Networked Games. In Ryohei Nakatsu and Matthias Rauterberg, editors, *Handbook of Digital Games and Entertainment Technologies*, pages 1–36. Springer Singapore, Singapore, 2015. ISBN 978-981-4560-52-8. doi: 10.1007/978-981-4560-52-8_23-1. URL http://link.springer.com/10.1007/978-981-4560-52-8_23-1.