A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games

SHENGMEI LIU, XIAOKUN XU, and MARK CLAYPOOL, Worcester Polytechnic Institute

Computer games, one of the most popular forms of entertainment in the world, are increasingly online multiplayer, connecting geographically dispersed players in the same virtual world over a network. Network latency between players and the server can decrease responsiveness and increase inconsistency across players, degrading player performance and quality of experience. Latency compensation techniques are software-based solutions that seek to ameliorate the negative effects of network latency by manipulating player input and/or game states in response to network delays. We search, find, and survey more than 80 papers on latency compensation, organizing their latency compensation techniques into a novel taxonomy. Our hierarchical taxonomy has 11 base technique types organized into four main groups. Illustrative examples of each technique are provided, as well as demonstrated use of the techniques in commercial games.

CCS Concepts: • **Applied computing → Computer games;**

Additional Key Words and Phrases: Video games, lag, quality of experience, user study, delay

ACM Reference format:
Shengmei Liu, Xiaokun Xu, and Mark Claypool. 2022. A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games. *ACM Comput. Surv.* 54, 11s, Article 243 (September 2022), 34 pages.
https://doi.org/10.1145/3519023

1 INTRODUCTION

The computer games market was worth more than $150 billion globally in 2019 and is expected to grow about 13% per year from 2020 to 2027 [57]. There are about 2.7 billion computer gamers worldwide [67] who are increasingly playing games online. Online games are estimated to reach $196 billion in revenue by 2022 and are one of the fastest-growing industries in the world [72].

Although many factors can affect a player's gameplay quality of experience, multiplayer network games—where physically separated players simultaneously interact in a shared, virtual game world—must overcome the challenges posed by computer networks: limited bitrates, lost game data from dropped packets, and latency from sending game information from one computer to another. Although increased capacities can overcome bitrate limitations and reduce packet loss, latency remains a challenge for network games and can never be eliminated entirely. Latency determines not only how players experience gameplay but also how network games must be designed and implemented to mitigate the effects of latency and meet player expectations. Unchecked, latencies

Authors' address: S. Liu, X. Xu, and M. Claypool, Worcester Polytechnic Institute, Worcester, MA 01609; emails: sliu7@wpi.edu, xxu11@wpi.edu, claypool@cs.wpi.edu.

0360-0300/2022/09-ART243 $15.00
https://doi.org/10.1145/3519023

can degrade both player performance (e.g., lower game score) and quality of experience (e.g., less fun).

For multiplayer network games, the canonical game state is typically controlled by an authoritative server, and each player is on a separate client computer with a copy of the game state. The clients capture and send player actions to the server, which updates the game world appropriately and sends the new game state back to each client. The time it takes from when the player provides input until the new game state is rendered is *latency* in this article (colloquially called *lag* by gamers). Thus, a primary impact of latency is degraded *responsiveness* for the player in that there is an observable delay between player input and resulting game state update. In addition, player input and game update messages take time to propagate and process before all client and server states are the same. Thus, another impact of latency is reduced *consistency* between the game states at the clients and the game state at the server.

Latency compensation techniques are software algorithms that run on the game client or game server (or both) that try to mitigate the impact of latency on the players. Such techniques might improve responsiveness or increase consistency (or both) but often sacrifice one for the other (i.e., increased consistency at the cost of reduced responsiveness and vice versa). The effectiveness of the techniques depends upon many factors, including but not limited to the game to which they are applied, the network conditions between clients and servers, and the familiarity and skill of the players with the game.

Understanding what latency compensation techniques are effective and for what games and which network and game conditions can be helpful for (1) game developers and game system developers who want to improve the multiplayer network game experience, and for (2) researchers who seek to improve upon existing techniques. Such understanding can be gained through a *survey* of latency compensation techniques that collates information to provide an overview of existing techniques—or, even better, through a *taxonomy* that categorizes latency compensation techniques to illustrate the relationships different techniques have in common and organize the information so as to make accumulated survey information accessible based on a user's need.

An early, often cited de facto survey of latency compensation techniques by Bernier [10] describes a few techniques in detail, but the survey is more than 20 years old and does not encompass the full set of techniques developed today nor describe their use in modern games. A survey of latency compensation techniques by Jiang et al. [71] provides for only two types of latency compensation techniques (synchronization and optimistic) and is more than 15 years old, so it may not represent techniques developed since. A more recent survey by Briscoe et al. [13] surveys techniques to reduce network latency but includes only techniques at the systems level and without specific attention to computer games. We are not aware of any other surveys or taxonomies for latency compensation techniques for multiplayer network games.

This work provides a survey in the form of a literature review of peer-reviewed publications on latency compensation techniques for computer games and similar interactive media. We began with a core set of known (to us) papers on latency compensation techniques, then expanded our pool via following forward and backward (in time) citations to these papers and searching Google Scholar. This yielded more than 80 peer-reviewed publications that dealt with latency compensation for games. We used these 80+ papers as the foundation for a novel taxonomy of latency compensation techniques, divided into four main groups (feedback, prediction, time manipulation, and world adjustment), each further broken down into 11 technique types: concealment, exposure, self-prediction, interpolation, extrapolation, speculative execution, incoming delay, outgoing delay, time warp, control assistance, and attribute scaling.

Our survey further identifies the games, game genres, and methods of evaluation for each paper, showing that more than half of the papers do not include evaluations with users and about

20% have no evaluation at all. As shown by others [108, 122, 136, 152], latency compensation algorithms must be tested and then evaluated to determine their effect. Thus, our work also provides an indication as to where additional work might be done to assess the efficacy of techniques over a broader range of games and game conditions.

It is believed that many latency compensation techniques are implemented into modern multiplayer network games. Unfortunately, as is typical of much commercial software, computer games are usually closed-source and implementations are not specified. However, we have located 20 presentations and blogs from commercial game developers that indicate their latency compensation approaches for their games. We have categorized these using our taxonomy and provided links as specific evidence of latency compensation techniques used in practice.

The main contributions of this work include the following:

- Examples illustrating the main effects of latency on multiplayer network games: *responsiveness* and *consistency*.
- A collection of 85 peer-reviewed research papers that deal with latency compensation, providing information on type of technique, evaluation method, game and game genre evaluated, and latency range studied.
- A visual depiction of a taxonomy for latency compensation techniques, showing a hierarchical representation of techniques grouped by similarity.
- Definitions for each technique ("node") in the taxonomy.
- A visual example for each "leaf" in the taxonomy, illustrating the application of each technique to a computer game.
- Identification of 20 presentations and blogs by commercial game developers detailing the latency compensation techniques applied to their games.

Section 2 describes our methodology to survey peer-reviewed research papers on latency compensation techniques. Section 3 provides background information on latency, interactivity, and games. Section 4 presents our taxonomy, and Section 5 lists presentations and blogs that provide evidence of latency compensation in commercial games. Section 6 discusses our insights provided by the survey and taxonomy, and Section 7 summarizes our work and presents possible future work.

2 METHODOLOGY TO OBTAIN PAPER POOL

Our objective is to provide a comprehensive review of peer-reviewed research papers on latency compensation techniques applied to multiplayer network games and similar interactive applications. This section describes the methodology to obtain a pool of all relevant papers.

A comprehensive literature review survey examines all (or nearly all) scholarly sources related to a specific research question or topic (in our case, latency compensation). Unfortunately, given the general nature of representative keywords, doing an exhaustive search via the Web is prohibitive. For example, as of November 2020, a search in Google Scholar with the keywords "Latency Compensation" had 144,000 results, "Latency Compensation and Interactive Media" had 24,800 results, and "Latency Compensation and Games" had 11,600 results.

Instead of a top-down search approach, we opted for a bottom-up approach. We started with about 20 initial research papers that dealt with latency compensation. Careful review of these papers provided the initial version of the taxonomy. From this list of papers, we expanded our reference list based on earlier works the original papers cited as well as forward (in time) references that cited these papers. Our main selection criteria were papers that dealt with latency compensation in some significant fashion. For example, a paper that proposed, discussed, or evaluated a latency compensation technique was included, whereas a paper that merely mentioned latency compensation in the related work section was not included.

This yielded a pool of about 50 papers.

From there, we did a search in Google Scholar with the keywords "Latency Compensation and Games," sorted them by relevancy, and went through the first 200 entries, adding those that involved latency compensation and did not yet have in our pool. We repeated these same steps for the keywords "Delay Compensation and Games."

For each paper in the pool, we categorized the work based on the initial taxonomy and recorded relevant details on the paper. We also noted equivalency terms used in the paper to map them to the terms used in our taxonomy. Papers were discussed by us to confirm details and placement in the taxonomy. Whenever appropriate (i.e., a paper's technique did not fit into our taxonomy), we would rework categories in the taxonomy until the technique could be placed. We also examined the list of works cited and added any missing previous references to our pool. We continued this process until the pool was exhausted (i.e., we had reviewed and classified all latency compensation papers known to us).

For inclusion in our list, we only considered peer-reviewed publications (i.e., not white papers, technical reports, theses, or patents) and made no determination of the "quality" of the peer-reviewed forum nor the quality of the paper itself. When in doubt (i.e., we were not sure if the paper was peer-reviewed or invited), we included the paper.

For topics, we only consider multiplayer interactive systems, so this excludes the related area of tele-operation, which can also suffer from network latency. The interested reader is encouraged to refer to a survey by Farajiparvar et al. [44] for more information on this topic.

We also exclude literature that only focuses on local system latency (i.e., not network latency), such as those that predict mouse movements [20], or hardware upgrades that might reduce local latency, such as monitors with low refresh rates [132], and computer mice with high polling rates. We also do not consider latency reduction in low-level controllers, such as latency compensation for power oscillation damping controllers [21].

Network system techniques for reducing latency are also excluded, including but not limited to choice of transport protocol, upgrades to link capacities, sizing of router buffers, and path routing. The interested reader is encouraged to refer to a survey by Brisco et al. [13] for more information on this topic.

Similarly, we also exclude techniques that reduce latency through system reconfiguration, such as moving a game server closer to a player's game client. Although server placement can have a marked impact on client-server latency, such techniques are not done in-game and must be done *a priori*. Moreover, moving servers often cannot be done for multiplayer games where players are geographically separated [55] or for games deployed without the resources needed for a distributed architecture.

Instead, the techniques surveyed assume there is a base level of latency present in the game system (i.e., local latency), and there is an unavoidable network latency between the clients and the server. The latency compensation techniques seek to mitigate the network latency by end-host computations on the clients or the server (or both), adjusting anything between player input and game actions to game state and rendered output.

3 BACKGROUND ON LATENCY, INTERACTIVITY AND GAMES

This section provides background information on game architectures (Section 3.1), latency in computer game systems (Section 3.2), and the effects of latency on computer games (Section 3.3).

3.1 Game Architectures

Underlying network game architectures can be peer-to-peer, client-server, or server pool [128].

A peer-to-peer architecture is when a player's computer connects directly to another player's computer, without an intermediate game computer processing game state. Peer-to-peer configurations have the advantage of not needing additional computers to support the game beyond those used by the players. However, two significant disadvantages are the following (1) preventing players from cheating can be difficult since there is no non-player computer controlling game state, and (2) players with less-powerful computers can often degrade the gameplay experience by all players in the peer-to-peer group.

A client-server architecture is where one computer (the server) is the relay for all communication between player computers (the clients). While having the disadvantage of needing an "extra" computer for the server, the client-server architecture overcomes the main disadvantages of the peer-to-peer architecture by making it more difficult for players to cheat if the server is controlled by the publisher and, if the server is a powerful computer, limiting the impact of under-powered client computers on the game.

A server pool architecture has several interconnected servers that communicate as peers while each player's client connects to a local server. Server pools can reduce the demands on any one server as the number of game clients increase, but they have additional complexity in sharing game state information and, most relevant to our work, can add additional latency for clients connected to one server to communicate with clients connected to a different server in the pool.

Most multiplayer network games use a client-server architecture. This architecture is popular for network games since firewall rules can make it difficult for clients to connect to each other. Moreover, having a single server can help a game scale with number of players while avoiding the extra latency from server pool architectures. Last but not least, architectures with a trusted authoritative server (e.g., managed by the game publisher) can help thwart player cheating since the server keeps the official game state and makes all final decisions about the outcome of player actions.

Typically, the server is at a well-known IP address and port and is public so as to be reachable by all clients playing together in one game. In some cases, a server browser setup allows game servers to register their individual IP addresses and ports with a central server, allowing clients to connect to the central server and browse available game servers, the individual game servers being differentiated based on configuration parameters (e.g., a certain game mode, map, or network latency). Once an individual game server is chosen, the central server provides the client with the game server IP address and port whereupon the client connects to the game server to play the game.

Some network game architectures have one client act as the host to which the other clients connect. Although at the network level these have peer-to-peer connections (i.e., one client communicates directly with another client), they can still be viewed as have a client-server architecture in that the host acts as a server, albeit one that also acts as a client for that player.

3.2 Latency in Computer Game Systems

In general, latency in a computer game affects how long it takes from when a player acts until the player sees/hears the results. And with computer games, there is *always* some latency between player actions and game output.

Latencies around 20 to 30 ms can be noticed for interactive music [103], and even lower at around 10 ms [68] and perhaps even 2 ms [104] for dragging-related tasks. Latencies around 50 ms are known to affect performance in mouse-based pointing tasks [68, 96]. Latencies under 100 ms have been demonstrated to affect game-related tasks as well, such as moving target selection and steering [52, 95, 109].

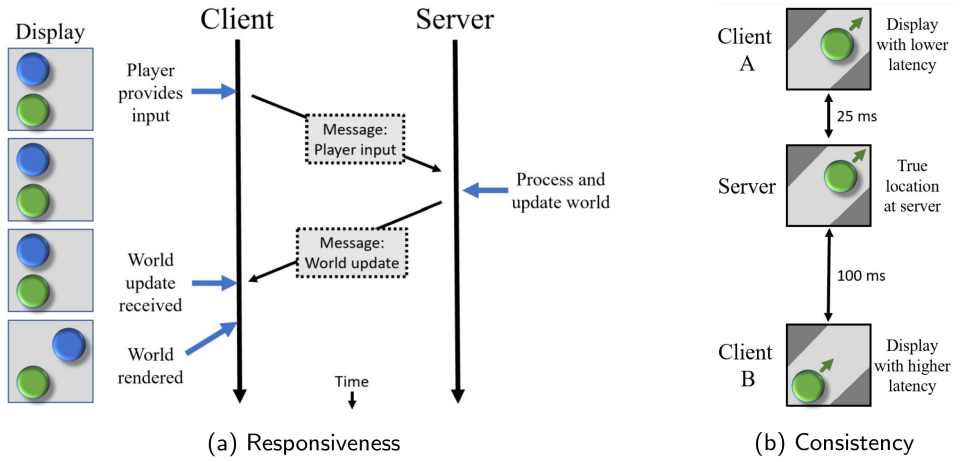

Fig. 1. The effects of latency on network games.

For a non-network game, the latency is from the player's local computer and includes delays from processing by the input devices, operating system, game and game engine, graphics cards, and display devices. Such latency is sometimes called *click to photon* by gamers, referring to the time from a mouse click until the monitor displays the result. Local latencies for games can range from high-performance game systems with latencies around 25 ms [94] to console and TV combinations that have latencies of more than 200 ms [66, 112].

For network games, the local system latency still impacts the player, but there is also *network latency*—the round-trip time between the game client and the game server, including all network processing on the end-hosts (the network interface cards and OS stack) and all intermediate devices/hops between. The primary focus of latency compensation techniques is to mitigate these network latencies. Network latencies can vary by several orders of magnitude, from milliseconds for a Local Area Network (LAN) to tens of milliseconds for an intra-continental network to hundreds of milliseconds for a inter-continental network. Wireless networks, such as WiFi, mobile, and satellite, with poor network conditions can even experience seconds of latency.

For a multiplayer network game, the player is vulnerable to *both* network latency and local latency. However, most latency compensation techniques are designed to overcome network latency and not local latency.

3.3 The Effects of Latency on Multiplayer Network Games

Latency can make a multiplayer network computer game *less responsive*. Consider the example in Figure 1(a) showing an event timeline for a network game with a client and a server, with time progressing top to bottom. The left side shows the display on the client over this time. In this example, the player provides input and the client encapsulates that input into a message that is sent to the server. The server receives the message, processes the input, updates the game world, and sends the world update back to the client. The client, upon getting the update, renders the new world for the player on the display. The time from the player input until the resulting change to the display is the response time. The larger the network latency, the greater the response time and the less responsive the computer game.

Latency can also make a network computer game *less consistent*. Consider the example in Figure 1(b) showing a multiplayer networked game with two clients with different latencies connected to a server. The authoritative server has the true location of an avatar depicted by the green

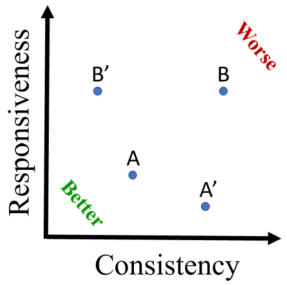

Fig. 2. Tradeoff between consistency and responsiveness.

circle. The avatar is moving up and to the right along the path. The server periodically sends position updates to each client. Ideally, in most game conditions, such as a race where avatars are running together around a virtual track, the view of the game worlds on both Client A and Client B would be the same (consistent), but because of latency they are not. Since Client A is 25 ms away from the server, the displayed position is close to that of the server. However, since Client B is more distant from the server at 100 ms, the displayed position is even farther away from the actual position. The client game worlds are inconsistent with each other and the server.

This impact of latency can be visualized as in Figure 2, where consistency and responsiveness are depicted by orthogonal axes. In this space, the best position for a multiplayer network game—the one with the highest responsiveness and most consistency—is at the origin at the bottom left. The farther away from the origin, the worse (i.e., the less responsive and/or more inconsistent). Client A from the example in Figure 1(b) might appear at point *A* somewhat near the origin, with Client B at point *B* being farther away. Latency compensation techniques seek to improve the responsiveness or consistency (or both) in the presence of network latency, moving a client closer to the origin. For example, one technique may improve the consistency of Client B to position *B'*, whereas another may improve the responsiveness of Client A to position *A'*, at the cost of a decrease in consistency.

Because there is always at least some latency with a network computer game, being at the origin in Figure 2 is impossible. In fact, there is often a tradeoff between consistency and responsiveness. Some latency compensation techniques designed to improve responsiveness in the presence of latency can decrease consistency, whereas other techniques that improve consistency can make a game less responsive.

There has been numerous studies on the effects of network latency on games, assessing the effects of latency on both player performance and **quality of experience (QoE)**.

For **first-person shooter (FPS)** games, Armitage [7] estimates that the player tolerance threshold for latency for Quake 3 is about 150 to 180 ms. Dick et al. [36] show via a survey that players generally believe that about 120 ms is the maximum tolerable latency for a network game, regardless of game genre, but their user study shows that players find 150 ms acceptable for the games Counter-strike and Unreal Tournament 2003 (UT2003). Quax et al. [111] find that for UT2003 players, latency and latency jitter as low as 100 ms can degrade player performance and QoE. Amin et al. [4] show that player experience determines the sensitivity to latency for Call of Duty, with competitive gamers more adept at compensating for impaired conditions. Liu et al. [93] for Counter Strike: Global Offensive find that decreasing latencies from 125 ms to 25 ms improves player accuracy by about 2%, score by about 15%, and QoE by about 20%.

For other genres, Pantel and Wolf [107] show that latencies of about 100 ms can affect car racing games. Fritsch et al. [53] find that players of the role-playing game Everquest 2 can tolerate

hundreds of milliseconds of network latency. Hohlfeld et al. [63] find that players of the casual game Minecraft are insensitive to network latencies of up to 1 second. Sheldon et al. [125] show that some aspects of play in the real-time strategy game Warcraft 3 are not affected by even a second of latency. Howard et al. [64] indicate that cooperation in the role-playing game Mass Effect 3 can be affected by latency of a teammate due to cascading effects on the game outcome.

Many of these studies use specific games but are intended to generalize to games from the same genres (e.g., FPS games) [4, 7, 9, 22, 25]. However, studies using commercial games include a game engine's built-in latency compensation techniques, often confounding the results.

Generally, the effects of latency depend upon the game type, the player actions in the game, and the characteristics of those actions [26, 116]. For example, games in which players take turns are more forgiving of latency than are games with tight interactivity. Claypool and Claypool [26] suggest that a game action's sensitivity to latency can be classified by precision and deadline—higher precision and tighter deadline mean more sensitivity to latency. For game combat, shooting with a high-precision weapon is affected by latency more than shooting with a weapon with a wide area of effect. For game navigation, driving a car around a race track is affected by latency more than sending an avatar across a large world to explore. Additional research suggests that the effects of latency also depend upon a player's skill, with higher-skill players more affected by latency than lower-skill players [93].

4 SURVEY AND TAXONOMY OF LATENCY COMPENSATION TECHNIQUES

Our methodology for obtaining peer-reviewed research papers on latency compensation techniques yielded 85 peer-reviewed publications that are surveyed and taxonomized in this section. Table 1 provides the list of papers:

- *Year* is when the paper was published.
- *Compensation Technique* maps the latency compensation technique(s) in the paper to our taxonomy. Note that many papers include more than one technique.
- *Study Type* is one of "User Study" involving human subjects, "Experiments" using games or simulated games but no active human players, or "Case Study" with a single instance described as a proof of concept. If the paper has no meaningful evaluation, the study type is listed as "None."
- *Game* provides the name(s) of the game(s) used in evaluation (if done). Game names are provided for commercial games or publicly available games—custom-built games are so indicated as "Custom." Studies evaluating a task rather than a game, such as drawing [6] or writing [98], are listed as "Task."
- *Genre* is the general category to which the evaluated games belong: "FPS" (first-person shooter), "RPG" (role-playing game), "RTS" (real-time strategy), "Casual" (also covering turn-based games), "Fighting," "Racing," "Sports," and "Arcade." The Arcade genre also serves as a catch-all covering games with limited game actions (and gameplay depth) and simple graphics. For cases where the evaluation is an experiment with generic interaction, the genre is listed as "Any."
- *Range* indicates the network latency values evaluated, where applicable and specified.
- *Users* has the number of users in the evaluation, where applicable.

Figure 3 depicts our taxonomy of latency compensation techniques for multiplayer network games. The techniques are arranged in groups based on shared characteristics. The proposed taxonomy is hierarchical, read from the top down, where each vertical level provides increased differentiation and refinement of the indicated technique (i.e., more general on the top to more specific on the bottom).

Table 1. Peer-Reviewed Publications with Research on Latency Compensation for Network Games

Year	Paper	Compensation Technique	Study Type	Game	Genre	Range	Users
1985	[70]	Time Warp	None	N/A	N/A	N/A	N/A
1992	[131]	Latency Concealment	User Study	Custom	N/A	0–380 ms	12
1994	[101]	Extrapolation	None	N/A	N/A	N/A	N/A
1998	[124]	Interpolation	Case Study	Custom	Arcade	0–2,000 ms	2
1999	[16]	Extrapolation, Latency Exposure	Experiment	N/A	N/A	N/A	N/A
1999	[141]	Attribute Scaling, Extrapolation, Latency Exposure	None	N/A	N/A	N/A	N/A
1999	[38]	Extrapolation, Incoming Delay	User Study	MiMaze	Arcade	0–100 ms	25
2000	[99]	Extrapolation, Time Warp	None	N/A	N/A	N/A	N/A
2000	[51]	Latency Exposure	None	N/A	N/A	N/A	N/A
2000	[145]	Self-Prediction	User Study	Custom	Arcade	120–300 ms	8
2000	[98]	Extrapolation, Incoming Delay, Time Warp	Case Study	Task	N/A	0–300 ms	N/A
2001	[123]	Extrapolation	None	N/A	N/A	N/A	N/A
2001	[148]	Extrapolation	Experiment	Custom	Any	Unspecified	N/A
2002	[108]	Extrapolation	Experiment	Custom	Arcade, Racing, Sports	100–200 ms	N/A
2002	[129]	Extrapolation	None	N/A	N/A	N/A	N/A
2002	[92]	Incoming Delay	Experiment	Custom	FPS	0–300 ms	N/A
2003	[30]	Extrapolation	None	N/A	N/A	N/A	N/A
2003	[59]	Extrapolation	User Study	Task	N/A	0–480 ms	8, 18
2003	[41]	Extrapolation	Experiment	Custom	Arcade	Unspecified	N/A
2004	[1]	Extrapolation	Case Study	BZ Flag	FPS	100–800 ms	2, 4
2004	[100]	Incoming Delay, Time Warp	Experiment	Custom	Arcade	40– 2,000 ms	N/A
2004	[130]	Interpolation	None	N/A	N/A	N/A	N/A
2004	[58]	Latency Exposure	User Study	Custom	Arcade	0–1,400 ms	40
2004	[150]	Extrapolation	Experiment	Task	N/A	Unspecified	N/A
2005	[2]	Extrapolation	Experiment	BZ Flag	FPS	200–800 ms	4
2005	[74]	Latency Concealment, Self-Prediction	None	N/A	N/A	N/A	N/A
2005	[126]	Extrapolation, Latency Concealment, Self-Prediction	Case Study	Custom	Arcade	0–200 ms	N/A
2005	[149]	Outgoing Delay	Experiment	Quake 2	FPS	0–400 ms	4
2006	[14]	Extrapolation, Incoming Delay, Outgoing Delay, Self-Prediction	Experiment	Custom	Any	0–500 ms	N/A
2006	[71]	Extrapolation, Time Warp	None	N/A	N/A	N/A	N/A
2006	[91]	Incoming Delay, Time Warp	Experiment	Quake 3	FPS	50–200 ms	2
2006	[15]	Latency Concealment, Self-Prediction	Experiment	Custom	Casual	N/A	N/A
2006	[144]	Extrapolation, Latency Exposure, Outgoing Delay	User Study	Custom	Sports	50–250 ms	24
2006	[61]	Extrapolation	Experiment	Task	N/A	N/A	4
2006	[151]	Extrapolation, Incoming Delay	User Study	Custom	Arcade	100–800 ms	2
2006	[87]	Extrapolation	User Study	Custom	Arcade	0–3,000 ms	30
2006	[106]	Extrapolation	Experiment	BZ Flag	FPS	0–1,600 ms	N/A
2007	[23]	Incoming Delay, Self-Prediction	User Study	Task	N/A	0–900 ms	18
2007	[79]	Incoming Delay	User Study	Custom	FPS	25–200 ms	10
2007	[88]	Extrapolation	User Study	Custom	Arcade	1,000–3,000 ms	37
2007	[139]	Self-Prediction	Experiment	Task	N/A	0–50 ms	N/A
2008	[136]	Incoming Delay	User Study	Custom	Arcade	0– 500 ms	24
2008	[115]	Extrapolation	User Study	Custom	Arcade	N/A	3
2008	[97]	Control Assistance	User Study	Custom	Arcade	N/A	13
2008	[89]	Extrapolation	User Study	Custom	Arcade	0–3,000 ms	37
2008	[105]	Incoming Delay	Experiment	Custom	Any	2–40 ms	N/A
2009	[152]	Incoming Delay	Experiment	Street Fighter 2	Fighting	0–400 ms	2
2009	[65]	Extrapolation, Incoming Delay	User Study	Custom	Racing	Unspecified	Unspecified
2010	[122]	Extrapolation, Incoming Delay, Outgoing Delay, Self-Prediction	Experiment	Custom	Arcade	100–746 ms	N/A
2010	[73]	Outgoing Delay	Experiment	Quake 3 Arena	FPS	0–150 ms	N/A
2011	[8]	Control Assistance	User Study	Custom	Arcade	N/A	24
2012	[75]	Extrapolation	Experiment	Custom	Any	Unspecified	N/A
2012	[62]	Extrapolation, Incoming Delay	User Study	Custom	Arcade	0–150	30
2012	[78]	Extrapolation, Incoming Delay	User Study	Custom	Arcade	80, 130 ms	20
2013	[121]	Extrapolation, Incoming Delay, Interpolation	None	N/A	N/A	N/A	N/A
2013	[147]	Extrapolation	User Study	Quake 3, World of Warcraft	FPS, RPG	N/A	16, 200
2013	[127]	Attribute Scaling	None	Custom	Sports	100–200 ms	N/A
2013	[146]	Attribute Scaling, Incoming Delay	User Study	Custom	Fighting	10–40 ms	20
2014	[119]	Interpolation	User Study	Custom	Arcade	125–500 ms	18
2014	[120]	Extrapolation, Incoming Delay	User Study	Custom	Arcade	50–200 ms	26
2014	[142]	Control Assistance	User Study	Custom	FPS	N/A	N/A
2015	[66]	Control Assistance	User Study	Custom	Arcade	10–160 ms	18
2015	[83]	Speculative Execution	User Study	Doom 3, Fable 3	FPS, RPG	0–400 ms	41
2015	[86]	Interpolation, Time Warp	User Study	CS:GO	FPS	0–150 ms	4
2016	[54]	Extrapolation	Experiment	Custom	Arcade	N/A	N/A
2016	[140]	Self-Prediction	Experiment	N/A	N/A	50–100 ms	N/A
2016	[43]	Latency Concealment	None	N/A	N/A	N/A	N/A
2016	[69]	Extrapolation	Experiment	Custom	Racing	N/A	15
2017	[84]	Time Warp	User Study	Custom	FPS	0–250 ms	4
2017	[80]	Self-Prediction	User Study	Task	N/A	33–99 ms	16
2018	[90]	Outgoing Delay	User Study	Assault Cube	FPS	20– 80 ms	10
2018	[116]	Attribute Scaling	User Study	Need for Speed, Somi, Table Tennis	Arcade, Racing	0–400 ms	25, 27
2018	[85]	Time Warp	User Study	Custom	FPS	50–250 ms	12
2018	[32]	Extrapolation	Experiment	Custom	Arcade	0–300 ms	N/A
2018	[24]	Extrapolation	Experiment	Custom	Racing	N/A	N/A
2018	[6]	Self-Prediction	User Study	Task	N/A	0–66 ms	10
2019	[82]	Attribute Scaling	User Study	Flappy Bird	Arcade	0–200 ms	12
2019	[137]	Time Warp	User Study	Custom	Arcade	0–800 ms	30
2020	[40]	Extrapolation	Experiment	World of Warcraft	RPG	N/A	N/A
2020	[118]	Attribute Scaling	User Study	Flappy Bird	Arcade	10–400 ms	18
2020	[117]	Attribute Scaling, Control Assistance	User Study	Custom	Arcade	0–200 ms	194
2020	[76]	Latency Concealment	User Study	Custom	FPS	25–105 ms	9
2021	[12]	Latency Concealment	Case Study	Custom	Arcade	Unspecified	N/A
2021	[102]	Extrapolation	Experiment	Custom	Arcade	0–100 ms	N/A
2021	[18]	Attribute Scaling	User Study	Custom	Arcade	0–125 ms	23

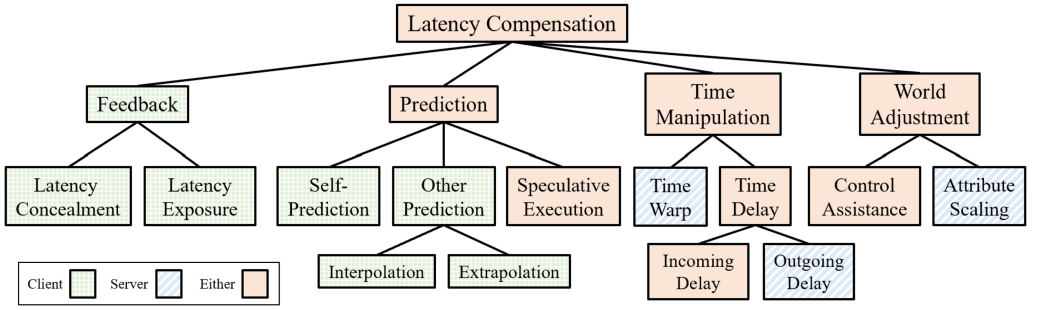

Fig. 3. A taxonomy of latency compensation techniques for network games.

At the top is *latency compensation*, grouping all latency compensation techniques.

Beneath latency compensation is the division of four classes of latency compensation: *feedback*, *prediction*, *time manipulation*, and *world adjustment*.

Each of these classes is further differentiated by families of techniques: *latency concealment* and *latency exposure* for feedback; *self-prediction*, *other-prediction*, and *speculative execution* for prediction; *time warp* and *time delay* for time manipulation; and *control assistance* and *attribute scaling* for world adjustment.

Some families of techniques are further refined: *interpolation* and *extrapolation* for other prediction and *incoming delay* and *outgoing delay* for time delay.

Although not shown in the figure, actual implementations of latency compensation techniques can also be differentiated based on algorithm parameters or code specifics.

Figure 3 also indicates by color and shading where the primary functionality for the compensation technique resides (see the legend in the bottom left corner): *Client* for techniques that are processed and displayed primarily on a player's client computer, *Server* for techniques that are primarily handled on the authoritative game server, and *Either* for techniques that can reside on either the client or the server or both.

Note that individual network games can, and often do, use more than one technique.

The rest of this section proceeds to define and describe the nodes in the taxonomy, providing an illustrative example for each "leaf" node in the picture. The examples are not necessarily the only latency compensation technique in each leaf node but are meant to provide clarity by an example.

4.1 Feedback

Feedback provides audible or visual information to the player based on latency, without actually changing the state of the game world.

4.1.1 Latency Concealment. Latency concealment visually masks latency from the client to the server so as to minimize the perception of unresponsiveness. For example, a game client might show the discharge and recoil of a weapon when the player pulls the trigger, even though the outcome of the shot (e.g., a hit or a miss) has not been registered and recorded by the server. As another example, a vehicle ordered to move by a player may immediately power up its engines, kicking up dust but not actually changing positions until the server has verified that movement is allowed.

Figure 4 depicts an example of latency concealment. On the left, at time t_0 the player is ready to act to move their avatar, the green circle. At time t_1 when the player provides input at the game client, the game responds by providing a visual effect (if this were a car, the engines might rev, kicking up dust), but the avatar does not actually change positions pending approval from the

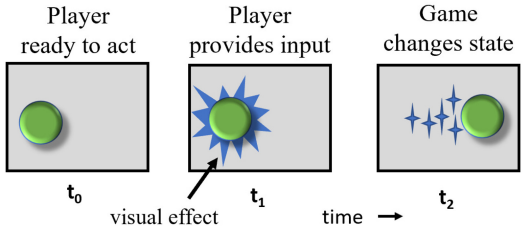

Fig. 4. An example of latency concealment.

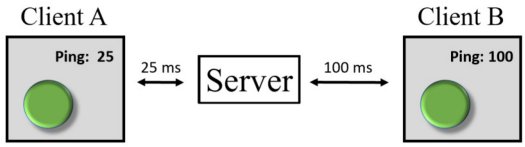

Fig. 5. An example of latency exposure.

authoritative server. At time t_2, the avatar actually moves since the server approval has arrived. The visual effect in this example conceals the latency from the client to the server by making the move command seem immediately responsive to the player, even though the actual position does not change until at least as long as the client-server latency.

Latency concealment includes image warping techniques that adjust the rendered image, ranging from simple camera view re-projection [131] to temporally updating the animated content [12, 43, 74, 76]. These concealment techniques are often used in virtual reality headsets (called *late warp*) to reduce latency for head motion to video output to help with simulator sickness, as well as other aspects.

Burgess, Hanna, Shelly, and Katchabaw [15, 74, 126] describe the use of a software design pattern that uses "padding" to hide latency. They make specific mention of using an animation played by the client in immediate response to a player's action while waiting for the latency from the server confirmation.

Efficacy. In general, there has not been much formal evaluation of latency concealment techniques. Some limited evaluation claims latency concealment can improve virtual reality head tracking with up to 380 ms of latency [131] and can reduce the "penalty" for in-game shooting with 80 ms of latency [76].

4.1.2 Latency Exposure. Latency exposure gives a visual indicator of the magnitude of the latency from the client to the server. For example, the client may decorate the corner of a display with a numeric value that directly reports the client's round-trip time to the server (often called the *ping* time by game players) or with "bars" that depict latency similar to those for mobile phone signal strength (i.e., more bars is a better, lower-latency connection).

Figure 5 depicts an example of latency exposure. There are two clients, Client A and Client B, each connected to the server in the middle. Client A has a round-trip time of 25 ms, and Client B has a round-trip time of 100 ms. The clients' displays expose their round-trip time latencies in the upper corner of their respective screen, noting this as a *ping* value, which is term commonly understood and used by network gamers.

Vaghi et al. [141] describe several mechanisms by which latency can be revealed to players: a gauge showing an amount of latency, an area of uncertainty (larger with more latency), "shadows,"

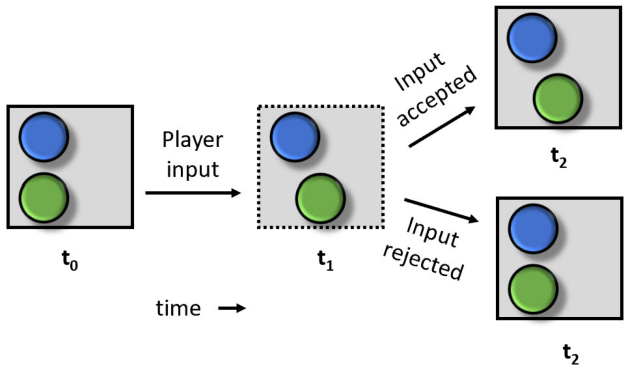

Fig. 6. An example of self-prediction.

and "ghosts" to represent the estimated position of objects based on latency, and visual emphasis indicating when events occur in time to make the presence of latency known.

Similarly, Fraser et al. [51] describe how the uncertainty due to latency can be rendered as a wireframe volume around an object, with the volume growing larger with an increase in latency, similar to the area of uncertainty.

Wikstrand et al. [144] use a "shadow" to represent the estimated position of a game avatar on the server based on the latency, similar to the shadows proposed by by Vaghi et al. [141].

Gutwin et al. [58], propose magnitude "decorators," which are visual representations for enhancing a user's awareness of latency. Decorator examples include numbers, translucency, color, and even pulsing or oscillation to indicate latency.

Efficacy. In general, there are few studies that actually evaluate the benefits of latency exposure. For the few that do, they suggest that latency exposure can improve player performance by about 25% for latencies as high as 800 ms [58] and increase player enjoyment by about a half-point on a 7-point scale for a latency of 500 ms [144].

4.2 Prediction

Prediction estimates the game state at a client without having the official game state from the server. Prediction takes advantage of the fact that the game client has information on the game world (e.g., object locations, world terrain) and processing capabilities (e.g., ability to compute physics and render the game world), providing the player with world representations before they are confirmed by the server. The drawback with predictive techniques is that they can be wrong in that they may not accurately represent the game world as ultimately determined by the authoritative server. In such cases, there is inconsistency between the client game state and the server game state so that when the client game state is inevitably corrected to align with the server game state, the fix is noticeable and may even be jarring to the player.

4.2.1 Self-Prediction. Self-prediction predicts the game state based on player input but before getting confirmation from the server. Self-prediction is a natural technique for game programmers since most clients run a fully functional game engine able to incorporate player input and compute game object interactions (e.g., physics, including collisions) without the server, and doing so can provide immediate feedback for the player.

Self-prediction is often called *client-side prediction* in papers, online blogs, and player posts.

Figure 6 depicts an example of self-prediction. A time t_0 on the left, the player has a view of the game world that is consistent with that of the server (not shown). At time t_1, the player has

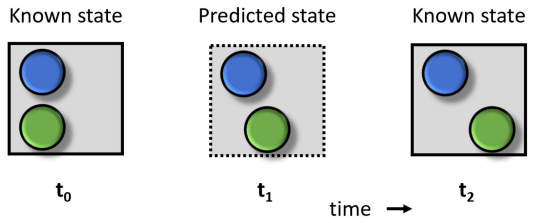

Fig. 7. An example of interpolation.

provided some game input (e.g., press the right arrow key) to move the green avatar to the right. The client assumes that this movement will be allowed by the server and so renders the world with the green avatar in the predicted location. Once the server receives and then responds to the player input at time t_2, there are two possibilities: in the first case, shown at the top, the server has accepted the input and the green avatar's new position is confirmed; in the second case, shown at the bottom, the server has rejected the player input (e.g., if the avatar is blocked by another object unknown to the client) and the client renders the world as specified by the server.

Burgess, Hanna, Shelly and Katchabaw [15, 74, 126] describe the use of a software design pattern based on the notion of "optimism," where a prediction on the client is assumed to be valid until the server does the official computation and returns the results. They provide additional techniques for synchronization and consistency checking.

Chen et al. [23] use a form of self-prediction called an *echo* to immediately show the player the effects of an action, even if additional delay is incurred before the official confirmation.

Wu and Ouhyoung [145] study "look-ahead" algorithms for 3D, head-mounted displays that predict object position and orientation. They study algorithms with different prediction complexities—simple to more complex. Also for a virtual environment with a head-mounted display, Tumanov et al. [139] describe predictions of "poses" for a player based on the position and orientation in the motion space.

Brun et al. [14] mention self-prediction as it affects game state consistency in multiplayer network games. They provide specific context for fairness in multiplayer games, where players may have different amounts of latency.

Le et al. [80] capture movements of the hand in a touch interface, use these movements in a neural network, and predict the location of future touch positions. Similarly, Antoine et al. [6] use high-frequency data gathered from a computer mouse to predict the future velocity and position.

Efficacy. Evaluation of self-prediction is fairly limited given the range of conditions to which it can be applied. Self-prediction for steering tasks on personal computers within virtual reality and on touch interfaces show about 15& to 20% improvement to user performance for latencies around 100 to 150 ms [6, 23, 145] with larger improvements (up to 80%) for higher latencies.

4.2.2 Other-Prediction. Other-prediction predicts the game state for objects controlled by other players (including AI-controlled players or objects) without having the actual state information from the server.

4.2.2.1 Interpolation. Interpolation predicts past states for objects controlled by other players based on the current state and previously known states. For example, the position of a vehicle can be interpolated to be in between a past known location and its current known location.

Figure 7 depicts an example of interpolation. At the left at time t_0, the client displays the known state of the game world at that time. At the right at time t_2, the client has gotten an update on the world state. At time t_1 in the middle, the client interpolates the position of the green avatar

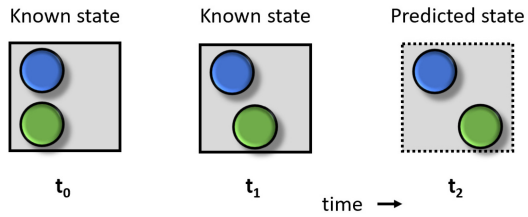

Fig. 8. An example of extrapolation.

based on the known position in the past at t_0 and the known position in the future at t_2. The client then renders this predicted state. Interpolation is typically used to visually smooth out game states rendered on the client in cases where the visual update rate happens more often than do updates received from the server.

Lee and Chang [86] evaluate how interpolation in the commercial FPS game Counter-Strike: Global Offensive (Valve, 2012) improves player accuracy.

Sharkey et al. [124] describe interpolation via *local perception filters* that provide for a continuous view of the game world (i.e., without any abrupt transitions) as client game states are updated with remote user actions. As Smed et al. [130] describe, interpolation via local perception filters can be tuned to keep predictions closer to actual game state for local players than for remote players to keep a smooth view of the game state while mitigating the effects of latency.

Savery and Graham [121] provide a toolkit for game developers to implement interpolation, including local perception filters and also smooth corrections to interpolate rendered game state toward the actual game state when a client gets updates. The authors also evaluate how smooth corrections are perceived by players compared to abrupt corrections when fixing an inconsistent game state [119].

Efficacy. There is little formal evaluation of the impact of interpolation on users. For player performance, interpolation may improve shooting accuracy by 2% [86] for FPS games and decrease annoyance with game state inconsistencies by about 50% for arcade-style games [119].

4.2.2.2 Extrapolation. Extrapolation predicts future states for objects controlled by other players assuming current behaviors continue. For example, the position of a vehicle can be extrapolated to a future location based on its last known position and kinematic state (velocity, acceleration, orientation, and angular velocity).

Figure 8 depicts an example of extrapolation. At the left at time t_0, the client display renders the known state of the game world at that time. In the middle at time t_1, the state of the world has been updated by the server and the client displays the green avatar's new position, moving to the right. At the right at time t_2, the client has not (yet) gotten an update on the world state but can extrapolate the position of the green avatar based on the last known position at time t_1 and the avatar's velocity. The client then renders this predicted state.

Since extrapolation techniques are by far the most widely researched latency compensation technique in the literature, we provide additional details on the well-known extrapolation technique called *dead reckoning* [143]. With dead reckoning extrapolation, object predictions are done with basic kinematic physics such as the last known position, velocity, and acceleration. Assume that p_t is the position at time t and the last update for an object's position was received at time t_0. With the simplest form of dead reckoning extrapolation, a client could assume that the location of the object at time t_1 is the same as the location of the object at time t_0:

$$p_{t_1} = p_{t_0}. \tag{1}$$

```
repeat
  receive new packet from server
  extract state update data {pos, vel, accel}
  if seen object before then
    update object attributes
  else
    add object to list
  end if
  for each object in list
    update predicted position // Equation 3
  end for
  render frame on screen
\vspace*{-10pt}
```

Listing 1. Client dead reckoning extrapolation algorithm.

```
repeat
  receive input from object controller
  update current position based on received input
  compute predicted position based on previous {pos, vel, accel} // Equation 3
  if (current position - predicted position) < threshold then
    marshall {pos, vel, accel} data
    send data in packet to client
  end if
\vspace*{-10pt}
```

Listing 2. Server dead reckoning extrapolation algorithm.

Assuming a constant velocity (v_{t_0}), using the velocity at time t_0, a slightly more sophisticated algorithm could predict the location of the object to be

$$p_{t_1} = p_{t_0} + v_{t_0}(t_1 - t_0). \tag{2}$$

Adding information for a constant acceleration (a) (e.g., gravity or rocket thrust), the location of the object could be predicted to be

$$p_{t_1} = p_{t_0} + v_{t_0}(t_1 - t_0) + \tfrac{1}{2}a(t_1 - t_0)^2. \tag{3}$$

Psuedo-code for a client using the dead reckoning extrapolation algorithm to predict an object's position is given in Listing 1.

The server runs a similar computation, determining what the client predicts for the extrapolated position. If the client's predicted position deviates by more than a pre-set threshold, the server sends an update to the client with the correct position (as well as any updates to velocity and acceleration). Psuedo-code for the server is given in Listing 2.

In general, units with high inertia are easy to predict (i.e., a rock rolling down a hill or a player in freefall from an airplane), whereas models with little inertia are harder (i.e., a pixie with 360 degrees of movement freedom or an avatar that can teleport). Game-specific prediction algorithms can even be crafted. For example, a real-time strategy game may define what it means for a unit to "return to base." As long as the unit continues to return to base, all clients can accurately predict the extrapolated position over time without any updates.

The information and processing used in the extrapolated position can vary from simple to complex, depicted in Figure 9 as a single dimension. Extrapolation techniques on the left use basic physics to make a prediction, the simplest of which is the last known position and the velocity. Small increments in processing and information are required for acceleration and orientation, with a bit more for forces (e.g., friction) and mass. In the middle, extrapolation techniques might use higher-order information such as environmental factors, routes available for travel, or the

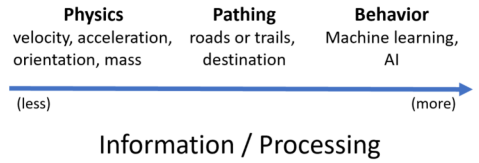

Fig. 9. Extrapolation information and/or processing dimension.

intended destination of the predicted object. On the right are extrapolation techniques that consider even more complex information, such as past behaviors, and complex processing such as machine learning and full AI control of the predicted objects.

Vaghi et al. [141] identify prediction as a way to anticipate future states (e.g., the "real" position of a ball) based on latency. Diot and Gauiter [38] implement a simple extrapolation technique for the open source game MiMaze [37] by replaying the position of a remote object if an update is not available when rendering. Mauve [99] describes how with extrapolation, mis-predictions may yield unexpected game states (e.g., dead players shooting).

Wikstrand et al. [144] describe extrapolation as one of several "predictor displays," stating that they work best when the extrapolated positions of objects are predictable. In their pong game implementation, they depict the extrapolation locations of paddles as shadows.

Shelly and Katchabaw [126] extrapolate the positions of server-controlled objects (e.g., an opponent's avatar) as part of their "optimistic" software design patterns to compensate for latency. Similarly, Jian et al. [71] describe extrapolation as an "optimistic technique," where the client computes and renders the location of server-controlled objects without waiting for the latency from the server response.

McCarty et al. [101] describe using dead reckoning for DIS (Distributed Interactive Simulations)—an IEEE standard for multiplayer combat simulation—for a virtual flight simulator. Mauve [98] describes dead reckoning use in DIS, including shortcomings to state consistency.

Smed et al. [129] describe dead reckoning as one of several aspects of networking in online games both for reduction in bitrates and for mitigating latency via extrapolation. Pantel and Wolf [108] propose extrapolations of two types: (1) positions of game objects via dead reckoning, and (2) positions of an input device (e.g., a joystick) which, in their racing game, results in moving a car to a predicted position.

Yu and Choy [148] and Zang and Georganas [150] propose adding an orientation threshold for sending dead reckoning updates above and beyond the position threshold typically used. Hanawa and Yonekura [61] reduce prediction errors in traditional dead reckoning by employing additional client-side computations that give more precision (e.g., by using a Taylor expansion). Cai et al. [16] extend dead reckoning by adapting the thresholds used to determine when to send updates based on the area of interest and sensitivity. Kharitonov [75] proposes and Almeida and Felinto [32] evaluate a dead reckoning extension that uses the motion patterns of the objects in addition to simple kinematic physics.

Aggarwal et al. [1] analyze the use of dead reckoning with synchronized clocks and timestamps to improve extrapolation accuracy after updates are received. The authors extend their ideas to fairness for players with different latencies by equalizing errors from incorrect extrapolations [2]. Zhang et al. [151] propose using Aggarwal et al.'s dead reckoning with synchronized clocks combined with incoming delay (see Section 4.3.2.1) to further reduce prediction errors. Hara et al. [62], Ishii et al. [65], and Kusunose et al. [78] also use extrapolation combined with incoming delay, providing a predicted position for updates outside of the incoming delay buffer. Jaya et al. [69] adjust the dead reckoning update threshold based on an interest management profile—the closer an entity is, the lower the threshold and vice versa.

Duncan and Gracanin [41] propose reducing possible state inconsistency in dead reckoning by pre-computing predicted states and sending updates *before* any inconsistency thresholds are breached. Palant et al. [106] explore extrapolation variants of dead reckoning, including using clock synchronization and a convergence algorithm that smooths out inconsistency in game states from extrapolation errors. Brun et al. [14] consider dead reckoning in terms of how it affects game state inconsistency for multiplayer games with different latencies for each player. Roberts et al. [115] propose an extension to dead reckoning whereupon state updates are sent using not just a threshold for spatial inconsistency but an accumulation of this inconsistency over time. Savery et al. [120–122], describe extrapolation, part of prediction, as a general technique to compensate for latency, with dead reckoning as a specific version. The authors provide a state consistency-centric view of the game world, including requirements on state divergence and different approaches to correct inconsistent states (e.g., smoothly or abruptly).

Li and Chen [87] extend dead reckoning by extrapolating object positions based on attraction (inferring that some objects may want to move closer to another) and repulsion (the opposite). Li et al. [88, 89] add the notion of personal habits, where the prediction takes into account actions that the predicted object would likely do based on past behavior. Similarly, Schirra [123] describes using the content of game objects to do dead reckoning using more than just position and velocity. For example, content-based dead reckoning might use "running along a path" or "playing a wall-pass to a team-mate" to extrapolate positions. Along these lines, Yahyavi et al. [147] consider the physics of a player avatar in predicting a location and the player's interest in their surrounding environment in terms of how it affects movement. Similarly, Gao et al. [54] describe modeling a player's behavior and using this behavior profile to extrapolate to positions of player-controlled objects. Their technique is combined with dead reckoning in an attempt to better predict actual avatar location in a pong game. Chen and Liu [24] extend traditional dead reckoning to an extrapolation technique that also uses human behavior and the virtual environment factors (e.g., the terrain) in doing predictions.

Duarte et al. [40] propose using machine learning to infer if dead reckoning is able to accurately extrapolate positions—if not, another machine learning algorithm is used to make the predictions.

Cronin et al. [30] examine how prediction techniques (e.g., dead reckoning) may be incompatible with cheat-detection algorithms since extrapolation may be exploited in some cases.

Efficacy. Studies evaluate latencies ranging from a low of about 100 ms [120, 144] to a high of several seconds [88, 89]. Most of the evaluations of extrapolation use simulations or traces to assess consistency, measured as the deviation from the predicted state on the client to the actual state on the server. There are few evaluations of the impact of extrapolation on player performance or QoE. In general, extrapolation studies with evaluation find that in the presence of latency, extrapolation reduces inconsistency between client game states and the server game states [1, 24, 32, 40, 41, 54, 61, 87, 106, 115, 122, 147, 150, 151]. The metrics for improvement reported vary, such as reducing inconsistency down from 5% to 0.5% for about 300 ms of latency [108, 122] to making under 100 ms of network latency unnoticeable [38] to improving consistency 10-fold from 100 to 200 ms of latency. Note that instead of improving responsiveness or reducing consistency, some research proposes using extrapolation to reduce network bitrates [16, 69, 75, 147, 148] since network messages with object updates do not need to be transmitted if clients can correctly extrapolate object positions.

The relatively few evaluations that involve users show that in addition to improving consistency, extrapolation also improves player QoE and fairness across players (exact amounts are not specified) [2, 62, 65, 120].

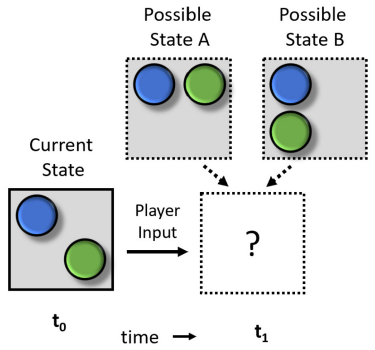

Fig. 10. An example of speculative execution.

4.2.3 Speculative Execution. Speculative execution computes the game world state based on possible player input before it has actually happened and adopts this pre-computed state if/when the input is provided. This enables the local game client to respond to player input immediately, matching the current game world to the pre-computed world, without waiting for the round-trip time to the server.

Figure 10 depicts an example of speculative execution. On the left at time t_0, the game client depicts the current game world state. At the top of the figure, the server has pre-computed and provided two possible game states that could exist at time t_1, named *State A* and *State B*. These two states are based on possible input that the player could provide to the game at time t_0—in this example, the player can either move the green avatar to the top right of the game world (resulting in State A) or to the bottom left of the game world (resulting in State B). Both of these states have been transmitted to the player before the player has provided input. When the actual input is provided, the state that corresponds to that input, either State A or State B, can be used immediately to represent the current game world state without the additional latency from client to server. The unused state is discarded.

Lee et al. [83] provide the only example of speculative executing found in the literature. Their *Outatime* system predicts future states based on a model of past user input and computes parallel states for input that is hard to predict. For cases of mis-prediction, Outatime uses corrective rendering to visually repair the mis-predictions and time warp with state check-pointing to limit error propagation.

Although not from a peer-reviewed forum (therefore, not part of our survey), Kopietz [77], from id Software (Doom and other titles), patented a speculative execution technique—rather than sending game states based on every possible combination of inputs, each input gets associated with a motion vector describing how the rendered game frame would change in response to that input. For some games, when there are multiple inputs, this may allow adding up vectors and applying the sum to a rendered image, short-circuiting latency to the server.

Efficacy. The evaluation of speculative execution finds that with the technique, players only see minor visual impairments for up to 128 ms of network latency and experience about a 1.5-point QoE improvement (on a 5-point scale) at 384 ms of network latency. Player performance with and without speculative execution is similar for 0 to 128 ms of latency, but dramatically better for 256 and 384 ms of latency.

4.3 Time Manipulation

Time manipulation alters the virtual time (i.e., the time in the game world) for computing the game state and/or resolving player actions.

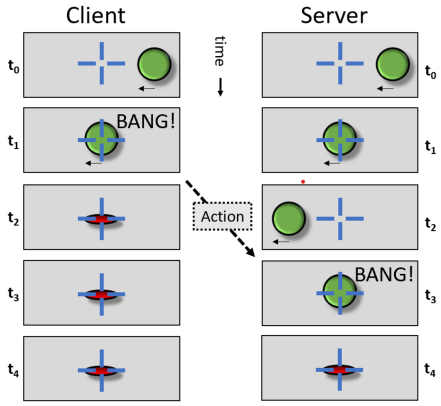

Fig. 11. An example of time warp.

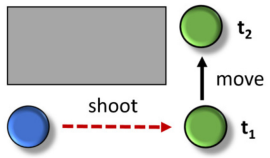

Fig. 12. An example of "shot around the corner."

4.3.1 Time Warp. Time warp rolls back the game state to when the player action occurred on the client, applies the action, then rolls the game state forward to the current time.

Figure 11 depicts an example of time warp. The figure shows the game world for a shooter game on a client and the server, with time advancing from top to bottom. The player on the client is shooting at a green avatar that is moving right to left, with the "plus" sign in the middle representing a weapon reticle. At time t_0 at the client, the green avatar is to the right of the reticle, moving into the reticle at time t_1 where the player pulls the trigger, and that action is sent to the server arriving just after time t_2. Meanwhile, on the server, the green avatar moves past the reticle at time t_1 and has continued right at time t_2. When the action arrives at the server, the server "warps" time back to when the action occurred at the client at time t_1, applying the action to the world representation at that time.

However, resolving an event in the past and rolling it forward may cause already rendered client game states to be inconsistent with the new game state. This is a well-known artifact of some shooting games and can result in "shot around the corner" as it is known by players, shown in Figure 12. At time t_1, at the blue avatar's client, the green avatar is in sight and the blue player fires. However, by time t_2 when the server receives the update, the green avatar has reached a safe position around the corner from the blue player and cannot be targeted. However, with time warp, the server, upon receiving the blue player's action at time t_2, rolls back time to the green avatar's position at time t_1 and applies the action. This hits the green avatar. Rolling the game world forward to the present time, with the green avatar hurt or killed, may feel like being "shot around the corner" for the green player.

Time warp is often called, somewhat confusingly, *latency compensation* or *lag compensation* in some papers and, more often, in online blogs and player posts. It is also sometimes simply called *rollback*. Also somewhat confusing, some image warping techniques (e.g., [43]) are called *time warp*, whereas we classify them as latency concealment (see Section 4.1.1).

Jefferson [70] first proposed *virtual time* as a paradigm for distributed computation, with time warp as an implementation. Although multiplayer games were not identified as a use at that time, distributed discrete event simulations were.

Mauve [99] describes how timestamps and time warp can be used to overcome mistakes made using extrapolation. In particular, an extrapolated state update may miss a key event, such as a player being killed that time warp can roll back to correct. Mauve [98] and Mauve et al. [100]

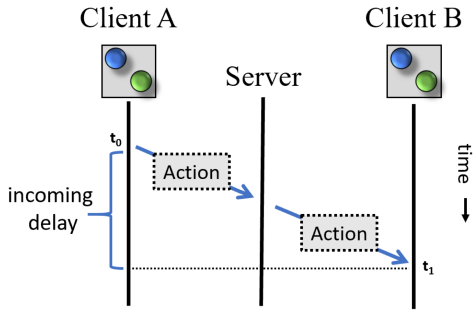

Fig. 13. An example of incoming delay.

provide a formalization of time warp in the context of continuous, interactive media, such as computer games. Jiang et al. [71] mention time warp in their survey as a means to overcome inconsistencies caused by prediction. Liang and Boustead [91] assess time warp both with and without incoming delay (see Section 4.3.2.1).

Lee and Chang [84, 86] describe and evaluate the "shot around the corner" problem in commercial FPS games, illustrated in Figure 12, whereby time warp can undo a player's move to a safe location to being damaged. Subsequently, Lee and Chang [85] describe how commercial FPS games provide a limit on how far back a server rolls back time and propose an advanced time warp technique to prevent "shot around the corner" whereby a client can identify that the player is currently safe (using their local time) and prevent rollback.

Sun and Claypool [137] implement and evaluate time warp for a cloud-based game streaming system.

Efficacy. Evaluation of time warp is fairly limited. Time warp may improve player aiming accuracy in an FPS game by 5% for 150 ms of latency [86] and overcome up to 100 ms of network latency [91], and improve the score in an arcade-style game by 50% for 400 ms of latency [137]. Time warp's "cost" to players is a reduced game state consistency with 8% of all hits in an FPS being "shot around the corner" for 250 ms of latency and a decrease in player perception of consistency by about a half-point on a 5-point scale for 100 to 800 ms of latency [137].

4.3.2 Time Delay. Time delay buffers game state updates to provide for more uniform latency across clients. For example, delaying updates by $D - l_i$, where l_i is the latency for each client and D is the $max(l_i)$, provides an equal latency for all players. Equal latencies across players can make the game more fair and game states more consistent.

4.3.2.1 Incoming. Delay buffers player actions before applying them so that actions arrive (and are applied) at all clients simultaneously.

Figure 13 depicts an example of incoming delay. The figure depicts downward timelines on a server and two clients, Client A and Client B. At time 0, player A responds to the game with an action. Client A immediately sends that action to the server and then to Client B. Because of the latency for Client A to the server and the server to Client B, that action does not arrive at Client B until time t_1. To have the action executed simultaneously on both clients, Client A adds a incoming delay equal to the latency from Client A to the server plus the latency from the server to Client B before applying the action so that Client A also applies the action at time t_1. Note that although the extra delay is shown at the client in this example, it can also be used at the server. Incoming delay is called *local lag* by some researchers.

Mauve et al. [98, 100] apply and formalize incoming delay as depicted in Figure 13. Savery and Graham [121] provide a toolkit for implementing incoming delay.

Diot and Gautier [38] describe a bucket synchronization incoming delay whereby all game state updates received by a client are stored (delayed) until their corresponding time interval.

Lin et al. [92] use incoming delay on both clients and server to synchronize game states temporally, calling their technique *sync-in* and *sync-out*.

Paik et al. [105] apply incoming delay at the server to account for different client latencies, with the delay time adjusted based on the number of clients within virtual proximity to each other.

Savery et al. [120] use incoming delay to improve consistency. Liang and Boustead [91] combine incoming delay with time warp (see Section 4.3.1) and assess the impact on player performance. Chen et al. [23] combine self-prediction (see Section 4.2.1) with incoming delay: the incoming delay reduces state inconsistency across players, and the self-prediction alleviates some of the reduced responsiveness caused by the extra delay. Zhang et al. [151] combine incoming delay with extrapolation (see Section 4.2.2.2): incoming delay to reduce inconsistency before updates are received and extrapolation to reduce inconsistency after updates are sent. Stuckel and Gutwin [136] evaluate incoming delay as well as incoming delay plus the self-prediction "echo" by Chen et al. [23]. Hara et al. [62], Ishii et al. [65], and Kusunose et al. [78] use a standard incoming delay technique that buffers incoming game state updates up to the buffer period, then predicts (extrapolates) object positions that do not have an update.

Brun et al. [14] mention using incoming delay as a way to adjust fairness among all players (i.e., players closer to the server are delayed to the same latency as players farther away), which has a tradeoff of decreasing responsiveness for some players. Similarly, Le and Liu [79] use incoming delay on the server by holding those packets from clients with lower than average delay so as to equalize latency across all clients.

Savery et al. [122] propose incoming delay in two forms: (1) immediately send out a player's input but delay applying it to reduce inconsistency across players, and (2) an analogy to streaming media applications whereby a state arriving from a game server is not immediately applied but rather buffered to smooth out playback (they call this *remote lag*). Xu and Wah [146] describe an incoming delay approach as well as a hybrid scheme combining incoming delay and prediction (see Section 4.2)—in this case interpolation to smooth out updates. Zhao et al. [152] use incoming delay to smooth input in a system supporting legacy games.

Efficacy. Evaluation of incoming delay using simulation shows that it can eliminate state inconsistency for games with up to 500 ms of network latency [38, 65, 92, 105, 122, 146, 151]. The "cost" for more consistency is a reduction in the responsiveness for some players, reducing performance in FPS games by 50% for 200 ms of delay [91]. Although there are general tradeoffs in the reduced responsiveness for the added delay and the improved consistency [120], for 250 ms of network latency, player performance may increase 30% but with little impact on QoE [136].

4.3.2.2 Outgoing. Delay buffers remote game state updates before sending to provide equal latencies across clients. Outgoing delay is similar to incoming delay, but whereas incoming delay adds the delay after receiving a message (e.g., a game action), outgoing delay adds the delay *before* sending a message.

Figure 14 depicts an example of outgoing delay. The figure depicts timelines advancing top to bottom on a server and two clients, with Client A having a lower latency to the server than Client B. At time t_0, the server has a game world update message to send to both Client A and Client B and sends the message to Client B. However, since Client A has a lower latency than Client B, the server delays sending the message until time t_1. Setting the outgoing delay $(t_1 - t_0)$ to the

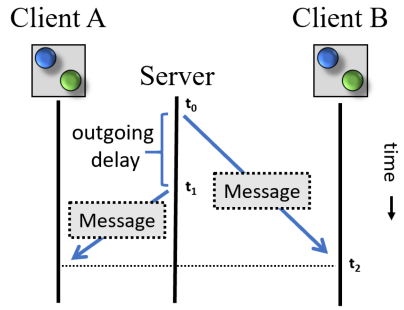

Fig. 14. An example of outgoing delay.

difference in latency for Client B and Client A means that the message arrives at Client A and Client B simultaneously, at time t_2.

Zander et al. [149] implement outgoing delay on a game server to add delay to outgoing messages to each player so as to equalize their latencies. The maximum total target delay threshold is adjusted depending upon game type.

Brun et al. [14] propose using outgoing delay in servers so as to manage fairness across clients with different latencies.

Wikstrand et al. [144] describe an "input buffering" scheme that uses outgoing delay to add latency to outgoing messages to other players so as to equalize their arrival times (they consider a peer-to-peer architecture).

Kaiser et al. [73] implement outgoing delay by concatenating game update messages at the client into one larger packet for an intended delay period before sending to the server. The server does something similar for updates to the clients.

Li et al. [90] implement outgoing delay for a cloud-based game platform where the amount delayed is inversely proportional to the latency for each client. They propose a maximum threshold to avoid extremely high latencies that might be required to accommodate some clients.

Efficacy. Outgoing delay has not been evaluated as extensively as incoming delay. For performance, outgoing delay can improve fairness for FPS players for up to 400 ms of latency, evening out a difference of about 30% in performance between two players [149]. For player QoE, outgoing delay can improve perceived fairness by 2 points on a 5-point scale for 80 ms of latency [90].

4.4 World Adjustment

World adjustment modifies game states to decrease difficulty akin to configurations with lower latency.

4.4.1 Control Assistance. Control assistance adjusts the outcome of player inputs to accommodate for inaccuracies due to latency.

Figure 15 depicts an example of one type of control assistance: "target magnetism." The blue avatar aims and shoots at the green avatar. Without assistance, the shot would miss, as indicated by the solid red line. However, with the target magnetism control assistance technique, the bullet trajectory is altered to bend toward the green avatar, making it easier for the blue player to hit the target. Other control assistance technique examples include "sticky targets" that reduce the cursor gain while near a target and "aim dragging" that cause the cursor to follow the direction of an intended target. The amount of assistance provided can be set proportional to the latency.

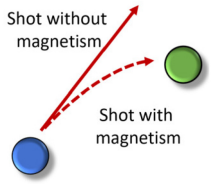

Fig. 15. An example of control assistance.

Mandryk and Gutwin [97] test the utility and perceptibly of the sticky targets control assistance technique applied to a computer mouse, albeit without additional latency.

Bateman et al. [8] compare three control assistance techniques for mouse pointing: cursors that cover an area, "gravity" toward a target, and the aforementioned sticky targets.

Vicencio-Moriera et al. [142] examine control assistance for aiming with a mouse in an FPS game considering five techniques: automatically locking on a target; bullet magnetism as in Figure 15; and the aforementioned area cursor, sticky targets, and target gravity. Again, their focus is on ability to help the player, not necessarily to overcome latency.

Ivkovic et al. [66] study the ability of control assistance for aiming a computer mouse in an FPS game and tracking a target. Their test conditions are for local system latency, not network latency, but pertain to cloud-based game streaming with network latency. Sabet et al. [117] also examine control assistance for cloud-based game streaming—in their case, for area cursors in 2D games with mouse-based aiming.

Efficacy. Most of the referenced studies evaluate local latency only, not network latency. In general, they show that control assistance can improve player performance in 2D and 3D aiming games in the presence of latency, improving player performance by up to 40% [66, 97, 142]. For multiplayer games, control assistance can reduce score differential in shooting gallery games and make the game more fun when players have disparate skills [8]. For low levels of correction, control assistance is not readily noticeable by players [97] and can improve input quality by about a half-point on a 5-point scale with about 200 ms of latency.

4.4.2 Attribute Scaling. Attribute scaling increases or decreases numeric attributes of objects and game world parameters to adjust game difficulty so as to make player actions easier to complete with higher latency. For example, decreasing the size of obstacles can make them easier to avoid during navigation, or increasing the speed of the player's avatar can make it more nimble. The intent can also be to make the action more resilient to mistakes. For example, avatar durability can be increased by adding extra lives or increasing the health or armor attributes.

Figure 16 depicts an example of attribute scaling for a game in which a player tries to navigate the green avatar from left to right through narrow gaps between obstacles. There are two different network latency scenarios depicted for the same game condition. On the left is a low-latency scenario that has relatively narrow gaps between the obstacles since the player's control is not significantly impacted by the latency. In contrast, on the right is a high-latency scenario in which the game attributes have been scaled to make the gaps between the obstacles larger since the player's control is more sluggish with the higher latency.

Vaghi et al. [141] mention adapting the pace of interaction required by a game with latency (i.e., the user interaction rate would slow down with higher latency).

Xu and Wah [146] extend the hitting time in a fighting game based on latency, where higher-latency players have a longer time period to execute a strike.

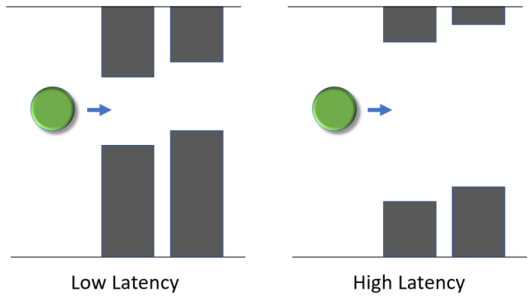

Fig. 16. An example of attribute scaling.

Shen and Zhou [127] change the speed of the ball in a pong game in relation to latency and the object's proximity to a player-controlled avatar that may interact with it (i.e., the ball slows down when nearing a player controlled object when there is high latency).

Sabet et al. [116] evaluate adjusting game attributes in response to higher latency by (1) increasing the deadline for a game action (i.e., giving the player more time to execute an action) and (2) decreasing the precision required for a game actions. Sabet et al. [117] extend this work with additional attribute scaling for (3) predictability, where randomness is decreased with latency; (4) the number of required actions per minute, where the number of required actions decreases with latency; and (5) a consequence where the impact of a detrimental action (e.g., a collision) is decreased with higher latency.

Lee et al. [82] propose adjusting the geometry of a game world based on latency. Specifically, they adjust the gap distance between pipes based on latency for the game Flappy Bird. Salay and Claypool [118] evaluate the same—pipe gap distance based on latency for Flappy Bird—in addition to allowing the player to manually adjust game attributes in response to latency.

Carlson et al. [18] study how adjustments to attributes for two custom gamesan FPS game and a rhythm game—affect players' scores with and without network latency. They then derive models they propose to be used to scale the games at runtime based on the players' experienced network latencies.

Efficacy. In general, attribute scaling has been shown to benefit player performance and QoE in the presence of network latency [18]. Attribute scaling can be effective for helping players perform 50% better with network latency for an arcade-style game with 400 ms of latency [118] and perform as well with network latency as without it for an arcade-style game with 80 ms [82] of network latency. Attribute scaling has been found to benefit player QoE by up to 1 point on a 5-point scale for up to 400 ms of latency [117] and reduce players' perception of delay from 85% down to 65% for 40 ms of latency in a fighting game [146].

4.5 Summary

Table 2 provides a summary table of the taxonomy in Figure 3 with the techniques rotated and presented top to bottom. For each technique, the corresponding publications are shown on the right.

From the table, extrapolation is the most published latency compensation technique, with 41 papers (about half) in this area, more than twice that of the next closest—incoming time delay with 19. Correspondingly, other-prediction and time delay are the most published techniques, belonging to the general group of prediction and time manipulation, respectively.

Table 2. Publications for Each Latency Compensation Technique

Technique				Publication
Latency Compensation	Feedback	Concealment		[12, 15, 43, 74, 74, 76, 126, 131]
		Exposure		[51, 58, 141, 144]
	Prediction	Self-Prediction		[6, 14, 15, 23, 74, 80, 126, 139, 145]
		Other-Prediction	Interpolation	[86, 119, 121, 124, 130]
			Extrapolation	[1, 2, 14, 16, 24, 30, 32, 37, 38, 40, 41, 61]
				[54, 62, 65, 69, 71, 75, 78, 87–89, 98]
				[99, 101, 102, 106, 108, 115, 120–123, 126]
				[129, 141, 144, 147, 148, 150, 151]
		Speculative Execution		[83]
	Time Manipulation	Time Delay	Incoming	[14, 23, 38, 62, 65, 78, 79, 91, 92, 98]
				[100, 105, 120–122, 136, 146, 151, 152]
			Outgoing	[14, 73, 90, 144, 149]
		Time Warp		[70, 71, 84–86, 91, 98–100, 137]
	World Adjustment	Control Assistance		[8, 66, 97, 117, 142]
		Attribute Scaling		[18, 82, 116–118, 127, 141, 146]

Note: Color intensity is proportional to the number of publications.

5 LATENCY COMPENSATION IN COMMERCIAL NETWORK COMPUTER GAMES

The objective of this section is to provide evidence of commercial network games using latency compensation techniques. Although there are invariably some non-commercial, open source games that have also used latency compensation, ascertaining this from the many games that are open source is beyond the scope of this survey. Moreover, the most popular games in the world are not open source but are instead commercial games, hence our focus here.

Unfortunately, most commercial games are "black boxes," wherein it is difficult to ascertain what, if any, latency compensation techniques are being deployed. However, the developers for some games have provided specific mention and even details about latency compensation techniques used in their software. This section provides an overview of the techniques and the games found in these developer presentations and blogs.

The games listed are not meant to indicate that these are the only games that use latency compensation techniques; far from it—it is our expectation and experience that most if not all AAA multiplayer network games use some form of latency compensation. Instead, this section includes known games where the game developers themselves describe, through either Web documents or online presentations, the latency compensation techniques implemented in their games.

In locating these presentations and blogs, we started with a list of four blogs we knew about previously.

We then searched two specific online forums that are known for having technical content posted by game developers: the *Game Developer's Conference*[1] and *Gamasutra*.[2]

Next, we visited forums that are maintained by the publishers for the top-10 eSports games [113] (the list is based on viewership and prize pools) and searched for latency compensation posts made by game developers. The list of games with forum citations are shown in Table 3, given in list order.

Last, we did general Internet searches for latency compensation presentations and blogs including only those made by game developers about their own commercial games.

[1] https://gdconf.com/.
[2] https://www.gamasutra.com/.

Table 3. Top 10 eSports Games Searched for Latency Compensation Posts

Game	Publisher	Year	Forum
League of Legends	Riot Games	2009	[81]
CS:GO	Valve Corporation	2012	[31]
Fortnite	Epic Games	2017	[42]
DOTA2	Valve Corporation	2013	[39]
Hearthstone	Blizzard Entertainment	2014	[48]
Call of Duty	Activision	2003	[27]
Overwatch	Blizzard Entertainment	2016	[49]
Rainbow 6 Siege	Ubisoft	2015	[50]
Rocket League	Psyonix	2015	[134]
NBA 2K	Sega Sports	1999	[47]

Table 4. Developer Presentations and Blogs with Details on Latency Compensation Implementations for Commercial Network Games

Year	Link	Compensation Technique	Game	Genre	Range
1996	[19]	Self-Prediction	QuakeWorld	FPS	Up to 200 ms
2001	[11]	Extrapolation	Age of Empires	RTS	Unspecified
2001	[10]	Extrapolation, Interpolation, Time Warp	Half-life	FPS	Unspecified
2011	[3]	Extrapolation, Incoming Delay, Latency Exposure	Halo	FPS	Up to 300 ms
2012	[17]	Self-Prediction, Extrapolation, Time Warp	GGPO (library)	Fighting	Unspecified
2016	[56]	Interpolation, Self-Prediction	Call of Duty: Black Ops III	FPS	Unspecified
2016	[46]	Extrapolation, Incoming Delay, Interpolation, Self-Prediction, Time Warp	Overwatch	FPS	Unspecified
2016	[138]	Extrapolation, Time Warp	MechWarrior Online	FPS	Unspecified
2017	[34]	Extrapolation, Interpolation	Watch Dogs 2	RPG	Unspecified
2017	[28]	Self-Prediction, Time Warp	Half-life	FPS	Up to 200 ms
2018	[33]	Extrapolation	Rocket League	Sports	Up to 600 ms
2018	[29]	Extrapolation	Rocket League	Sports	Unspecified
2018	[133]	Incoming Delay, Time Warp	Injustice 2, Mortal Kombat	Fighting	Up to 300 ms
2018	[5]	Interpolation, Time Warp	Unity's FPS Sample Game	FPS	Up to 200 ms
2019	[60]	Extrapolation, Incoming Delay	COD: Modern Warfare	FPS	Unspecified
2019	[110]	Incoming Delay, Time Warp	Various Fighting games	Fighting	Up to 150 ms
2019	[45]	Extrapolation, Incoming Delay	Overwatch	FPS	Up to 1,000 ms
2020	[135]	Time Warp	Valorant	FPS	Unspecified
2020	[114]	Time Warp	Valorant	FPS	Unspecified
2020	[35]	Incoming Delay, Latency Exposure, Self-Prediction	Valorant	FPS	20–50 ms

Table 4 summarizes the results. *Year* is when the presentation/blog was published. *Compensation Technique* maps the latency compensation technique(s) in the presentation/blog to our taxonomy—note that many indicate that their games implement more than one technique. *Game* provides the names of the game(s) described in the presentation/blog. *Genre* provides for the general category of game to which the targeted games belong: "FPS" (first-person shooter), "RPG" (role playing game), "Fighting," and "Sports." *Range* indicates the latency values mentioned in the presentation/blog, or "unspecified" if none.

From the table, extrapolation (11) is referenced the most, followed by time warp (19), incoming delay (7), self-prediction (6), interpolation (5), and latency exposure (2). About two-thirds (14 of 20) of the references are to FPS games, probably owing to their sensitivity to network latency [26] and heavy use in competitive gaming (e.g., eSports).

6 DISCUSSION OF TAXONOMY

Based on the sheer number of publications, there has been the most research done in latency compensation via prediction, specifically extrapolation. However, although prediction is well researched in general, the area of speculative execution is not. Improvements to speculative execution may rely upon emerging AI techniques to predict player input and hence game state before it happens. Although predicting player actions in general may be difficult, doing so for the constrained conditions provided by many games seems possible.

Manipulating the virtual time in game worlds has been fruitful, most prominently by delaying incoming actions to smooth input and provide fairness, and in "warping" time to resolve past actions. Commercial games make heavy use of both of these techniques.

Conversely, there is substantially less research in feedback, despite feedback perhaps being the most direct connection to the player via response to input. Latency exposure in the form of "ping" values is common to many games, but research on how such exposures mitigate latency as well as more advanced exposure approaches merit additional exploration.

There is considerable potential for research in world adjustments as well, with most control assistance techniques not having been assessed for latency and approaches to attribute scaling as broad and deep as games themselves.

In general, many latency compensation techniques could use additional evaluation to study a broader range of latencies but also with a broader set of users. Ideally, a representative sample from a user study would cover the broad demographic range of today's gamers.

Although the blogs obtained from game developers provide evidence of latency compensation in commercial games, the extent to which academic research has been incorporated into commercial products is not well known. In fact, the relatively narrow range of techniques in Table 4 suggests that many techniques proposed by the scientific literature may not yet be incorporated into commercial games.

Latency poses a particular challenge for a relatively recent game system type—that of cloud-based game streaming,[3] such as Microsoft's *Xcloud*, Amazon's *Luna*, NVidia's *GeForce Now*, Google's *Stadia*, and Sony's *PlayStation Now*. Unlike in traditional network games, with cloud-based game streaming the client does not have the game state nor does it typically do any "heavy-weight" processing such as 3D graphics rendering. Instead, the game frames are rendered at the cloud-based server and streamed down to the client similar to video. The client captures player input (e.g., mouse movements, button presses) and sends those back to the server for processing. In this type of system, the "thin" client cannot do the processing required by some of the latency compensation techniques. Specifically, the green shaded techniques in Figure 3 labeled "Client" cannot be done in cloud-based game streaming. Cloud-based game streaming systems generally can only apply the "Server" or "Either" techniques. This suggests the opportunity and challenge to develop new techniques to compensate for latency in cloud-based game streaming.

To make it easier to find the resources referred to in this article, the list of peer-reviewed research papers and developer presentations/blogs (with links) are available at the following URL:

https://web.cs.wpi.edu/~claypool/papers/lag-taxonomy/.

[3] https://en.wikipedia.org/wiki/Cloud_gaming.

7 SUMMARY AND FUTURE WORK

Computer games continue to grow in popularity for entertainment and professionally through eSports. Computer games are also often multiplayer, connecting computers over a network to enable geographically separated players to interact with each other in the same virtual world. These multiplayer network games need to exchange player actions and game states across the network fast and frequently to provide for a smooth, interactive player experience. Unfortunately, network latency adds delay for game updates from the server and player actions from the client, degrading the QoE by making the game less responsive and increasing inconsistency across client game states.

Latency compensation techniques use software at the client or server (or both) to mitigate the negative effects of network latency. Although some latency compensation techniques were established decades ago and are well used in the commercial game industry today, to the best of our knowledge, the space of latency compensation techniques has not been surveyed, nor have techniques been grouped based on their characteristics. Such a survey and organization can provide guidance for game and game system developers for techniques to deploy and for researchers who seek to invent new techniques.

We located and surveyed 85 peer-reviewed publications that dealt with latency compensation techniques. Our records list the type of techniques used and summary of evaluation, including game(s) studied, type of evaluation, and latency range. Our search also yielded evidence of latency compensation for commercial games, with 20 developer blogs and presentations detailing their latency compensation implementations. The latency compensation techniques are placed into a novel taxonomy based on similarity, with four main groups—feedback, prediction, time manipulation, and world adjustment—leading to 11 final technique types: concealment, exposure, self-prediction, interpolation, extrapolation, speculative execution, incoming delay, outgoing delay, time warp, control assistance, and attribute scaling. The most popular latency compensation categories based on peer-reviewed literature are prediction in the form of extrapolation and time manipulation in the form of incoming delay.

Our future work is to continue to develop and evaluate novel latency compensation techniques for multiplayer network computer games. In particular, there is a need for latency compensation techniques for cloud-based game streaming systems—since such systems cannot do heavyweight computation on the client, viable techniques must be server-side only. Our future work is also to assess latency compensation techniques on the increasingly broad range of gaming devices and inputs, such as touch screens and virtual reality.

REFERENCES

[1] Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. 2004. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'04)*. 161–165.

[2] Sudhir Aggarwal, Hemant Banavar, Sarit Mukherjee, and Sampath Rangarajan. 2005. Fairness in dead-reckoning based distributed multi-player games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'05)*. ACM, New York, NY, 1–10.

[3] David Aldbridge. 2011. I shot you first: Networking the gameplay of Halo: Reach. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=h47zZrqjgLc.

[4] Rahul Amin, France Jackson, Juan E. Gilbert, Jim Martin, and Terry Shaw. 2013. Assessing the impact of latency and jitter on the perceived quality of Call of Duty Modern Warfare 2. In *Proceedings of the 15th International Conference on Human-Computer Interaction: Users and Contexts of Use*. 97–106.

[5] Peter Andreasen. 2018. Deep dive into networking for Unity's FPS Sample game—Unite LA. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?app=desktop&v=k6JTaFE7SYI&ab_channel=Unity.

[6] Axel Antoine, Sylvain Malacria, and Géry Casiez. 2018. Using high frequency accelerometer and mouse to compensate for end-to-end latency in indirect interaction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'18)*. ACM, New York, NY, 1–11. https://doi.org/10.1145/3173574.3174183

[7] Grenville Armitage. 2003. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Proceedings of the 11th IEEE International Conference on Networks (ICON'03)*.
[8] Scott Bateman, Regan L. Mandryk, Tadeusz Stach, and Carl Gutwin. 2011. Target assistance for subtly balancing competitive play. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'11)*.
[9] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. 2004. The effects of loss and latency on user performance in Unreal Tournament 2003. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'04)*.
[10] Yahn Bernier. 2001. Latency compensating methods in client/server in-game protocol design and optimization. In *Proceedings of the Game Developers Conference (GDC'01)*. https://www.gamedevs.org/uploads/latency-compensation-in-client-server-protocols.pdf.
[11] Paul Bettner and Mark Terrano. 2001. 1500 Archers on a 28.8: Network programming in Age of Empires and beyond. *Gamasutra*. Retrieved March 6, 2022 from https://www.gamasutra.com/view/feature/131503/1500_archers_on_a_288_network_.php.
[12] Ben Boudaoud, Pyarelal Knowles, Joohwan Kim, and Josef Spjut. 2021. Gaming at warp speed: Improving aiming with late warp. In *Proceedings of the ACM SIGGRAPH Emerging Technologies Workshop*.
[13] Bob Briscoe, Anna Brunstrom, Andreas Petlund, David Hayes, David Ros, Ing-Jyh Tsang, Stein Gjessing, Gorry Fairhurst, Carsten Griwodz, and Michael Welzl. 2016. Reducing internet latency: A survey of techniques and their merits. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2149–2196.
[14] Jeremy Brun, Farzad Safaei, and Paul Boustead. 2006. Managing latency and fairness in networked games. *Communications of the ACM* 49, 11 (Nov. 2006), 46–51. https://doi.org/10.1145/1167838.1167861
[15] Shayne Burgess and Michael Katchabaw. 2006. Design and implementation of optimistic constructs for latency masking in online video games. In *Proceeedings of the 2nd Annual North American Game-On Conference (GameOn'NA'06)*.
[16] Wentong Cai, Francis B. S. Lee, and L. Chen. 1999. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the 13th IEEE Workshop on Parallel and Distributed Simulation (PADS'99)*.
[17] Tony Cannon. 2012. Fight the lag! The trick behind GGPO's low-latency netcode. *Game Developer Magazine*. Retrieved March 6, 2022 from https://drive.google.com/file/d/1cV0fY8e_SC1hIFF5E1rT8XRVRzPjU8W9/view.
[18] Edward Carlson, Tian Fan, Zijian Guan, Xiaokun Xu, and Mark Claypool. 2021. Towards usable attribute scaling for latency compensation in cloud-based games. In *Proceedings of the Game Systems Workshop (GameSys'21)*.
[19] John Carmack. 1996. Here is the new plan. *Blog*. Retrieved March 6, 2022 from https://fabiensanglard.net/quakeSource/johnc-log.aug.htm.
[20] Elie Cattan, Amelie Rochet-Capellan, Pascal Perrier, and Francois Berard. 2015. Reducing latency with a continuous prediction: Effects on users' performance in direct-touch target acquisitions. In *Proceedings of the International Conference on Interactive Tabletops and Surfaces (ITS'15)*.
[21] Nilanjan Chaudhuri, Swakshar Ray, Rajat Majumder, and Balarko Chaudhuri. 2009. A new approach to continuous latency compensation with adaptive phasor power oscillation damping controller (POD). *IEEE Transactions on Power Systems* 25, 2 (May 2009), 939–946.
[22] K. Chen, P. Haung, G. Wang, C. Huang, and C. Lee. 2006. On the sensitivity of online game playing time to network QoS. In *Proceedings of IEEE INFOCOM (INFOCOM'06)*.
[23] Ling Chen, Gen-Cai Chen, Hong Chen, Jack March, Steve Benford, and Zhi-Geng Pan. 2007. An HCI method to improve the human performance reduced by local-lag mechanism. *Interacting with Computers* 19, 2 (March 2007), 215–224.
[24] Youfu Chen and Elvis S. Liu. 2018. Comparing dead reckoning algorithms for distributed car simulations. In *Proceedings of the ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*.
[25] Mark Claypool. 2005. The effect of latency on user performance in real-time strategy games. *Elsevier Computer Networks, Special Issue on Networking Issues in Entertainment Computing* 49, 1 (Sept. 2005), 52–70.
[26] Mark Claypool and Kajal Claypool. 2006. Latency and player actions in online games. *Communications of the ACM* 49, 11 (Nov. 2006), 40–45.
[27] CoD Forums. n.d. Home Page. Retrieved March 6, 2022 from https://www.codforums.com/.
[28] Valve Developer Community. 2019. Source multiplayer networking. *Valvesoftware*. Retrieved March 6, 2022 from https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.
[29] Jared Cone. 2018. It IS rocket science! The physics of Rocket League detailed. *YouTube*. Retrieved March 6, 2022 from https://youtu.be/ueEmiDM94IE?t=23m33s.
[30] Eric Cronin, Burton Filstrup, and Sugih Jamin. 2003. Cheat-proofing dead reckoned multiplayer games. In *Proceedings of the 2nd International Conference on Application and Development of Computer Games*.
[31] CSGO Forums. n.d. Home Page. Retrieved March 6, 2022 from https://www.csgoforum.com/.
[32] Luis Fernando Kawabata de Almeida and Alan Salvany Felinto. 2018. Evaluation of the motion-aware adaptive dead reckoning technique under different network latencies applied in multiplayer games. In *Proceedings of the 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames'18)*.

[33] Rocket Science-Halfway Dead. 2018. RL netcode and lag explained—Rocket Science #16. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=c373LsgiXBc.
[34] Matt Delbosc. 2017. Replicating chaos: Vehicle replication in Watch Dogs 2. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=_8A2gzRrWLk.
[35] Matt deWet and David Straily. 2020. Peeking into Valorant's netcode. *Riot Games*. Retrieved March 6, 2022 from https://technology.riotgames.com/news/peeking-valorants-netcode.
[36] Matthias Dick, Oliver Wellnitz, and Lars Wolf. 2005. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of the ACM Workshop on Network and Systems Support for Games (NetGames'05)*.
[37] C. Diot and L. Gautier. 1998. Design and evaluation of MiMaze—A multi-player game on the internet. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*.
[38] C. Diot and L. Gautier. 1999. A distributed architecture for multiplayer interactive applications on the internet. *Network Magazine of Global Internetworking* 13, 4 (July 1999), 6–15. https://doi.org/10.1109/65.777437
[39] vBulletin. n.d. Dota2 Forum. Retrieved March 6, 2022 from https://dev.dota2.com/.
[40] Elias P. Duarte Jr., Aurora T. R. Pozo, and Pamela Beltrani. 2020. Smart reckoning: Reducing the traffic of online multiplayer games using machine learning for movement prediction. *Entertainment Computing* 33 (2020), 100336.
[41] Thomas Duncan and Denis Gracanin. 2003. Algorithms and analyses: Pre-reckoning algorithm for distributed virtual environments. In *Proceedings of the 35th Conference on Winter Simulation: Driving Innovation (WSC'03)*.
[42] Fortnite. n.d. Fortnite Forum. Retrieved March 6, 2022 from https://www.epicgames.com/fortnite/en-US/home.
[43] Daniel Evangelakos and Michael Mara. 2016. Extended timewarp latency compensation for virtual reality. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*.
[44] Parinaz Farajiparvar, Hao Ying, and Abhilash Pandya. 2020. A brief survey of telerobotic time delay mitigation. *Frontiers in Robotics and AI* 7 (Dec. 2020), 578805.
[45] Timothy Ford. 2019. Overwatch gameplay architecture and netcode. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?app=desktop&v=W3aieHjyNvw&ab_channel=GDC.
[46] Tim Ford and Philip Orwig. 2016. Fighting latency on Call of Duty: Black Ops III. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=vTH2ZPgYujQ.
[47] 2K Forums. n.d. Nba2k Forum. Retrieved March 6, 2022 from https://forums.2k.com/forumdisplay.php?214-NBA-2K.
[48] Blizzard. n.d. Hearthstone Forum. Retrieved March 6, 2022 from https://us.forums.blizzard.com/en/hearthstone/.
[49] Blizzard. n.d. Overwatch Forum. Retrieved March 6, 2022 from https://us.forums.blizzard.com/en/overwatch/.
[50] Ubisoft Forums. n.d. Rs6 Forum. Retrieved March 6, 2022 from https://discussions.ubisoft.com/category/607/rainbow-six?lang=en-US.
[51] Mike Fraser, Tony Glover, Ivan Vaghi, Steve Benford, Chris Greenhalgh, Jon Hindmarsh, and Christian Heath. 2000. Revealing the realities of collaborative virtual reality. In *Proceedings of the 3rd International Conference on Collaborative Virtual Environments (CVE'00)*.
[52] Sebastian Friston, Per Karlstrom, and Anthony Steed. 2016. The effects of low latency on pointing and steering tasks. *IEEE Transactions on Visualization and Computer Graphics* 22, 5 (May 2016), 1605–1615.
[53] Tobias Fritsch, Hartmut Ritter, and Jochen H. Schiller. 2005. The effect of latency and network limitations on MMORPGs: A field study of Everquest 2. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'05)*.
[54] Chen Gao, Haifeng Shen, and M. Ali Babar. 2016. Concealing jitter in multi-player online games through predictive behaviour modeling. In *Proceedings of the IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD'16)*.
[55] Steven Gargolinski, Christopher St. Pierre, and Mark Claypool. 2005. Game server selection for multiple players. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'05)*.
[56] Benjamin Goyette. 2016. Fighting latency on Call of Duty: Black Ops III. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=EtLHLfNpu84.
[57] Grand View Research. 2020. *Video Game Market Size, Share & Trends Report*. Report GVR-4-68038-527-4. Grand View Research. https://tinyurl.com/y4s5vfrm.
[58] Carl Gutwin, Steve Benford, Jeff Dyck, Mike Fraser, Ivan Vaghi, and Chris Greenhalgh. 2004. Revealing delay in collaborative environments. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'04)*.
[59] Carl Gutwin, Jeff Dyck, and Jennifer Burkitt. 2003. Using cursor prediction to smooth telepointer jitter. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work*. 294–301.
[60] Paul Haile and Mitch Sanborn. 2019. Call of Duty: Modern Warfare netcode explained! *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=tCpYV4k_izE.
[61] Dai Hanawa and Tatsuhiro Yonekura. 2006. A proposal of dead reckoning protocol in distributed virtual environment based on the Taylor expansion. In *Proceedings of the International Conference on Cyberworlds*. 107–114. https://doi.org/10.1109/CW.2006.10

[62] Yusuke Hara, Yutaka Ishibashi, Norishige Fukushima, and Shinji Sugawara. 2012. Adaptive delta-causality control scheme with dynamic control of prediction time in networked haptic game. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'12)*.

[63] O. Hohlfeld, H. Fiedler, E. Pujol, and D. Guse. 2016. Insensitivity to network delay: Minecraft gaming experience of casual gamers. In *Proceedings of the International Teletraffic Congress (ITC'16)*.

[64] Eben Howard, Clint Cooper, Mike Wittie, Steven Swinford, and Qing Yang. 2014. Cascading impact of lag on quality of experience in cooperative multiplayer games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'14)*.

[65] Naoki Ishii, Yutaka Ishibashi, and Shinji Sugawara. 2009. Enhancement of adaptive delta-causality control with adaptive dead-reckoning for multiplayer online games. In *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'09)*.

[66] Zenja Ivkovic, Ian Stavness, Carl Gutwin, and Steven Sutcliffe. 2015. 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 (CHI'15)*.

[67] J. Clementa. 2021. Number of Video Gamers Worldwide in 2020. Retrieved March 6, 2022 from https://tinyurl.com/y23x256z.

[68] R. Jota J. Deber, C. Forlines, and D. Wigdor. 2015. How much faster is fast enough? User perception of latency and latency improvements in direct and indirect touch. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'15)*.

[69] Iryanto Jaya, Elvis S. Liu, and Youfu Chen. 2016. Combining interest management and dead reckoning: A hybrid approach for efficient data distribution in multiplayer online games. In *Proceedings of the 20th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'16)*. IEEE, Los Alamitos, CA.

[70] David R. Jefferson. 1985. Virtual time. *ACM Transactions on Programming Language and Systems* 7, 3 (July 1985), 404–425. https://doi.org/10.1145/3916.3988

[71] Xinbo Jiang, Farzad Safaei, and Paul Boustead. 2005. Latency and scalability: A survey of issues and techniques for supporting networked games. In *Proceedings of the 13th IEEE International Conference on Networks (ICN'05)*.

[72] Katie Jones. 2020. Online gaming: The rise of a multi-billion dollar industry. *Visual Capitalist*. Retrieved March 6, 2020 from https://tinyurl.com/y5l5ppmt.

[73] Arnaud Kaiser, Nadjib Achir, and Khaled Boussetta. 2010. Improving energy efficiency and gameplay fairness for time-sensitive multiplayers games in MANET. In *Proceedings of the 2010 IEEE International Conference on Communications Workshops*. 1–5. https://doi.org/10.1109/ICCW.2010.5503904

[74] M. Katchabaw and R. Hanna. 2005. Bringing new HOPE to networked games: Using optimistic execution to improve quality of service. In *Proceedings of the DiGRA Conference*.

[75] Vasily Kharitonov. 2012. Motion-aware adaptive dead reckoning algorithm for collaborative virtual environments. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry (VRCAI'12)*.

[76] Joohwan Kim, Pyarelal Knowles, Josef Spjut, Ben Boudaoud, and Morgan Mcguire. 2020. Post-render warp with late input sampling improves aiming under high latency conditions. In *Proceedings of the ACM High Performance Graphics Conference*.

[77] Michael Kopietz. 2019. Systems and methods for player input motion compensation by anticipating motion vectors and/or caching repetitive motion vectors. US Patent 10,341,678 B2.

[78] Yuji Kusunose, Yutaka Ishibashi, Norishige Fukushima, and Shinji Sugawara. 2012. Adaptive delta-causality control with prediction in networked real-time game using haptic media. In *Proceedings of the 18th Asia-Pacific Conference on Communications (APCC'12)*.

[79] Anh Le and Yanni Ellen Liu. 2007. Fairness in multi-player online games on deadline-based networks. In *Proceedings of the 4th IEEE Consumer Communications and Networking Conference (CCNC'07)*. IEEE, Los Alamitos, CA, 670–675. https://doi.org/10.1109/CCNC.2007.137

[80] Huy Viet Le, Valentin Schwind, Philipp Göttlich, and Niels Henze. 2017. PredicTouch: A system to reduce touchscreen latency using neural networks and inertial measurement units. In *Proceedings of the ACM International Conference on Interactive Surfaces and Spaces (ISS'17)*.

[81] LeagueBoards. n.d. League of Legends Forum. Retrieved March 6, 2022 from https://www.reddit.com/r/leagueoflegends/.

[82] Injung Lee, Sunjun Kim, and Byungjoo Lee. 2019. Geometrically compensating effect of end-to-end latency in moving-target selection games. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'19)*. 1–12.

[83] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'15)*. 151–165.

[84] Steven Lee and Rocky Chang. 2017. On 'shot around a corner' in first-person shooter games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'17)*. IEEE, Los Alamitos, CA, 1–6.
[85] Steven W. K. Lee and Rocky K. C. Chang. 2018. Enhancing the experience of multiplayer shooter games via advanced lag compensation. In *Proceedings of the ACM Multimedia Systems Conference (MMSys'18)*. 284–293.
[86] Wai-Kiu Lee and Rocky Chang. 2015. Evaluation of lag-related configurations in first-person shooter games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'15)*. IEEE, Los Alamitos, CA.
[87] Shaolong Li and Changja Chen. 2006. Interest scheme: A new method for path prediction. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'06)*. ACM, New York, NY.
[88] Shaolong Li, Changjia Chen, and Lei Li. 2007. A method of using personal habits for path prediction in network games. In *Proceedings of the 1st International Symposium on Data, Privacy, and E-Commerce (ISDPE'07)*.
[89] Shaolong Li, Changja Chen, and Lei Li. 2008. A new method for path prediction in network games. *Computers in Entertainment* 5, 4 (March 2008), Article 8, 12 pages.
[90] Zhi Li, Hugh Melvin, Rasa Bruzgiene, Peter Pocta, Lea Skorin-Kapov, and Andrej Zgank. 2018. Lag compensation for first-person shooter games in cloud gaming. In *Autonomous Control for a Reliable Internet of Services*. Springer International Publishing, Cham, Switzerland, 104–127.
[91] Dingliang Liang and Paul Boustead. 2006. Using local lag and timewarp to improve performance for real life multiplayer online games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'06)*.
[92] Yow-Jian Lin, Katherine Guo, and Sanjoy Paul. 2002. Sync-MS: Synchronized messaging service for real-time multiplayer distributed games. In *Proceedings of the 10th IEEE International Conference on Network Protocols*. 155–164. https://doi.org/10.1109/ICNP.2002.1181396
[93] Shengmei Liu, Atsuo Kuwahara, James Scovell, Jamie Sherman, and Mark Claypool. 2021. L33t or N00b? How player skill alters the effects of network latency on first person shooter game players. In *Proceedings of the Game Systems Workshop (GameSys'21)*.
[94] Shengmei Liu, Atsuo Kuwahara, Jamie Sherman, James Scovell, and Mark Claypool. 2021. Lower is better? The effects of local latencies on competitive first person shooter game players. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'21)*.
[95] M. Long and C. Gutwin. 2018. Characterizing and modeling the effects of local latency on game performance and experience. In *Proceedings of the ACM Symposium on Computer-Human Interaction in Play (CHI Play'18)*.
[96] I. Scott MacKenzie and Colin Ware. 1993. Lag as a determinant of human performance in interactive systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'93)*. 6.
[97] Regan L. Mandryk and Carl Gutwin. 2008. Perceptibility and utility of sticky targets. In *Proceedings of Graphics Interface 2008 (GI'08)*. 65–72.
[98] Martin Mauve. 2000. Consistency in replicated continuous interactive media. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*.
[99] Martin Mauve. 2000. How to keep a dead man from shooting. In *Proceedings of Interactive Distributed Multimedia Systems and Telecommunication Services Workshop (IDMS'00)*.
[100] Martin Mauve, Jürgen Vogel, Volker Hilt, and Wolfgang Effelsberg. 2004. Local-lag and timewarp: Providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia* 6, 1 (2004), 47–57.
[101] W. McCarty, S. Sheasby, C. Switzer, P. Amburn, and M. R. Stytz. 1994. A virtual cockpit for a distributed interactive simulation. *IEEE Computer Graphics and Applications* 14, 1 (Jan. 1994), 49–54.
[102] Takato Motoo, Jiei Kawasaki, Takuya Fujihashi, Shunsuke Saruwatari, and Takashi Watanabe. 2021. Client-side network delay compensation for online shooting games. *IEEE Access* 9 (Sept. 2021), 125678–125690.
[103] Teemu Maki-Patola and Perttu Hamalainen. 2004. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *Proceedings of the International Computer Music Conference (ICMC'04)*.
[104] A. Ng, M. Annett, P. Dietz, A. Gupta, and W. Bischof. 2014. In the blink of an eye: Investigating latency perception during stylus interaction. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI'14)*.
[105] Doowon Paik, Chung-Ha Yun, and Jooyeon Hwang. 2008. Effective message synchronization methods for multiplayer online games with maps. *Computers in Human Behavior* 24, 6 (Sept. 2008), 2477–2485.
[106] Wladimir Palant, Carsten Griwodz, and Pål Halvorsen. 2006. Evaluating dead reckoning variations with a multiplayer game simulator. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'06)*. ACM, New York, NY, Article 4, 6 pages.
[107] Lothar Pantel and Lars C. Wolf. 2002. On the impact of delay on real-time multiplayer games. In *Proceedings of the Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'02)*.
[108] Lothar Pantel and Lars C. Wolf. 2002. On the suitability of dead reckoning schemes for games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'02)*.

[109] A. Pavlovych and C. Gutwin. 2012. Assessing target acquisition and tracking performance for complex moving targets in the presence of latency and jitter. In *Proceedings of Graphics Interface 2012 (GI'12)*. 109–116.
[110] Ricky Pusch. 2019. Explaining how fighting games use delay-based and rollback netcode. *arsTECHNICA*. Retrieved March 6, 2022 from https://tinyurl.com/y69jt5fz.
[111] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vleeschauwer, and Natalie Degrande. 2004. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'04)*.
[112] Kjetil Raaen and Andreas Petlund. 2015. How much delay is there really in current games? In *Proceedings of the 6th ACM Multimedia Systems Conference (MMSys'15)*. 89–92.
[113] Jared Ramsey. 2020. Online gaming: The rise of a multi-billion dollar industry. *Lineups*. Retrieved March 6, 2022 from https://www.lineups.com/esports/top-10-esports-games/.
[114] Brent Randall. 2020. Valorant's 128-tick servers. *Riot Games*. Retrieved March 6, 2022 from https://technology.riotgames.com/news/valorants-128-tick-servers.
[115] David Roberts, Damien Marshall, Seamus Mcloone, Declan Delaney, Rob Aspin, and Tomas Ward. 2008. Bounding inconsistency using a novel threshold metric for dead reckoning update packet generation. *Simulation* 84, 5 (May 2008), 239–256. https://doi.org/10.1177/0037549708092221
[116] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Carsten Griwodz, and Sebastian Moller. 2018. Towards applying game adaptation to decrease the impact of delay on quality of experience. In *Proceedings of the IEEE International Symposium on Multimedia (ISM'18)*. 114–121.
[117] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Babak Naderi, Carsten Griwodz, and Sebastian Möller. 2020. A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience. In *Proceedings of the ACM Multimedia Systems Conference (MMSys'20)*.
[118] Robert Salay and Mark Claypool. 2020. A comparison of automatic versus manual world alteration for network game latency compensation. In *Proceedings of the ACM Annual Symposium on Computer-Human Interaction in Play (CHI PLAY'20)*.
[119] Cheryl Savery, Nicholas Graham, Carl Gutwin, and Michelle Brown. 2014. The effects of consistency maintenance methods on player experience and performance in networked games. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'14)*.
[120] Cheryl Savery, Nicholas Graham, Carl Gutwin, and Michelle Brown. 2014. The effects of consistency maintenance methods on player experience and performance in networked games. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing*. 1344–1355.
[121] Cheryl Savery and T. C. Graham. 2013. Timelines: Simplifying the programming of lag compensation for the next generation of networked games. *Multimedia Systems* 19, 3 (June 2013), 271–287. https://doi.org/10.1007/s00530-012-0271-3
[122] Cheryl Savery, T. C. Nicholas Graham, and Carl Gutwin. The human factors of consistency maintenance in multiplayer computer games. In *Proceedings of the 16th ACM International Conference on Supporting Group Work*. ACM, New York, NY, 187–196.
[123] Jörg R. J. Schirra. 2001. Content-based reckoning for internet games. In *Proceedings of the Simulation and AI in Games Conference (GAME-ON'01)*.
[124] P. M. Sharkey, M. D. Ryan, and D. J. Roberts. 1998. A local perception filter for distributed virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'98)*.
[125] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. 2003. The effect of latency on user performance in Warcraft III. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'03)*.
[126] Gabriel Shelley and Michael Katchabaw. 2005. Patterns of optimism for reducing the effects of latency in networked multiplayer games. In *Proceedings of FuturePlay*.
[127] H. Shen and Suiping Zhou. 2013. Achieving critical consistency through progressive slowdown in highly interactive multi-player online games. In *Proceedings of the IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD'13)*.
[128] J. Smed, T. Kaukoranta, and H. Hakonen. 2002. Aspects of networking in multiplayer computer games. *Electronic Library* 20, 2 (2002), 87–97.
[129] Jouni Smed, Timo Kaukoranta, and Harri Hakonen. 2002. Aspects of networking in multiplayer computer games. *Electronic Library* 20, 2 (2002), 87–97.
[130] Jouni Smed, Henrik Niinisalo, and Harri Hakonen. 2004. Realizing bullet time effect in multiplayer games with local perception filters. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'04)*. 8.
[131] Richard So and Micheal Griffin. 1992. Compensating lags in head-coupled displays using head position prediction and image deflection. *Journal of Aircraft* 29, 6 (Jan. 1992), 1064–1068. https://eprints.soton.ac.uk/429168/.

[132] Josef Spjut, Ben Boudaoud, Kamran Binaee, Jonghyun Kim, Alexander Majercik, Morgan McGuire, David Luebke, and Joohwan Kim. 2019. Latency of 30 ms benefits first person targeting tasks more than refresh rate above 60 Hz. In *Proceedings of SIGGRAPH Asia*. 110–113.

[133] Michael Stallone. 2019. 8 Frames in 16ms: Rollback networking in Mortal Kombat and Injustice 2. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=7jb0FOcImdg&feature=youtu.be&ab_channel=GDC.

[134] Steam Forum. n.d. Rocket League. Retrieved March 6, 2022 from https://steamcommunity.com/app/252950/discussions/.

[135] David Straily and Dave Heironymus. 2020. Netcode and 128-servers. *YouTube*. Retrieved March 6, 2022 from https://www.youtube.com/watch?v=_Cu97mr7zcM.

[136] Dane Stuckel and Carl Gutwin. 2008. The effects of local lag on tightly-coupled interaction in distributed groupware. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'08)*.

[137] Jiawei Sun and Mark Claypool. 2019. Evaluating streaming and latency compensation in a cloud-based game. In *Proceedings of the 15th IARIA Advanced International Conference on Telecommunications (AICT'19)*.

[138] Neema Teymory. 2016. Why making multiplayer games is hard: Lag compensating weapons in MechWarrior Online. *Gamasutra*. Retrieved March 6, 2022 from https://tinyurl.com/gamasutra-teymory-2016.

[139] Alexey Tumanov, Robert Allison, and Wolfgang Stuerzlinger. 2007. Variability-aware latency amelioration in distributed environments. In *Proceedings of the IEEE Virtual Reality Conference*. 123–130. https://doi.org/10.1109/VR.2007.352472

[140] Rosane Ushirobira, Denis Efimov, Géry Casiez, Nicolas Roussel, and Wilfrid Perruquetti. 2016. A forecasting algorithm for latency compensation in indirect human-computer interactions. In *Proceedings of the European Control Conference (ECC'16)*. IEEE, Los Alamitos, CA, 1081–1086.

[141] Ivan Vaghi, Chris Greenhalgh, and Steve Benford. 1999. Coping with inconsistency due to network delays in collaborative virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'99)*.

[142] Rodrigo Vicencio-Moreira, Regan L. Mandryk, Carl Gutwin, and Scott Bateman. 2014. The effectiveness (or lack thereof) of aim-assist techniques in first-person shooter games. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'14)*.

[143] Wikipedia Contributors. 2021. Dead reckoning. *Wikipedia*. Retrieved December 28, 2021 from https://en.wikipedia.org/wiki/Dead_reckoning.

[144] Greger Wikstrand, Lennart Schedin, and Fredrik Elg. 2004. High and low ping and the game of pong effects of delay and feedback. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'04)*.

[145] Jiann-Rong Wu and Ming Ouhyoung. 2000. On latency compensation and its effects for head motion trajectories in virtual environments. *Visual Computer* 16 (March 2000), 79–90.

[146] Jingxi Xu and B. Wah. 2013. Concealing network delays in delay-sensitive online interactive games based on just-noticeable differences. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME'13)*. 1–6.

[147] Amir Yahyavi, Kevin Huguenin, and Bettina Kemme. 2012. Interest modeling in games: The case of dead reckoning. *Multimedia Systems* 19 (June 2012), 255–270. https://doi.org/10.1007/s00530-012-0275-z

[148] Seok-Jong Yu and Yoon-Chul Choy. 2001. A dynamic message filtering technique for 3D cyberspaces. *IEEE Computer Communications* 24, 18 (2001), 1745–1758.

[149] Sebastian Zander, Ian Leeder, and Grenville Armitage. 2005. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'05)*.

[150] Mao-Jun Zhang and Nicolas Georganas. 2004. An orientation update message filtering algorithm in collaborative virtual environments. *Journal of Computer Science and Technology* 19, 3 (2004), 423–429.

[151] Yi Zhang, Ling Chen, and Gencai Chen. 2006. Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games. In *Proceedings of the ACM Workshop on Network and System Support for Games (NetGames'06)*.

[152] Sili Zhao, Du Li, Hansu Gu, Bin Shao, and Ning Gu. 2009. An approach to sharing legacy TV/arcade games for real-time collaboration. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS'09)*. IEEE, Los Alamitos, CA, 165–172.

Received August 2021; revised January 2022; accepted February 2022