

# NETCODE IN MODERN FIGHTING GAMES



Vicente Francisco Cupello

[verdalg@gmail.com](mailto:verdalg@gmail.com)

UNIVERSIDAD FRANCISCO DE VITORIA

FACULTAD DE CIENCIAS DE LA COMUNICACIÓN

GRADO DE CREACIÓN Y NARRACIÓN DE VIDEOJUEGOS

2022 - 2023

Tutor: Gabriel Peñas Rodríguez

[gabriel.penas@ufv.es](mailto:gabriel.penas@ufv.es)

## **Resumen**

A lo largo de este trabajo, revisaremos los fundamentos de sistemas de redes y cómo funciona Internet. Analizaremos cómo los juegos comunican datos entre los ordenadores y cómo ha cambiado en este aspecto a lo largo de la historia de los videojuegos, particularmente en los juegos de lucha. Además, recopilaremos una breve historia de los juegos de lucha, y su diseño en relación con los avances en los sistemas de redes.

Por último, propondremos un ejemplo básico sobre cómo se podría construir diferentes tipos de códigos de red y cómo funcionan, lo cual nos permitirá ver las ventajas y desventajas de los diferentes métodos. Sobre todo, nos enfocaremos en los códigos de red más significativos de hoy en día: “Rollback” y “Delay-Based” y su evolución en la industria de los juegos de lucha.

*Palabras Clave:* Código de Red, Juegos de Lucha, Rollback, Multijugador, Sistemas de Red

# NETCODE IN MODERN FIGHTING GAMES

## **Abstract**

Throughout this paper, we will look at the basics of networking and how the internet functions, as well as how to synchronize a game between two computers, and how this has changed over the history of the video games industry, specifically in the genre of fighting games. We will also look at a brief history of fighting games and how they work design wise to understand the importance of networking advancements.

Finally, we will take a theoretical but basic approach for how to build different types of netcode and how these netcodes work in-game, as well as the advantages and disadvantages of each type of netcode. We will discuss how different netcodes, such as “Rollback” and “Delay-Based” netcodes have evolved, as well as their significance in the fighting game industry today.

*Keywords:* Netcode, Fighting Games, Rollback, Multiplayer, Network Systems

## Index

<b>1. Introduction</b>	<b>1</b>
1.1 Hypothesis	2
1.2 Objectives	2
1.3 Methodology	2
<b>2. Background</b>	<b>3</b>
2.1 Networking Systems for Online Implementation	3
2.1.1 The OSI Model According to Craig Hunt	4
2.2 TCP vs UDP	6
2.3 History of Fighting Games	9
2.3.1 Balance and Lag Tactics	13
2.4 The Game Loop	15
2.5 A History of Poor Netcode	17
2.6 Game Networking Concepts	17
2.6.1 Sending State or Input	17
2.6.2 Lockstep	18
2.7 Delay-based Netcode	18
2.8 Rollback Netcode	21
2.8.1 Deterministic Simulation	24
2.8.2 Independent Simulation of the Game	26
2.8.3 Serialization at Any Moment	27
2.9 Blended Delayed Rollback Netcode	28
2.10 Case Studies of Popular Fighting Games	29
2.10.1 Brand Value	30
2.10.2 Netcode Differences	31
<b>3. Development</b>	<b>34</b>
3.1 Dolphin Versus Slippi	33
3.2 Rollback Example	36
<b>4. Results of the Experiment</b>	<b>40</b>
4.1 Quantitative Analysis	41
4.2 Statistical Analysis	47
<b>5. Conclusions and Limitations</b>	<b>49</b>
<b>Bibliography</b>	<b>51</b>
<b>Ludography</b>	<b>55</b>
<b>Glossary</b>	<b>57</b>
<b>Appendix</b>	<b>58</b>

## 1. Introduction

In the past few decades, video games exploded in popularity, slowly becoming accepted into our society as a mainstream form of entertainment. The video games industry has undergone exponential growth, and with it, the amount of time and money people invest in them. With this, the technology and systems in video games have also evolved (Howarth, 2022).

The idea of multiplayer games together has always existed, as shown in early games such as *Tennis for Two* (Higinbotham & Dvorak, 1958) and *Spacewar!* (Russell et al., 1962). However, with Internet access becoming more widespread a few decades later, multiplayer games began being developed with the intention of being able to play them across a network, meaning two or more simulations of the same game would be connected to each other. As a result, programmers had to develop code that ensured synchronization between the instances of the games (Glazer & Sanjay, 2015).

One of the main issues that programmers had to consider was latency, also known as the amount of time it takes for data to be transferred over a network. Latency adds a delay between a player's inputs and feedback from the game, meaning a higher latency generally leads to a worse game experience, and in more severe cases, players leaving games. Over time, programmers (Chen et al., 2006).

Fighting games, such as *Street Fighter*, are generally played by 2 players in which players compete to reduce their opponent's health bar down to 0 to claim victory. By nature, fighting games are competitive and they generally require a high level of precision and strategy for a player to succeed, leading them to be closely connected with esports (a term for competitive video games, akin to real sports) (Ehlert, 2021).

With this connection to esports comes a necessity for a reliable and efficient practice environment, meaning online play had to be optimized to fulfill this need, especially in recent years with the COVID-19 pandemic putting a stop to offline in-person tournaments (Cohu, 2021). The aim was to create an online game experience where the inherent lag was reduced as much as possible, attempting to mimic the experience of offline play as much as possible in that regard (Cannon, 2012).

Over time, programmers have developed ingenious solutions that synchronize the game as well as reduce the effects of lag, namely Delay-Based networking and rollback networking, which are the two main networking methods that we will investigate throughout this paper (Pusch, 2019). We will also take a practical approach by exploring an example of rollback pseudocode. Finally, we will statistically analyze player satisfaction in relation to netcode to gauge its importance in concurrent and future development of online multiplayer games.

### **1.1 Hypothesis**

Implementing rollback netcode in fighting games leads to a more positive player experience by:

- Increasing responsiveness: Players should feel capable of reacting to their opponent's actions.
- Providing more stable gameplay: Latency in networks often fluctuates which can be frustrating to players, rollback netcode attempts to inhibit the effect of latency fluctuation on gameplay.

### **1.2 Objectives**

- Raise awareness about different types and the importance of netcode for fighting game developers.
- Give an understanding of rollback netcode at a basic level for people playing fighting games; If players understand how rollback netcode works, they can ensure that all technical requirements are met prior to playing the game, preventing a poor online experience.

### **1.3 Methodology**

1. Gain a background understanding of networking functions and how information is sent/received.
2. Analyze the history of fighting games, release dates, different franchises, and how the game loop of a fighting game works.
3. Investigate the development of netcode in video games and its evolution over time in order to compare the different types of netcode both in performance and in practicality.
4. Develop and present pseudocode that would aid in understanding how rollback netcode works as well as outlining the steps a developer might take to implement it.
5. Conduct an experiment where test subjects play Super Smash Bros Melee on Delay-Based netcode (on Dolphin emulator) and rollback netcode (on Slippi) under realistic network

conditions (connecting players from Spain to players in Italy and the UK). This is followed up by a questionnaire in which the test subjects will answer questions about their player experience with the different netcodes. This data will be analyzed to evaluate the netcodes' effect on the player experience.

## 2. Background

### 2.1 Networking Systems for Online Implementation

The reference 'Open Systems Interconnection' or OSI model, helps us understand the basics of data networking and is still referenced a lot to this day, simply because it is a primitive version of how systems communicate with each other. The OSI model itself encapsulates the networking process in seven different layers.

In Figure 1, the OSI model is displayed visually. The layers display the data flow as moving downwards, because when a user wants to communicate with another system, this is the path the data must take in order to be processed and transmitted correctly. Keep in mind that each layer defines a data function for communication, not a singular protocol. As an example, a file sharing protocol and file transfer protocol might share the same layer (later defined as the Application layer), despite being different protocols, due to them providing user services which both the user and the computer can interact with (Hunt, 2002).

Layers generally have specific duties, such as (Meyer & Zobrist 1990):

- Accepting data from a higher layer to transmit
- Packaging data up with a header and in some cases a footer (they have relevant information for the source/destination of the data such as IP addresses or MAC addresses)
- Passing data to a lower level for further transmission
- Receiving data from a lower level and unpackaging it by removing the header
- Forwarding transmitted data to a higher layer for further processing

#### 2.1.1 The OSI Model According to Craig Hunt

We can outline the functionalities of each layer of the OSI model (refer to Figure 1), as defined by Hunt in his book: *TCP/IP Network Administration*, written in 2002.

**Layer 1: Physical Layer**

The physical layer defines the physical details of the data connection, such as the hardware needed for the data transmission signal, such as cables, connectors, or network interface cards. Other characteristics such as voltage, type of cable (e.g., a fiber optical cable), or network adapters are defined in this layer. The data in this layer is sent as a raw bit stream of 0's and 1's.

**Layer 2: Data Link Layer**

This layer adds the physical addressing (MAC Address) and defines the format of the data on the network. As mentioned, the data from the physical layer may have errors, thus the data link layer will break the raw bit stream into frames or fragments. Then, for each frame a checksum is computed and appended so that way both the source and the destination devices can ensure that the data is sent in its entirety without errors. Finally, it also controls how fast data is transmitted between the source and the destination to prevent dropping of packets at the receiving end.

**Layer 3: Network Layer**

The network layer deals with organizing the data and preparing it for transfer and reassembly. The data's path is determined along with its addressing (e.g., the IP of the source and destination). Essentially, the layer provides addresses to the data which aids them in finding their path.

**Layer 4: Transport Layer**

The transport layer encompasses the protocols that occur in the transportation of data between devices. In the OSI model, the transport layer makes sure that the data is received exactly as it was sent, however in the TCP/IP model, there is a different transport layer service, known as *User Datagram Protocol* or UDP. The main difference is that UDP does not perform reliability checks, meaning data packets may be lost.

**Layer 5: Session Layer**

The session layer manages the connections between the source and destination applications. In the OSI model, it is used to establish the connection, however the TCP/IP model performs this functionality within its transport layer and omits the term "session".



### Layer 6: Presentation Layer

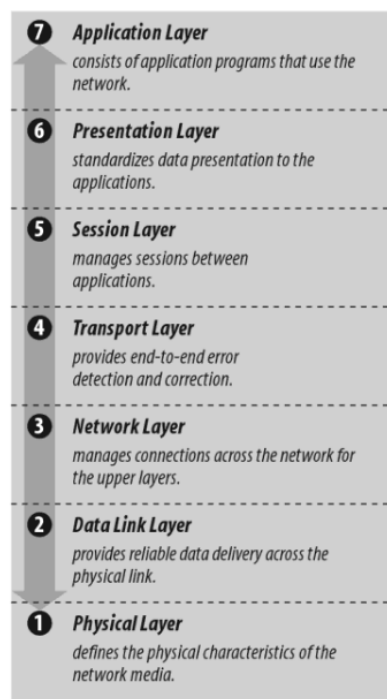
Since both the source and destination applications must cooperate to transfer data, they must both have a standard to follow when representing data. In the OSI model, the presentation layer is where this standard is handled in its functions, such as encryption. In the TCP/IP model, this layer's functionality is handled in the final layer also known as the application layer.

### Layer 7: Application Layer

The application layer is where the user connects with the application, with the ability to access specific network processes, such as sending an email or purchasing an item. It includes processes which the users can directly interact with as well as some processes which the users are not aware of.

### Figure 1

*The OSI Model (Hunt, 2002)*



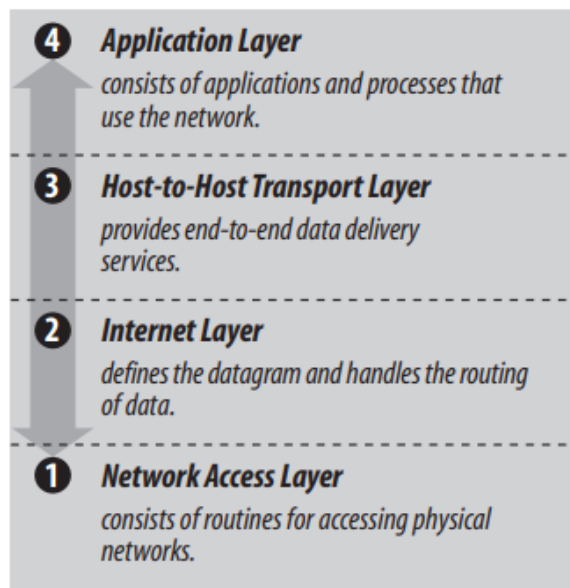
Generally, the flow of data tends to move from layer 7 to layer 1 in the source's device, which then is transferred over the network, into the receiver's layer 1, which then is brought up to layer 7 (Meyer & Zobrist 1990). As previously mentioned, the OSI model functions as a reference to understand how computers may communicate data over the internet, but currently the TCP/IP

model is used more frequently in video games, due to a more protocol-based approach, which is more practical within the development of applications (Hunt, 2002).

In game development, normally the TCP/IP model is used, thus our focus will be on understanding how the TCP/IP model works. Figure 2 represents the TCP/IP model, which is reduced to four layers, as opposed to the OSI model. In this model the ‘Application Layer’ encapsulates the ‘Presentation’ and ‘Session’ layers, performing all the functionality and protocols that those layers would be in charge of. Likewise, the network access layer encapsulates the physical and data link layers (Hunt, 2002).

**Figure 2**

*The TCP/IP model (Hunt, 2002)*



## 2.2 TCP vs UDP

We can investigate the common topic of when to use TCP (transmission control protocol) or UDP (user datagram protocol) so that we understand the importance of the OSI model and its layers. All networked multiplayer games have some level of need for network reliability, to make sure packets of data are sent and received appropriately. TCP is generally more reliable as it guarantees that all data will be delivered properly and in order, as well as limiting data flow speed so that it does not overwhelm intermediate routers between the source and destination (Glazer & Sanjay, 2015). Despite this, TCP has a drawback in games where game state is rapidly changing

because its packets must be sent reliably and processed in order. We can discuss two different situations (in real games) where this might be an issue:

1. Loss of low priority data interfering with high priority data.

In *League of Legends* or any other MOBA game, a movement packet is generally considered to be of high priority, while a chat packet might be low priority. If Client A receives a chat message and shortly after Client B uses an ability, the fluctuating network traffic might cause a packet to be dropped. If the low priority chat message packet were dropped, it would stop the flow of what is happening on Client A's screen until the chat message is received, and they may be too late to react to the high priority ability packet, since their system would have to wait until the low priority packet is fully received and processed before processing the high priority packet. Client A wouldn't be too annoyed at the chat message packet being dropped, but they would understandably be mad at the ability package being dropped (Glazer & Sanjay, 2015).

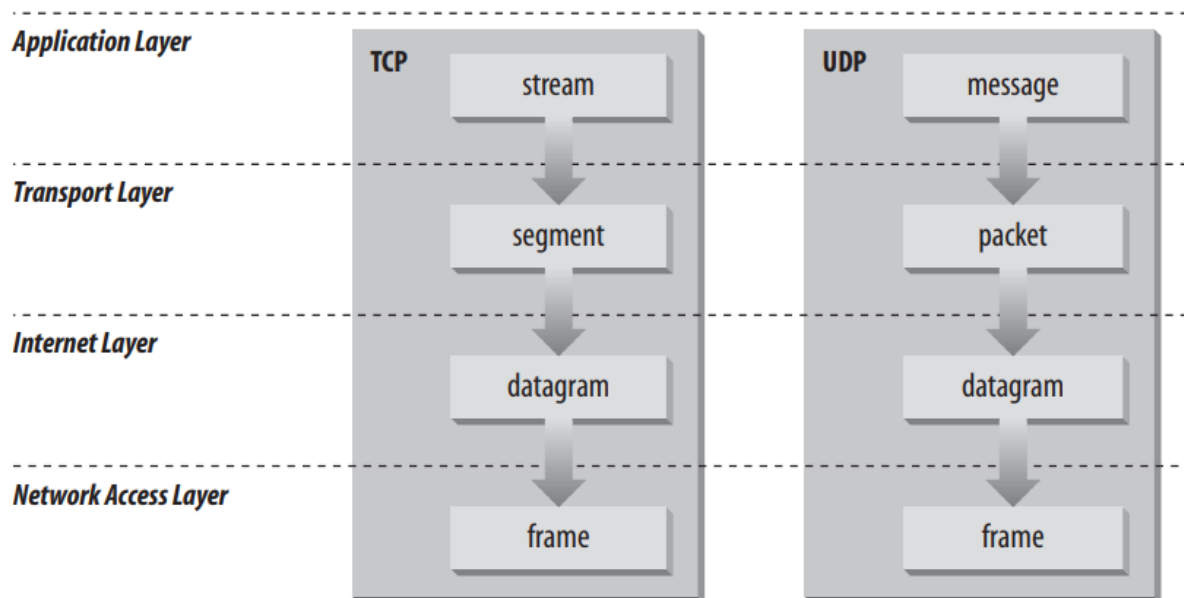
2. Retransmission of stale game state.

In a traditional 2D fighting game, such as *Guilty Gear*, players may move horizontally or jump, apart from other attacks and movement techniques. Imagine player B runs at player A over the course of 2 seconds and attempts to attack them, going from position  $x = 0$  to  $x = 100$ . In TCP, the packets are sent at a fixed rate, so if any of the packets within those 2 seconds are dropped, the server will resend them, meaning while player B is approaching player A, player A might be receiving packets which have player B closer to their initial position at  $x = 0$ , which is not acceptable for player A (Glazer & Sanjay, 2015).

UDP on the other hand does not have the built-in reliability that TCP provides, meaning it does not ensure all packets arrive at their destination. Nevertheless, it allows us to implement a custom-made system that might allow for both reliable and unreliable data, or a system that might send only the newest information when replacing dropped packets, which would fix problem #2 mentioned above. Memory can be managed in a specific manner that allows for specific behavior in dealing with specific game behavior. Obviously, this comes at the cost of higher-level engineering and development and testing time. Additionally, UDP comes with an increased risk as some routers can be configured to treat UDP packets as lower priority (Glazer & Sanjay, 2015).

**Figure 3**

*TCP vs UDP data structures (Hunt, 2002).*



Generally, if data is needed to be reliable and processed in a specific order such as in a card game, like *Magic the Gathering Arena* (Wizards of the Coast, 2018), or *Legends of Runeterra* (Riot Games, 2020), then TCP is recommended. This is often the case for turn based games, where data does not need to be sent and received at a high rate (Glazer & Sanjay, 2015). For most games however, a lot of data is sent and received in small increments of time, meaning UDP is necessary so that the game’s flow is not stopped due to packet loss, with a custom implementation of the data reliability system at the ‘Application Layer’, which is where the main topic of this paper takes place: Netcode (Glazer & Sanjay, 2015).

We can define *netcode* as the code at the ‘Application Layer’ that “refers to the technical implementation of real-time networking and state synchronization in online video games” (Ehlert, 2021). The game’s netcode determines the game’s behavior when it encounters a situation such as the ones mentioned previously, where a packet might be lost, and arrive late due to the packet loss which comes with using UDP (Glazer & Sanjay, 2015). The two main models for netcode that are used in multiplayer games today are known as Delay-Based netcode and rollback netcode (Huynh & Valariano, 2019).

Prior to discussing netcode, we must first gain a background understanding of fighting games and how the game loop works in them. The game loop occurs in frames, also known as the smallest unit of time in fighting games. In this time, the game samples for player input, processes the inputs to update the game state, and then renders the result of the game state onto the screen (Ehlert, 2021). Most fighting games run at 60 frames per second, therefore the entire process of the game loop must be carried out within 16.66 milliseconds (Cannon, 2012).

### 2.3 History of Fighting Games

Fighting games are a video game genre in which two or more players engage in combat, with the goal of bringing the opponent's health down to 0. Normally, these games occur in what look like 3D arenas, but are actually 2D spaces with a fixed size, which accomplishes the goal of forcing interaction between players by restricting movement (Ehlert, 2021). Games such as *Street Fighter V* (Capcom, 2016), *Guilty Gear Strive* (ArcSystemWorks, 2021), and *Dragon Ball FighterZ* (ArcSystemWorks, 2017) all use this model for their gameplay, as seen in Figure 4 (Capcom, 2016).

**Figure 4**

*Traditional fighting game example - Dragon Ball FighterZ (ArcSystemWorks, 2017).*



However, there are some games that break some of these rules in unconventional ways. For example, *Tekken 7* (Bandai Namco Entertainment, 2015) and *Soul Calibur VI* (Bandai Namco Entertainment, 2018) allow the player to move their character in a limited 3D space, as seen in Figure 5. The 3D space players can move in often acts as a ring, much akin to martial arts sports, such as taekwondo, boxing, and wrestling.

**Figure 5**

*3D Fighting Game - Tekken 7 (Bandai Namco Entertainment, 2015).*



Lastly, there is also the platform fighter archetype, such as the *Super Smash Bros.* (Nintendo, 1999) franchise or *Rivals of Aether* (Fornace, 2017), where the objective of the player is to push their opponent off-screen, instead of lowering the opponent's HP (or health points) to 0, and features much more freedom of movement around the stage as seen in Figure 6.



**Figure 6**

Example of platform fighter: *Super Smash Bros. Melee* (Nintendo, 2001). Image taken from *Slippi.gg*'s replay visualizer.



The origin of fighting games can be attributed to martial arts films, especially those that featured Bruce Lee, such as *Game of Death* and *Enter the Dragon*, which held foundational concepts of fighting games (Patrick, 2020). The game *Kung Fu Master* was based on *Game of Death*, and it followed the same storyline of Bruce Lee battling up a tower with increasingly more difficult opponents (Patrick, 2020). This later became the basis of *Street Fighter* (Capcom, 1987), which is one of the most popular fighting game titles of all time (Patrick, 2020).

Later, the release of *Street Fighter II* (Capcom, 1991) revolutionized the fighting game genre by resolving joystick and button scanning routines by implementing an input buffer system that ignores irrelevant inputs. This also permits inputs slightly earlier than when the character can execute them, saving the input for a fixed amount of time (varies from game to game), and allowing for its execution if the character is actionable during the buffer window (Marino, 2021). This made multi-button special moves (refer to Figure 7), known for requiring motion inputs, more accurate, despite the high-level of execution they required previously (Ehlert, 2021).

**Figure 7**

*Special Moves in Dragon Ball FighterZ (ArcSystemWorks, 2017).*



Another massive change to the industry was the fact that the game was now PvP (or player versus player), instead of the player combatting computer-controlled fighters (Marino, 2021). Fighting game concepts such as ‘true combos’ started to appear, which is when a player combines attacks in quick succession that prevents the opponent from responding, if timed correctly, facilitated by the hardware advancements (Marino, 2021). This success led to arcades adopting fighting games as a mainstream genre, which in turn, led to the rebound of arcades in the early 1990s (Carter, 1993).

The nature of fighting games being competitive led to arcades adopting a competitive culture in which fighting game players would line up to play at an arcade machine, and players would agree upon a ruleset which was based on what we know today as “King of the Hill”, winner stays, loser goes away. Competition was the main driving factor for players to spend hours at the arcade, and winning was everything (Carter, 1993).

Major fighting game tournaments such as Tougeki - Super Battle Opera and Evolution Championship Series began to rise and fighting games as an eSport became mainstream, with



massive viewership, and thus, fairness and balance started becoming a concern, as is normal in competition (Gifford, 2010).

### 2.3.1 Balance and Lag Tactics

In *Street Fighter II*, attacks apply a random amount of stun, which quickly became an issue, and its successors changed the system and applied a fixed amount of stun based on the attack used in order for outcomes of matches to be more skill-based (Capcom, 1991). Online play also changes the dynamic of fighting games. Super Smash Bros. Melee presents a game-changing example known as *reaction tech chasing* (or RTC). In Figure 8, Captain Falcon has thrown Fox, leaving Fox knocked down. After being knocked down, Fox must recover in one of 4 ways:

- Normal Getup (Standing up)
- Roll Left
- Roll Right
- Getup Attack (Attack while standing up, leaves you vulnerable if the attack is blocked/dodged)

#### Figure 8

*Captain Falcon crouch tech chasing Fox after a down-throw in Super Smash Bros. Melee (Nintendo, 2001).*

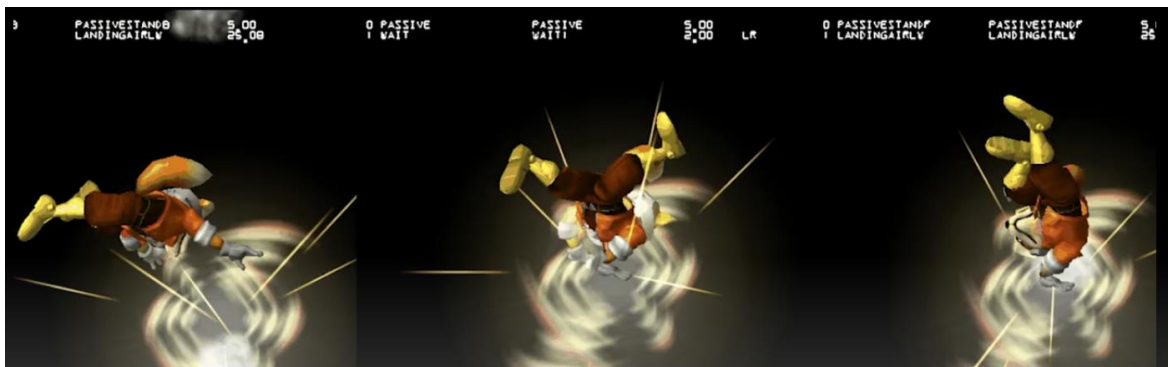


This presents the Captain Falcon player with an advantageous position in which they get to act before Fox and potentially extend their lead by landing a further hit after Fox chooses a recovery option. However, reaction tech chasing turns this situation from a potential combo into a guaranteed one. In Figure 9, we can see Fox performing one of the available recovery options, and they all have different animations: Roll left, Normal Getup, and Roll Right. These distinctions in the animations happen early enough where if the player looks for them, they have enough time to react to the recovery and land a hit in a guaranteed manner. Humans react to visual stimuli at roughly around 250ms (milliseconds) (Jain, 2015).

Tech recoveries last for 26 frames in Melee, with 6 frames of vulnerability. This means that humans have roughly 430ms -  $(26 \times 16.66)$  to react and land an attack at the Fox's recovery location, and doing so already takes some time depending on the movement speed of the character and the attack's speed, so the player is left with a small margin (roughly 300ms) to continue their combo.

### Figure 9

*Fox performing a left tech roll, neutral tech and right tech roll in Super Smash Bros. Melee (Nintendo, 2001).*



Reaction tech chasing is commonly used at top level play in offline tournaments, but online gameplay is different. With more input delay added, reacting to stimuli becomes harder as the margin for error becomes smaller, meaning latency can also affect how people play the game. Tony Cannon refers to this as “lag tactics”, essentially, changing your strategy with latency in mind such as using attacks that might be hard to react to, essentially rendering them impossible to react to in online environments. As a result, ‘optimal’ play in the same game differs due to network conditions, which is unacceptable (Cannon, 2012).

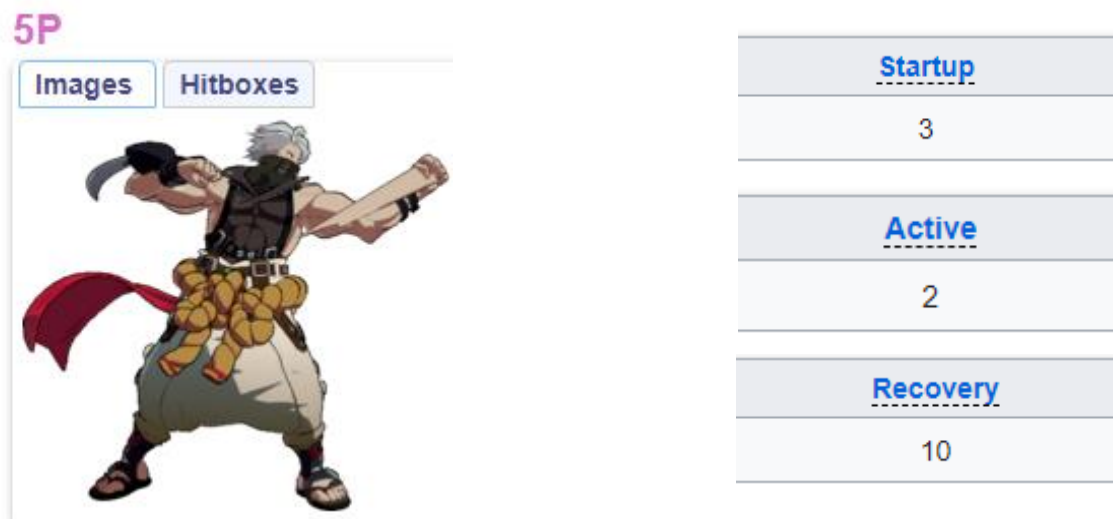
With the rise in popularity in online multiplayer games and competition at the esports level, development of netcode for fighting games began being explored, with a recent resurgence due to the COVID-19 pandemic putting a halt to offline tournaments (Cohu, 2021). However, prior to understanding rollback netcode, we must understand how the game loop in a fighting game works.

## 2.4 The Game Loop

In fighting games, the timing method of the game loop is directly tied to a lot of aspects of game logic, such as a character's move duration. This also means that animations are tied to frames, such as the number of hit-stun frames, or block stun frames, active frames, or even recovery frames (Refer to Figure 11) (Huynh & Valariano, 2019). As an example, Figure 10 displays Chipp Zanuff's frame data from *Guilty Gear Strive*. Chipp Zanuff's punch lasts for 15 frames, but it is split up into 3 startup frames, 2 active frames, and 10 recovery frames.

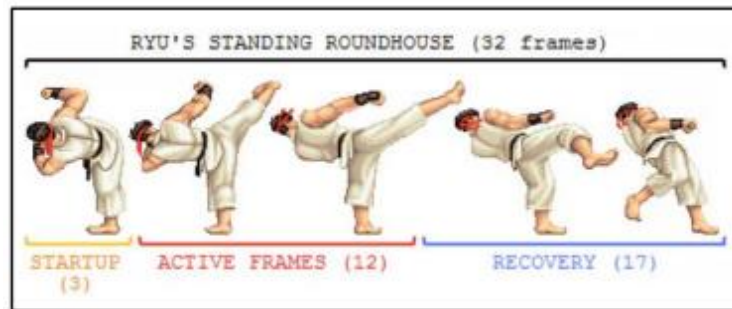
**Figure 10**

*Chipp Zanuff's punch in Guilty Gear Strive with frame data. (ArcSystemWorks, 2021).*



**Figure 11**

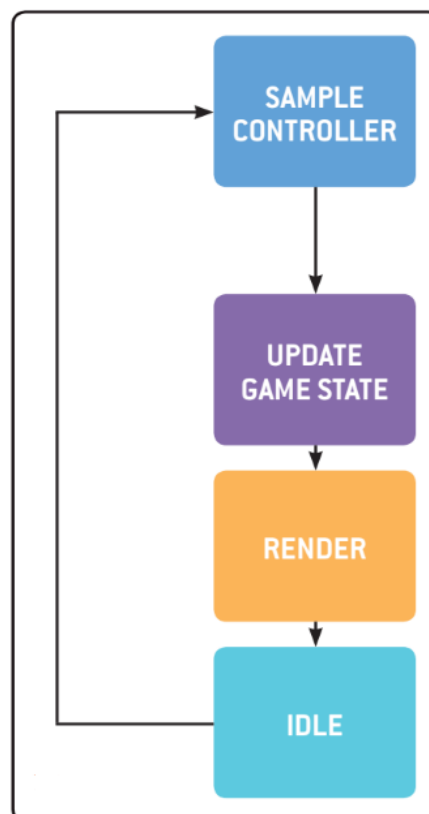
*Frame diagram for Ryu's Standing Roundhouse Kick (Huynh & Valariano, 2019)*



Fighting games normally require a system that is able to handle the game loop consistently at 60 frames per second without fail. In offline multiplayer fighting games, the game loop is quite simple, since the game loop is generally more consistent, inputs are sampled, processed and then rendered as shown in Figure 12 (Cannon, 2012).

**Figure 12**

*The typical arcade game loop (Cannon, 2012)*



However, networking solutions cause the game loop to be more complex since multiple instances of the game have to be running in parallel but synchronized so that both players have the same information about what is happening in the game. If we ignore synchronization of each of the players' games, then network latency differences cause desyncs, where each game, despite being connected, has a different state, rendering the game unplayable (Huynh & Valariano, 2019).

## 2.5 A History of Poor Netcode

Fighting game netcode in the early days did not accomplish the goal of creating a solid practice environment (Sasaki, 2006). *Tony Cannon*, professional fighting game player and tournament organizer described the online experience as “literally unplayable” (Orland, 2011). For the longest time, players overlooked this issue because the highest level of competition was always offline (Huynh & Valariano, 2019). As mentioned before, when the COVID-19 pandemic began, online tournaments became the norm, along with online gaming in general increasing in popularity (Howarth, 2022). This meant that within the fighting game community (or FGC), games that had better online functionality had a better chance at success.

## 2.6 Game Networking Concepts

### 2.6.1 Sending State or Input

One of the most important choices one has to make from the beginning is choosing what information your game will send and receive in the network. As long as both players have the **same game state**, the game is synchronized which is our goal when making multiplayer games.

We can send game state as such (Sun, 2019):

1. Player 1 sends `Jump()` to the server or authoritative node.
2. The server receives the request and processes the command. After calculating the game state, it relays Player 1's `Jump()` to the other players.
3. Player 1 receives the response from the server and finally performs `Jump()` locally. Player 1's character performs `Jump()` on all other players' screens as well, since the server relayed the method call.

In this example, our server or authoritative node acts as a central point for our network. There are cases where a player can also run the server's functionality, often referred to as 'host', since it is performing both the server and client's jobs (Huynh & Valariano, 2019).

Now when we send input, it might look like this:

1. Player 1 inputs the 'A' button on their controller.
2. Player 1 sends the 'A' input to the server and their inputs are relayed to the other players.
3. Player 1's character performs the action bound to the 'A' button.

As opposed to the example above, we are now sending inputs rather than game state. In the first example, the authoritative node or server performs the calculations necessary to process the frame, but in the second example, each one of the player's simulations simply receives inputs and simulates the frames on their own.

However, a problem that arises is when the game logic has some sort of imprecision in how it determines game state. The same input may differ on the various simulations of the game, causing a desync (Huynh & Valariano, 2019). To solve this, we must make sure our game logic is 'deterministic', which is further discussed in the 'rollback netcode' section below (Huynh & Valariano, 2019).

### **2.6.2 Lockstep**

One of the first solutions that might come to mind might be to pause the game until all the info between the connected users is acquired, in this case being the inputs. This is usually referred to as 'Lockstep' (Huynh & Valariano, 2019). This normally works for turn based games, where a turn only progresses once the player has thought about their move and decided upon it. If there is a discrepancy in latency in a turn-based game, players won't notice any delay, since turns are expected to last for a while.

However, fighting games commonly run at 60 frames per second, meaning "turns" are supposed to last 16ms (or 1/60th of a frame), so when inputs are pressed, the game confirms that both simulations have received inputs, effectively adding input delay to gameplay (Huynh & Valariano, 2019). A different implementation of lockstep was used in major game franchises by NetherRealm Studios such as *Mortal Kombat* and *Injustice* and ended up being unsatisfactory, known as Delay-Based netcode (Stallone, 2019).

### **2.7 Delay-Based Netcode**

We can attempt to solve the problem found in a lockstep simulation by adding an artificial latency, which applies a delay to each of the players' inputs locally. The delay would provide the

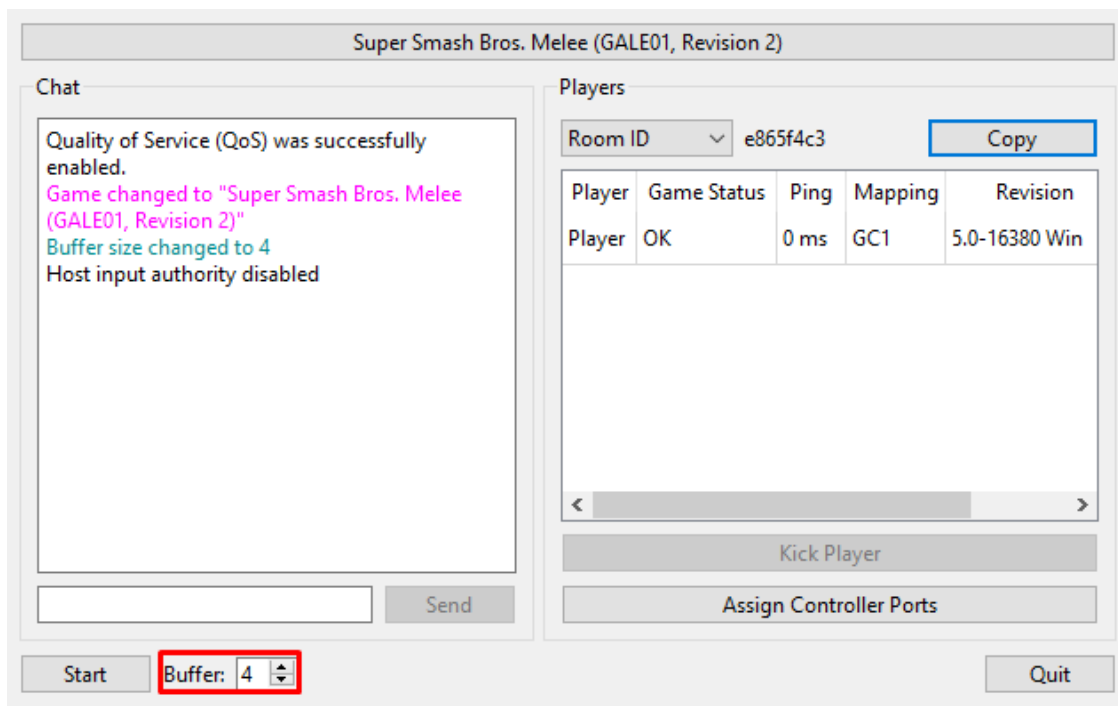
time necessary for the input packets to arrive to each player. As an example, if player 1 inputs an attack on frame 1, that attack will begin on frame 1 normally. With the 6 frames of input latency applied, that attack would be delayed for 6 more frames, meaning the attack would start on frame 7. The game uses the latency frames (frames 1-6) as a margin for the remote player's inputs to be transmitted (Huynh & Valariano, 2019).

Dolphin is an emulator for the Nintendo GameCube and Wii consoles that uses this input latency for online play. In Figure 13, highlighted in a red box, is the input buffer. This indicates the amount of input latency in frames. For every frame of buffer added, the game adds an additional 16 milliseconds of input latency.

Ideally, the players should set up the buffer in accordance with their maximum expected ping, but if the ping never reaches that point, then we are adding an unnecessary delay. If ping exceeds the expected amount, then the game may need to pause or slow down (essentially lag), since it is not prepared for that fluctuation in network latency, which makes the experience undesirable.

**Figure 13**

*Input buffer setting in Dolphin Emulator Netplay window.*



We cannot assume network latency will be constant between the two players, which is where variable input latency may resolve our problem. By making our added artificial latency dynamic, the game can detect fluctuations in the connection quality and adjust the input latency accordingly to guarantee inputs are sent and received and optimize the player experience. When the system determines the connection is poor, input latency is increased, and vice-versa.

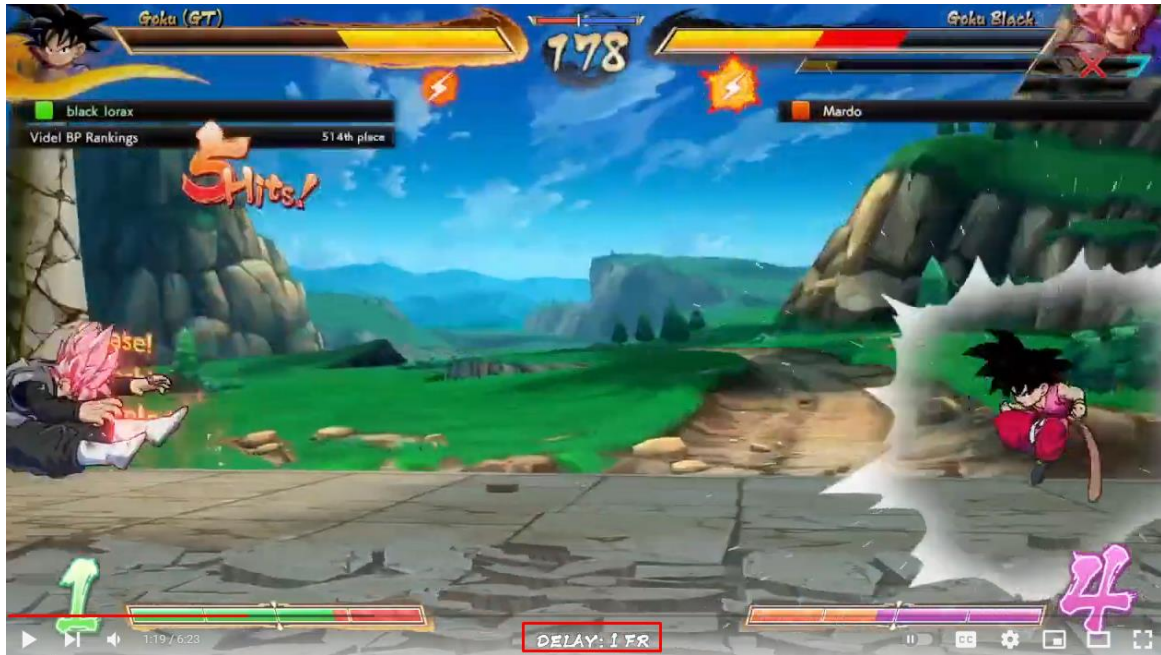
However, fighting games need to be responsive and consistent, as they tend to have strict timings for inputs, usually no more than 6 frames, which is about 1/10th of a second. Sometimes the window can go as small as a singular frame (0.016s approximately) (Cannon, 2012). This netcode solution alleviates responsiveness by a small margin, but at a loss in consistency, which explains why NetherRealms implementation of this method was unsuccessful (Huynh & Valariano, 2019).

In Figures 13 and 14, the input latency is displayed at the bottom of the screen. The players' connection quality fluctuated slightly from frame to frame, and the game recognized this discrepancy and added a frame of input delay to compensate. This variable input latency is small in this case, but larger fluctuations such as 5-10 frames are possible, and this completely changes the game experience. If a player was chaining attacks in a combo, and halfway through the combo, input latency changes can cause the player to lose the combo, or it cause the defending player to get hit again. It adds variance to the outcome which cannot be controlled by the players' actions in-game.

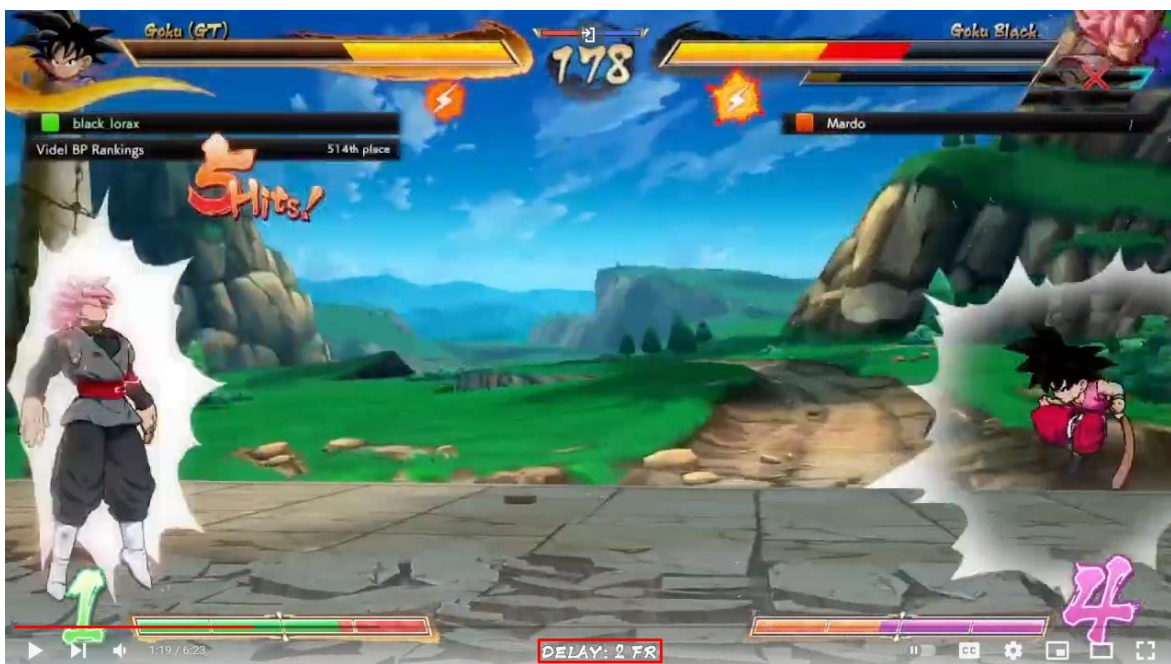


**Figure 13**

*Variable input delay in Dragon Ball FighterZ (ArcSystemWorks, 2017).*

**Figure 14**

*Variable input delay (next frame) in Dragon Ball FighterZ (ArcSystemWorks, 2017).*



This is known as Delay-Based netcode, which has been the norm for the fighting game genre until very recently (Terrano & Bettner, 2001). Delay-Based netcode has various advantages, such as its ease of implementation, in terms of its relationship with the core game loop. Since this solution only applies a delay, the core game loop does not need to be modified, and it is also not CPU intensive (Cannon, 2012).

These benefits are a factor in why Delay-Based netcode has been the norm for so long, despite the loss in consistency and responsiveness. Such disadvantages however are what make online tournaments less prestigious, due to there being more variance in outcome not related to skill, since latency adds another variable to outcome. The precision that players hone for years becomes insignificant with Delay-Based networking, leaving high-level online competition as unviable in fighting games.

## 2.8 Rollback Netcode

Rollback networking takes a very distinct approach to synchronizing both instances of the game simulations. There are three main differences to rollback netcode (Ehlert, 2021):

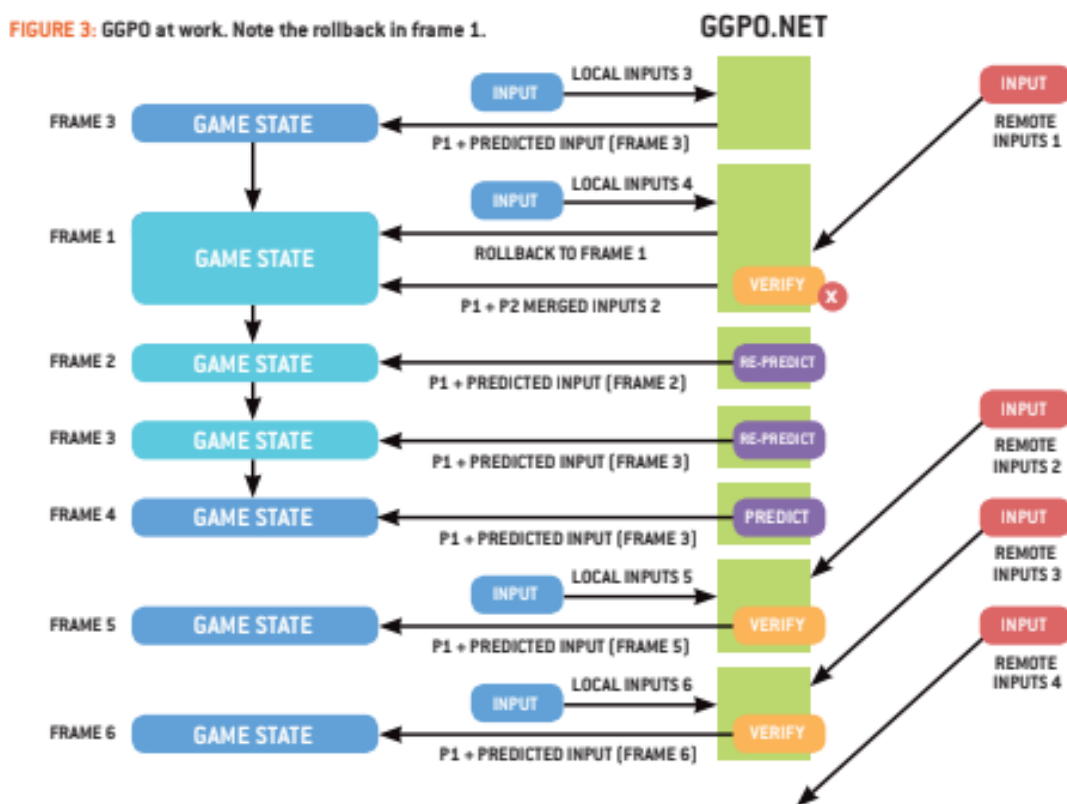
- Only inputs need to be sent and received. No game state information is necessary to be sent.
- Both players' inputs are not required to advance the frame, only the local player's input is necessary. This part completely eliminates the responsiveness problem inherent to Delay-Based netcode, as the game no longer awaits player 2's inputs to process player 1's inputs.
- The remote player's inputs are predicted ahead of time and used to advance the frame. (Client-side prediction)

Considering the same example presented above, Chipp Zanuff's punch is active 3 frames after the input is processed, meaning if punch is pressed on frame 0, the punch will be active on frame 2. This remains true for the local player, but in the remote player's case, their inputs are predicted.

The algorithm for prediction may vary, but usually, the game predicts the same action that the remote player was performing previously. As an example, if Chipp Zanuff was blocking on frame 10, then the system will predict block on frame 11. If this prediction is incorrect, the game will perform a rollback, and re-simulate the frames from the latest game state with correct inputs from both players (See Figure 15).

**Figure 15**

Diagram of rollback netcode in action, from GGPO (Cannon, 2012).



Most humans cannot perform more than 6 digital inputs per second (this includes joystick/analog stick inputs), and if we factor in the norm of 60 FPS in competitive games, this prediction is correct about 90% of times or more (Cannon, 2012).

In Figure 17, the data is from one of *Super Smash Bros. Melee*'s most prestigious tournaments, Smash Summit, where only top-level players are invited. *Super Smash Bros. Melee* is one of the most difficult fighting games in terms of precision with inputs and level of execution necessary to succeed.

Top-level players are normally at 2-3 digital inputs per second approximately, (number does not consider the fact that there are pauses throughout the match due to respawns) which means that even in the most intense parts of the match, top players generally do not surpass more than 6 inputs per second, proving the client-side prediction tends to be correct in most cases (Cannon, 2012).

**Figure 17**

Data from Grand Finals of Smash Summit 11 - Mango vs Zain. Source: Slippi.gg

Category	Player 1 (P1)	Player 2 (P2)
Damage / Opening	27.1	26.1
<b>Defense</b>		
Actions (Roll / Air Dodge / Spot Dodge)	1 / 0 / 3	1 / 1 / 2
<b>Neutral</b>		
Neutral Wins	13 (59%)	9 (41%)
Counter Hits	4 (44%)	5 (56%)
Beneficial Trades	1 (100%)	0 (0%)
Actions (Wavedash / Waveland / Dash Dance / Ledgegrab)	21 / 12 / 50 / 6	35 / 3 / 48 / 12
<b>General</b>		
Inputs / Minute	482.7	531.5
Digital Inputs / Minute	143.6	183.0
L-Cancel Success Rate	94% (49 / 52)	86% (44 / 51)

There is also another situation in which a rollback can be avoided. Fox's (from *Super Smash Bros. Melee*) grab attack lasts for 29 frames, and Fox is not actionable until that attack is over (Refer to Figure 18). If the player who is controlling Fox were to attempt to jump while performing a grab, nothing would happen as Fox is currently locked in an action.

As such, we can apply this concept to rollback networking's client-side prediction formula. If the remote player's inputs do not match with what was predicted, we can also check if the inputs would change the game state by verifying the game state and their character's state. If the remote player's input would not modify the state of the game, then the remote player's inputs are ignored, and the game can progress as normal, even though the client-side prediction was incorrect, further favoring the rollback method.

**Figure 18**

*Fox's grab frame data. Source: MeleeFrameData*



Clearly, in comparison to delay based netcode, rollback netcode is favored in terms of responsiveness, but it is inherently complicated and challenging to integrate into a game. However, a software development kit developed by Tony Cannon in 2009, known as GGPO (“Good Game, Peace Out”) helps with implementation and partially takes care of the rollback architecture. Before being able to implement rollback netcode however, there are three main necessities for the development of our games.

### **2.8.1 Deterministic Simulation**









A deterministic simulation is defined as a simulation where the identical inputs will produce identical outputs every single time. Achieving deterministic simulations does come with some challenges:

- Floating point imprecisions: In programming, floating point numbers are imprecise, and the same calculation with the same numbers will produce very slightly different results. A small difference in results can cause the same action to produce different game state, causing both simulations to diverge, thus producing a desync (Huynh & Valariano, 2019).
- Randomness: All simulations of the game must begin the match with the same seed that is used in the calculation of random numbers (Sun, 2019). Referring to Figure 19, Peach’s turnip special attack in all versions of Super Smash Bros. Peach pulls a turnip and depending on the type, it may do different damage, which can turn the tides of a match completely. If the seed is not consistent, then a rollback may produce a different turnip and cause a desync.

- Execution order: The execution order of different scripts may not be consistent in different game engines. In Unity, there is a setting for script execution order, which allows us to specify which scripts should be executed before others (Unity Technologies, 2022).

**Figure 19**

*Peach's turnips probabilities. Source: Liquipedia.net*

Type	Damage	Probability
 Normal	6%	4445/7424 = <b>59.873%</b>
 Eyebrow Eyes	6%	762/7424 = <b>10.264%</b>
 Line Eyes	6%	635/7424 = <b>8.553%</b>
 Circle Eyes	6%	381/7424 = <b>5.132%</b>
 Carrot Eyes	6%	381/7424 = <b>5.132%</b>
 Wink	10%	508/7424 = <b>6.843%</b>
 Dot Eyes	16%	127/7424 = <b>1.711%</b>
 <b>Stitch Face</b>	34%	127/7424 = <b>1.711%</b>

A deterministic simulation is necessary for rollback systems as it depends on transferring solely the input data. The synchronizing occurs through deterministic simulation and rollbacks when correct inputs are received. When the game performs a rollback, it must ensure that both players' games are in sync, and a miniscule difference in the simulations can produce a butterfly effect, causing the tiny difference to grow throughout the match, leading to a desync.

### ***2.8.2 Independent Simulation of the Game***

As we have noted, the core game loop for most games involves sampling input, calculating the game state, and then rendering the graphics in that specific order. For rollback systems, the game must be capable of calculating game state various times within one frame, because it may take more than one frame for the remote player's inputs to arrive. This means player 1 may receive

player 2's inputs from frame 3 on frame 10, and in this case, the game needs to perform a rollback to frame 3 and re-simulate all the frames in between.

To avoid pauses and maintain a seamless gameplay experience, all of this must be done in a singular frame, and thus, the core game loop must be modified so that it can calculate game state seven times, from frames 3 to 10, before rendering the graphics. Thus, the complexity of development drastically increases when using a rollback system, in comparison to a Delay-Based system, where no changes are needed to the core game loop (Huynh & Valariano, 2019).

The simulated frames are different from normal frames. These frames exclude all audio and visuals and saving, and the system focuses solely on calculating game state until it reaches the frame at which both players are currently on, which lowers the performance of the game as multiple frames of game logic must be computed in a single frame. Michael Stallone speaks of the struggle of implementing 7 rollback frames while maintaining fluid gameplay in his GDC talk: "8 Frames in 16ms: Rollback Networking in *Mortal Kombat* and *Injustice 2*" (Pusch, 2019).

The game must also be capable of advancing real frames without the remote player's inputs, and as mentioned, the game attempts to predict the remote player's inputs, by simply assuming the remote player will be making the same inputs they were making in the previous frame. The prediction algorithm is quite simple and can also be applied to real-life situations. As obvious as it sounds, predicting what a person will do from frame to frame (every 16 milliseconds) can be done with a high level of accuracy. If a person was sleeping, they will most likely be sleeping 16 milliseconds later, or in the next 'frame'.

In fighting games, the same concept is applied, players are unlikely to perform more than 6 inputs per second, and as we noted above, this means the system predicts the remote player's inputs correctly 54 out of 60 times, or 90% success rate, which is the worst-case scenario (Cannon, 2012).

Another challenge that might arise is the rifts or teleports may appear on the screen when a rollback is performed. Because a rollback is technically 'going back in time' and re-simulating frames, there are cases where a character might teleport a short distance, or one of their animations might get interrupted. The game resolves this by skipping the startup animation frames of the move that the correct inputs correspond to, instead of beginning unintended actions and undoing them.



However, if the attack is fast enough, such as Fox's shine in *Melee*, which is active on frame 1, then a rift may still occur (Cannon, 2012). Another evident real-life example is singing the 'Happy Birthday' song. If a person joins late, they normally skip the first few words and synchronize with the other people, and it is hardly noticeable (Core-A-Gaming, 2020). When we consider that a computer is much more precise than a human, it is clear that this method of prediction is not only reliable, but also very optimal performance-wise.

### **2.8.3 *Serialization at Any Moment***

Saving and loading a game state is necessary for rollback networking. The means of doing so may vary from game to game, since different games have different necessities in terms of the information that corresponds to the game state, but the most important thing is that the state can be recreated with all its information.

As for when to save the game state, generally this is done when remote inputs are received, since that way we can ensure that the frame is correct with all its data. These remote inputs act as checkpoints, whether they caused a rollback or not. Ideally, the most recent confirmed frame is saved so that the game is ready to roll back if necessary.

This can be extended with the example mentioned above with Fox's grab and some inputs not affecting game state. If we receive inputs from the remote player while their character is performing a grab, those inputs will not change the game state as the character is locked in the grab state, until the 30th frame (the attack ends on the 29th frame). Thus, we can optimize the serialization by serializing on the last confirmed frame, which would be the 29th frame (Ehlert, 2021).

## **2.9 Blended Delayed Rollback Netcode**

Now that the responsiveness issue is solved through rollback networking, we can apply one final step to add consistency and stability to online play: Adding an input delay much like the one seen in Delay-Based netcode. In delay based netcode, the input delay is variable, adapting to the connection quality, but in rollback networking the input delay is static, because the purpose of the delay is to allow for information packets from player 2 to be received by player 1 (Huynh & Valariano, 2019).



This acts as a buffer or a margin of error to reduce the amount of visually jarring elements or rifts, which can fix the problem described above with Fox's shine attack being active on the first frame. Now with a fixed input delay, the game has a margin of error based on the setting (normally 3 or less), that allows the remote inputs to be received on time, while masking rollbacks visually. This greatly reduces the number of rollbacks performed, and hides almost all the lag (Stallone, 2019).

This is used in *Guilty Gear Strive*, as seen in Figure 20, though the menu says 'Rollback Frames'. The number of frames the game is rolling back in this case depends on the quality of the connection. A solid connection will have fewer frames of rollback; it won't advance as many frames without receiving updates from the other client.

Nevertheless, a high amount of rollback frames means that while your inputs will still go through, there will be a lot of visual teleporting of one or both characters on potentially both clients and a gameplay experience that is likely unreadable and hence, unplayable (Sun, 2019). Overall, rollback netcode softens the impact of latency on gameplay, but it does not eliminate connection problems entirely.

## Figure 20

*Adaptive input delay applied to Guilty Gear Strive online matches. (ArcSystemWorks 2021)*



## 2.10 Case Studies of Popular Fighting Games

*Dragon Ball FighterZ* is one of the most popular fighting games across the Steam Database, along with titles such as *Street Fighter V*, *Mortal Kombat 11* (NetherRealm Studios, 2019), *Tekken 7*, and *Guilty Gear Strive*. Examining Figure 21, we can notice *Dragon Ball FighterZ* at fourth place in terms of concurrent players and peak players within 24 hours, however *Dragon Ball FighterZ*'s all-time peak concurrent players is the highest. This might be due to a number of reasons:

**Figure 21**

*Fighting games sorted by concurrent viewers (Steam database, 2022).*

#	Name	Current	24h Peak	All-Time Peak	
1.	 TEKKEN 7	6,605	7,535	18,966	-
2.	 MORTAL KOMBAT 11	2,938	3,446	35,147	-
3.	 STREET FIGHTER V	2,902	3,937	14,783	-
4.	 DRAGON BALL FIGHTERZ	1,367	2,552	44,303	-
5.	 GUILTY GEAR -STRIVE-	1,349	2,475	31,156	-

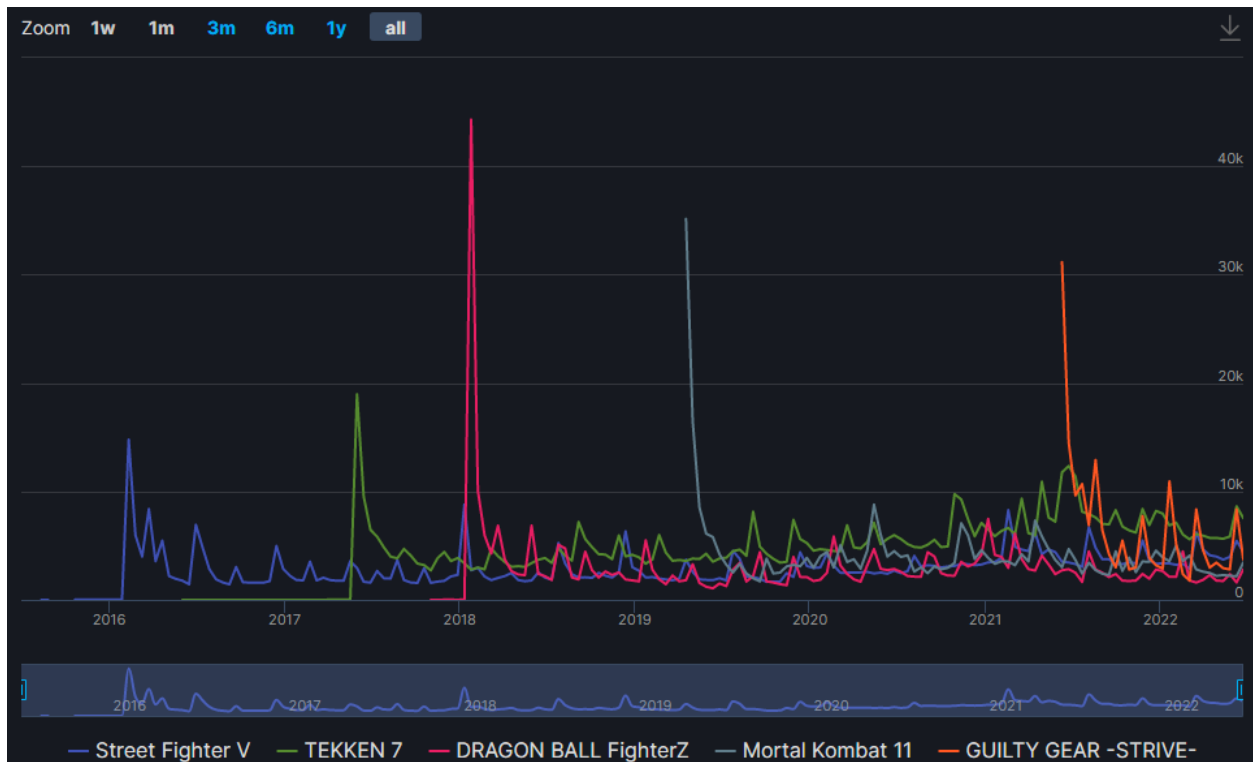
### 2.10.1 Brand Value

Video games are products that are sold that have a brand attached to them, much like general products one might find in a supermarket. Brand value can heavily influence the sales of a product, because it provides the consumer with an expectation about the quality of the product (Gupta et al., 2020).

While most of these games are installments of an already established franchise, none of them have the same amount of brand value as *Dragon Ball*. Out of the five video games mentioned, *Dragon Ball FighterZ* might be the most recognizable one to the general public, largely due to other types of media linked to the franchise, such as the manga and anime, which have been around ever since 1984, and has been featured on television internationally all around the world. For established games such as the five mentioned above, they tend to peak in concurrent players during the first weeks of release, due to the amount of hype the games generate prior to being released (See Figure 22).

**Figure 22**

*Trend lines of concurrent players across 5 different games until today. (Steam Database, 2022).*



Referring to Figure 22, *Dragon Ball FighterZ*'s trendline has the largest amount of amplitude, having both the highest peak, and then being consistently among the lowest in number of concurrent players. This begs the question: Why does *Dragon Ball FighterZ* have a lower player count in comparison to the other fighting games? All of these fighting games were released from 2016 onwards, the oldest one being *Street Fighter*, followed by *Tekken 7*, *Dragon Ball FighterZ*, *Mortal Kombat 11*, and lastly, *Guilty Gear Strive*.

Therefore, we can rule out any argument about game novelty. Obviously, there are core differences in game play that might cause people to play one over the others, but when we consider that all of those games have rollback netcode, except for *Dragon Ball FighterZ*, we begin to see why it remains as the least popular on Steam's database (Hills, 2021).

### 2.10.2 Netcode Differences

Multiplayer fighting games are designed for competition, with another human being, because we get a higher level of satisfaction from defeating another person as opposed to an AI.

This is partly due to the level of execution that is needed to play the game. If an AI is simply programmed to not block in specific cases, which is seen in *Ultra Street Fighter IV* (Capcom 2014), then winning becomes really easy, and is not gratifying for the player.

Generally, people experience a higher level of satisfaction when they overcome a challenge, than when it is handed to them for free (Core-A-Gaming, 2016). Thus, we arrive at the necessity of an opponent when playing the game.

Fighting games have a particularly high barrier of entry when learning the game, because there are a lot of concepts and fundamentals to learn. Players may spend hours, days or even weeks in the ‘Training Mode’ most fighting games have, to hone their abilities, and then attempt to play versus other people to test themselves and what they learned.

However, not all people are ready to travel for miles to attend a tournament (Core-A-Gaming, 2016). This is where online play might become useful, since it offers experience versus another person, rather than an AI, but if the network conditions are poor, the experience can be frustrating, due to the delay between inputs and actions performed in-game.

*Super Smash Bros. Ultimate* (Nintendo, 2018) is the newest fighting game from Nintendo, released in 2018. Generally, when a fighting game developer releases a new installment in a fighting game series, most people abandon the previous game and play the newer one (See Figure 23).

*Ultimate* was successful in terms of sales and offline tournaments, but during the pandemic lockdown, all competition ceased to exist. Only one tournament was attempted, which was shut down due to poor gameplay quality. Professional players simply did not want to play the game in online conditions, whereas *Super Smash Bros. Melee*, developed by HAL Laboratories, debuted in 2001, and has had three successors, and despite all of this, still has a vibrant competitive community. Multiple major tournaments have been run by its grassroots community throughout the pandemic, with no signs of stopping.

Part of *Melee*’s success can be attributed to its fast-paced gameplay and challenging but rewarding mechanics, but the difference in competition can be solely attributed to the online experience. *Ultimate* uses Delay-Based netcode, whereas *Melee* uses rollback netcode, which was

developed by a passionate player who goes by the name of ‘Fizzi’, who created *Slippi.gg* (Cohu, 2021). It is no coincidence that a twenty-year-old game is able to outshine a modern fighting game of the same series in terms of popularity and competition.

### Figure 23

3 Fighting games with their predecessors in terms of player count. (Steam Database).

#		Name	Current	24h Peak	All-Time Peak	
1.		Mortal Kombat 11	3,616	3,644	35,147	-
2.		Street Fighter V	3,123	3,725	14,783	-
3.		GUILTY GEAR -STRIVE-	1,791	2,475	31,156	-
4.		Mortal Kombat X	616	804	15,743	-
5.		Ultra Street Fighter IV	256	311	5,547	-
6.		GUILTY GEAR Xrd -REVELATOR-	132	182	1,614	-

Another interesting case is actually *Mortal Kombat X* (NetherRealm Studios 2015) as well as *Dragon Ball FighterZ*. *Mortal Kombat X* used delay based netcode ever since its release in 2015, but throughout its product lifetime the players were constantly disappointed with its netcode. Delay-Based netcode causes all inputs to be delayed, which hinders people who attempt to react to what they see on the screen, because everything they input is now delayed. This forces them to predict what the opponent is going to do, which adds uncertainty in outcome, defined as luck by *Magic the Gathering* creator, Richard Garfield (Core-A-Gaming, 2017).

This caused *Mortal Kombat X*'s players to push the developers to implement rollback netcode, which NetherRealm Studios accomplished, but not without its struggles (Gordon, 2022). *Mortal Kombat X* was not prepared for rollback netcode, and because of this, NetherRealm Studios took approximately “eight man-years of time split between many programmers in 10 months” while implementing rollback netcode.

However, after developing the system, NetherRealm Studios had an easier time implementing rollback netcode into their future titles, such as *Injustice 2* and *Mortal Kombat 11* (Pusch, 2019). On the other hand, the case of *Skullgirls* (Autumn Games, 2012) implemented Tony

Cannon's GGPO framework very early on into development, which heavily facilitated their development process of the netcode (Pusch 2019).

*Dragon Ball FighterZ* faces this problem to this day, and for years, fans have been pushing for better netcode, but despite their efforts, no indications have been given from ArcSystemWorks about it, and fans remain hopeful, but from Steam's Database, *Dragon Ball FighterZ's* average play time has dropped, and is the lowest of the five games mentioned above (Gordon, 2022).

### 3. Development

In the development, we will outline the experiment referred to in the methodology section, as well as the questions used in the questionnaire following the experiment. Then, we will outline an example of rollback pseudocode to understand how it could be implemented.

#### 3.1 Dolphin Versus Slippi

As mentioned, the experiment consisted of having people play the same exact game (*Super Smash Bros. Melee*) online, but with two different netcodes. Dolphin emulator uses delay based netcode, and Slippi uses rollback netcode. To ensure consistent connections, the test subjects played with assistants for the experiment from Italy and the UK, resulting in a network latency of roughly 50-90ms.

Modern fighting games such as *Guilty Gear Strive* and *Street Fighter V* use regional matchmaking to ensure low latency values for the players. As such, limiting the connection to the European region makes the experiment environment both practical and realistic.

The play time for each person was 2 minutes on each emulator. For Dolphin emulator, we were averaging roughly 60-90ms of latency. Since it uses Delay-Based netcode, we configure an input pad buffer system (see Figure 13), which essentially adds a number of frames of delay so that the game has enough time to send and receive the data required to advance to the next frame.

The Dolphin emulator Netplay guide recommends 1 frame of pad buffer per 15ms of latency, but throughout the experiment we were forced to raise the pad buffer higher due to connection quality fluctuation. We tested the recommended amount of 6, but then added 10 more frames of buffer due to performance, therefore 16 frames of pad buffer were used for this

experiment. On the other hand, Slippi never had performance issues and we were able to maintain the delay at 2 frames throughout.

After the play time, test subjects were asked to fill out a questionnaire based on their experience playing on the different emulators with their respective netcodes. The questionnaire consisted of nine questions, three of which acted as the metrics we used to gauge the player experience (The answers to these questions were recorded on a scale from 1-6):

- The connection's stability: if the connection is unstable, then players are likely to quit the game (Tseng et al., 2011).
- The player's capacity to react in-game: as mentioned, fighting games require lots of precision and quick reactions; if players feel incapable of reacting to what the opponent does, then the strategies employed when the game is played at a high level can change, aforementioned as lag tactics.
- Input responsiveness: if the player finds trouble in controlling their character due to the inherent delays caused by online play, then they may feel restrained in how they play, leading to a negative player experience.

We will refer to these metrics as 'Stability', 'Reactability', and 'Responsiveness' respectively in the results section.

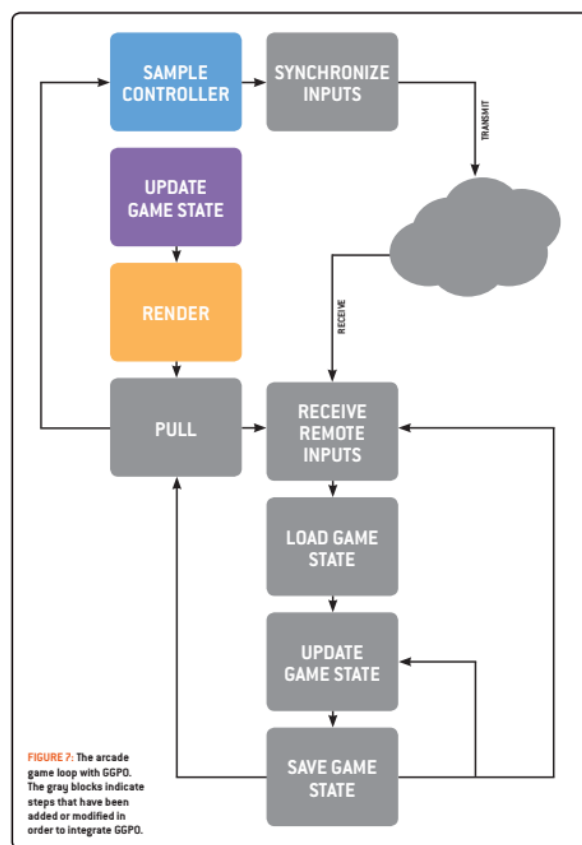
After that, they were asked questions about their method of connection as well as how they would define the quality of their connection at home. This can give insight into how many people use Wi-Fi or an Ethernet connection, which matters because it is common knowledge that a wired connection is both faster and more stable than a wireless connection. We could predict that people that use a wired connection tend to experience less latency normally, and therefore, may notice a bigger difference when playing at a high latency or delay.

Likewise, the participants were also asked about their experience in both competitive and fighting games. Fighting game players may be more prone to understand the differences in netcode quality, but people with experience in competitive games such as *Counter Strike: Global Offensive* or *League of Legends*: may also be perceptive to the differences in latency and connection quality in the two netcodes, because of how connection can influence the outcome of a match.

Finally, we also asked the participants generic demographic questions which may be interesting to look at, since we might be able to find some form of correlation between demographics and their ratings of their experience. For reference, all of the questions asked are available in the appendix. In the next section, we will analyze the results from this experiment.

### 3.2 Rollback Example

In this section, we will explore an example of rollback netcode in pseudocode, following Tony Cannon's diagram representing GGPO (see figure 29). As for our constants and variables, we store information relevant to the synchronization of the game as seen in figure 30.



**Figure 25:** Game loop with GGPO (Cannon, 2012).

The code presented below has comments on functionality and instructions for implementation.



```
#region Constants
    uint MAX_ROLLBACK_FRAMES; // Specifies the maximum number of frames
    that can be re-simulated within one frame.
    uint FRAME_ADVANTAGE_LIMIT; // Specifies the number of frames the
    local client can progress ahead of the remote client
    int INITIAL_FRAME; // Specifies the initial frame the game starts
    in, cannot rollback before this frame
#endregion

#region Variables

    int localFrame; // Tracks the latest frame for the local client
    int remoteFrame; // Tracks the latest frame we received for the
    remote client
    int syncFrame; // Tracks the latest frame where we confirmed a
    synchronization
    int remoteFrameAdvantage; // Tracks the frame advantage from the
    remote client

#endregion

#region Methods

    void InitializeVariables()
    {
        localFrame = INITIAL_FRAME;
        remoteFrame = INITIAL_FRAME;
        syncFrame = INITIAL_FRAME;
        remoteFrameAdvantage = 0;
    }

    void StoreGameState() // Store the game state for the current frame
    locally
    {
        // Store game state ==> Inputs, animation, UI, object lifetime,
        sound fx, particle simulation
    }

    void RestoreGameState() // Restore game state to the frame set in
    sync frame
    {
        localFrame = syncFrame;

        // Restore game state ==> Inputs, animation, UI, object
        lifetime, sound fx, particle simulation
    }
}
```

```
void UpdateInput() // Prepare inputs for both local and remote player
(predict remote player's inputs)
{
    // 1. Predict remote player's input if not available yet
    // 2. Setup inputs from both the local and remote player for use
in UpdateGame()
}

bool RollbackCondition() // A condition that prevents rolling back
if we are not ahead of the latest synced frame
{
    return localFrame > syncFrame && remoteFrame > syncFrame;
}

// A condition that only allows the local client to progress a fixed
amount ahead of the remotes
bool IsTimeSynchronized()
{
    // How far is the local client ahead of the remote client?
    int localFrameAdvantage = localFrame - remoteFrame;

    // Frame advantage from the remote client vs local client
    int frameAdvantageDifference = localFrameAdvantage -
remoteFrameAdvantage;

    return localFrameAdvantage < MAX_ROLLBACK_FRAMES &&
frameAdvantageDifference <= FRAME_ADVANTAGE_LIMIT;
}

#endregion

void OnStartGame()
{
    InitializeVariables();
}

void Update()
{
    // Update Network
    // remoteFrame = latest frame received from the remote client
    // remoteFrameAdvantage = (localFrame - remoteFrame) received
from the remote client

    // Update Sync
    //// Determine the sync frame
```

```

        int finalFrame = remoteFrame; // We only check inputs until the
remote frame, no more inputs after

        // If the remote client is ahead of local client, do not check
past the local frame
        if (remoteFrame > localFrame) finalFrame = localFrame;

        // Find the first frame where predicted and remote inputs do not
match (syncFrame + 1 until finalFrame)

        // if (mismatchFound) sync frame = foundFrame - 1;
        //// Found frame is the first frame where inputs do not match,
we assume previous frame is correct

        // else syncFrame = finalFrame;

        if (RollbackCondition()) // If we should rollback
        {
            // 1. Execute Rollbacks
            // RestoreGameState();

            // 2. Use inputs from(syncFrame + 1) through localFrame

            // 3. Rollback Update until we reach the local frame -
Resimulation
                // Update Input()
                // Update Game()
                // Store Game State()
        }

        if (IsTimeSynchronized())
        {
            // GameUpdate() - Game Logic

            // Incrementlocalframe - localFrame++;

            // GetLocalInput() - Read input and associate it with its
frame

            // SendInputNetwork() - Send input to remote client

            // UpdateInput()

            // UpdateGame()

            // StoreGameState()
        }

```

There are various optimizations that can be made and must be made because simulating various frames of game logic within a single frame can lower the game's performance. In order to increase the amount of frames we can roll back to, these optimizations are required. This pseudocode simply serves as a basic conceptual map of how rollback netcode might be built.

## 4. Results of the Experiment

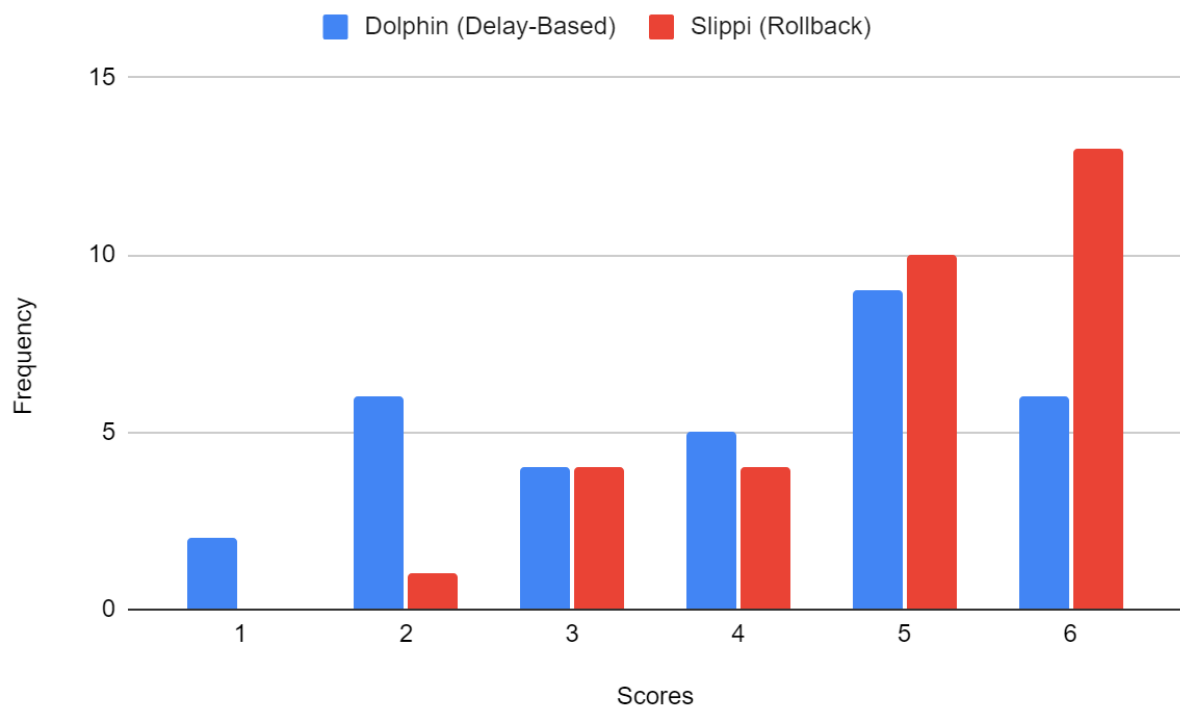
To reiterate, our hypothesis states that rollback netcode would affect the player experience in a positive way by improving the stability of gameplay and its responsiveness by reducing the impact of network latency, as only inputs from the local player are needed to advance to the next frame. The null hypothesis would then mean there is no improvement by implementing rollback netcode, which requires the scores to either favor Dolphin emulator or to not be very different.

### 4.1 Quantitative Analysis

For the first metric (Connection stability), the data favors Slippi as the preferred emulator, meaning rollback netcode did prove to be more stable in comparison to Dolphin emulator. Dolphin emulator had an average score of 3.97, while Slippi had an average score of 4.94.

**Figure 24**

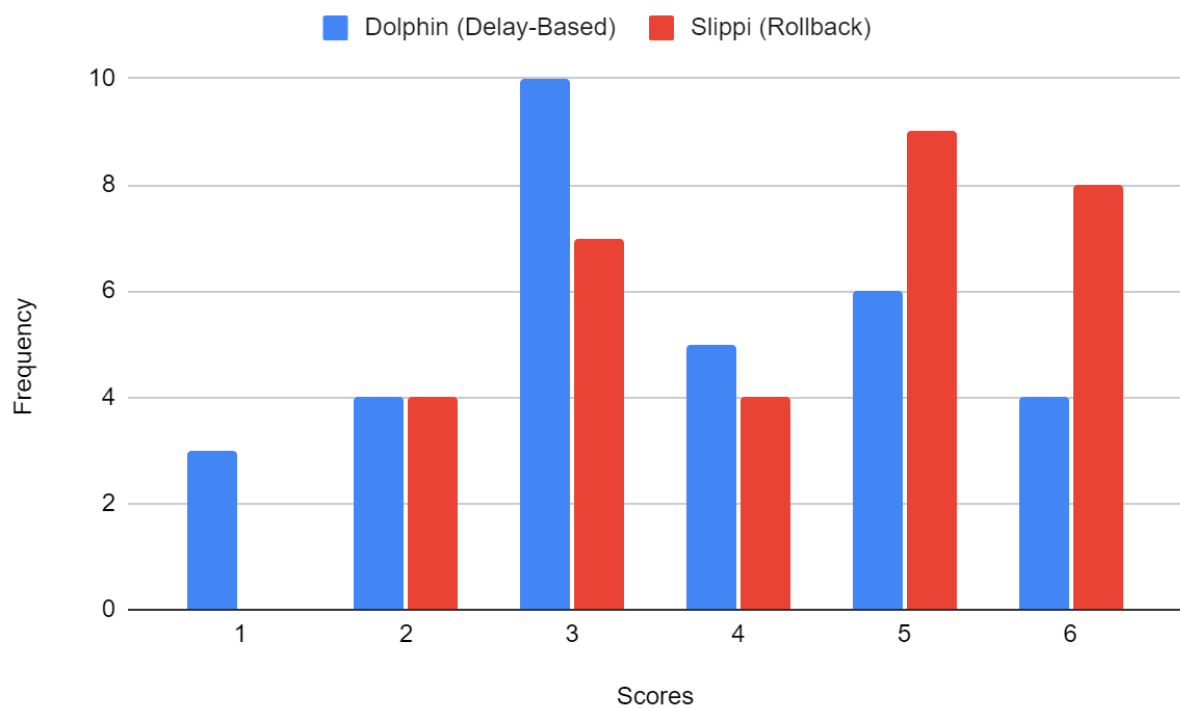
*Network Stability in Relation to Netcode*



Question 2 asks the test subjects how capable they felt of reacting to their opponent's moves, shown in Figure 25. In this case, Dolphin emulator had an average score of 3.59, while Slippi had 4.31, which once again favors rollback netcode.

**Figure 25**

*Capacity for Reacting in Relation to Netcode.*



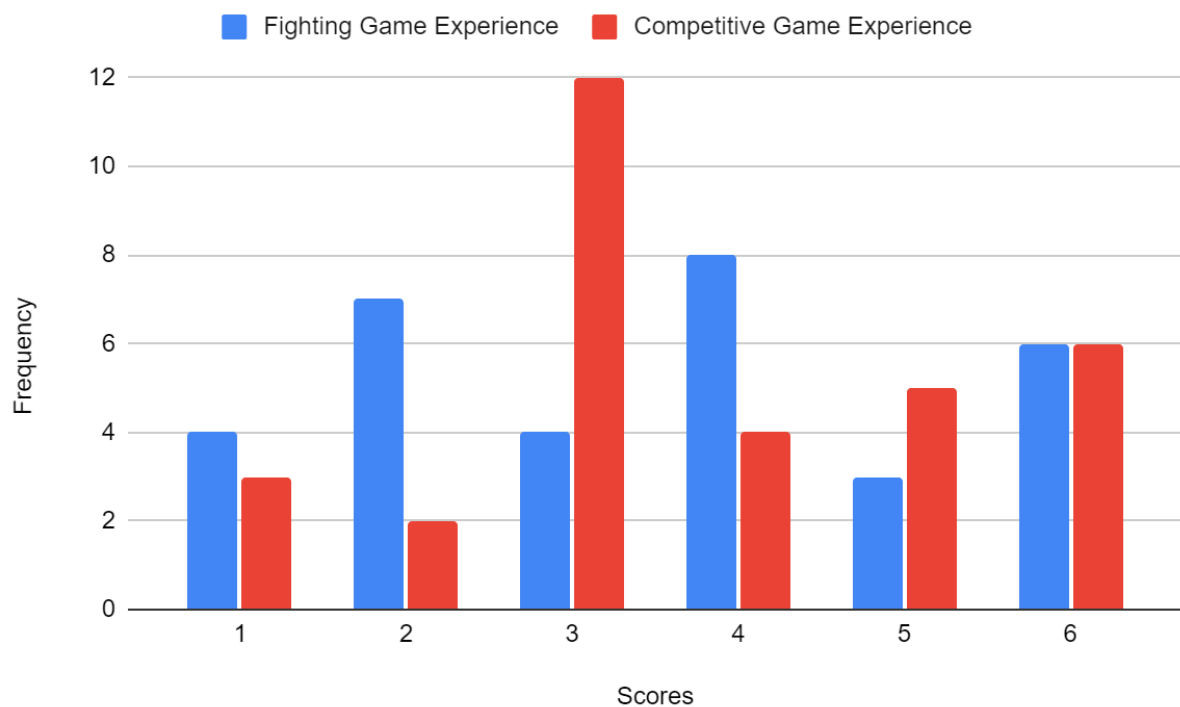
However, unlike the first metric, the results in this metric are slightly more skewed towards the center, which could be caused by the nature of the question itself. Reactions in fighting games can mean various things. Sometimes it can mean reacting to a specific move that may be slow, or it can mean reacting to your opponent's movement preceding the attack, which usually requires knowing how the opponent's character functions as well as your own, as well as knowing their frame data to choose an appropriate response.

As mentioned, there were questions relating to the level of experience of the participants in both fighting and competitive games, shown in Figure 26. The mean scores were 3.53 and 3.75 respectively, which is equivalent to having played a few fighting games as well as having some experience with ranked modes in online games (as specified in the options in answering the question).

This could mean that the participants simply did not know what to look for when reacting which could impact results for this metric. Nevertheless, the data shows that the participants felt more comfortable reacting with rollback netcode (which is expected due to the faster response time).

**Figure 26**

*Competitive and Fighting Game Experience.*



The third question focuses on input responsiveness. In terms of input delay, there was a 14-frame input delay difference between the different netcodes (Dolphin used 16 frames, Slippi used 2 frames, equates to 224ms of delay difference), therefore the difference is expected to be very notable in the results.

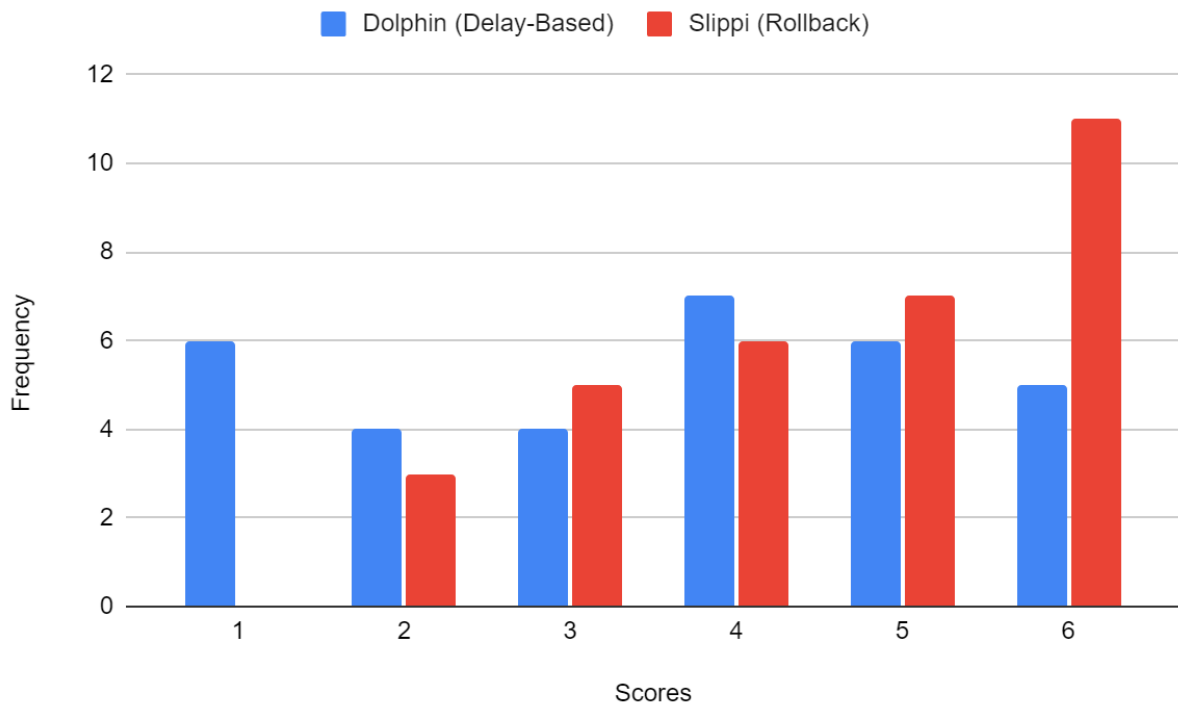
Referring to Figure 27, the results once again favor rollback netcode, in a more drastic manner. Input responsiveness is something that is felt directly in gameplay because the player expects their character to move in a specific way.

However, depending on the user, they may be accustomed to more input delay than others, as developers often force VSync on consoles, while on PC, users usually have the option to disable

it. VSync forces the GPU to match the monitor's refresh rate, which generally slows down the speed of gameplay by adding a delay. If a person plays more often on consoles rather than PC's, they may be less perceptive of latency fluctuations and delays.

**Figure 27**

*Input Responsiveness in Relation to Netcode*



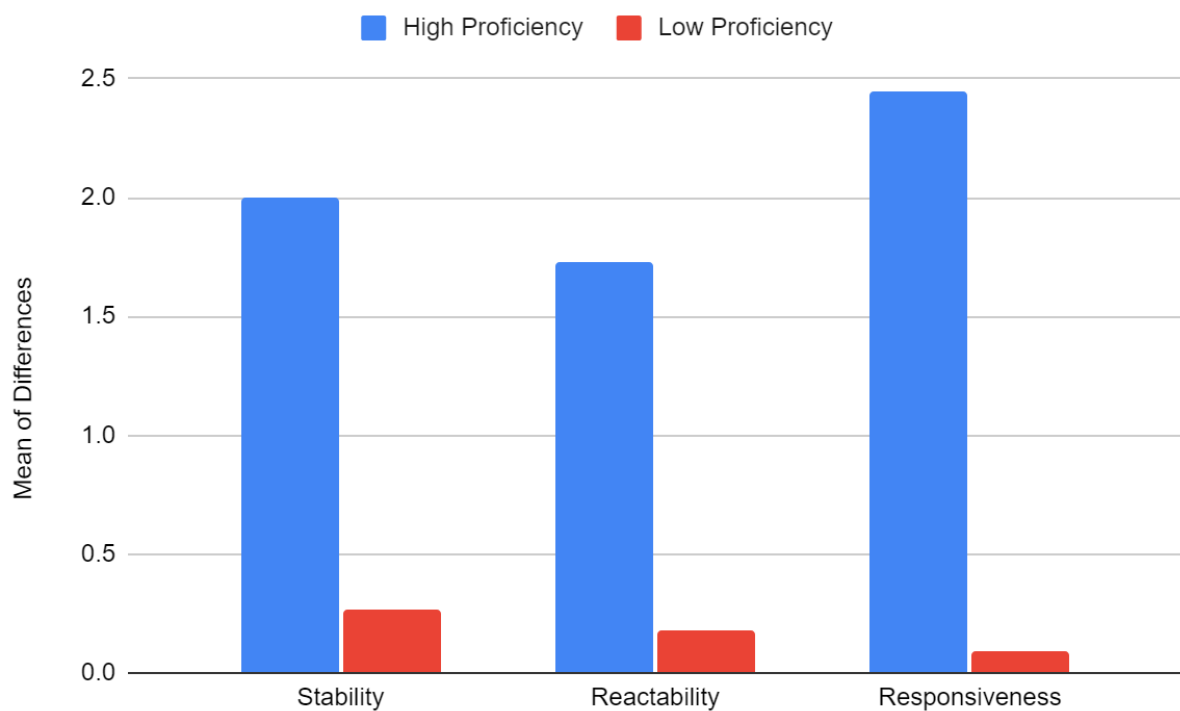
Next, we can filter data by competitive and fighting game experience to determine if there is a possible correlation between experience and their perception of the netcodes. To do so, we use the data obtained that indicates their experience/proficiency with competitive and fighting games, and average it, to obtain an 'overall proficiency score' for each person. Then, we compare high level proficiency scores (4 or more) vs low level proficiency scores (3 or less).

Figure 28 displays the mean of the differences in each metric for the two different categories of proficiency we have described. Undeniably, highly proficient players had bigger disparities in their ratings of the netcodes, which suggests a strong correlation between proficiency/experience and their ability to perceive differences in gameplay caused by the netcodes.

Likewise, Figure 29 shows data filtered by type of connection used at home for the three metrics. We compare Ethernet to Wi-Fi on the three metrics, with the expectation that ethernet users will be more perceptive of the differences in the netcode, due to them being accustomed to the lower latency and higher stability that Ethernet provides over Wi-Fi.

**Figure 28**

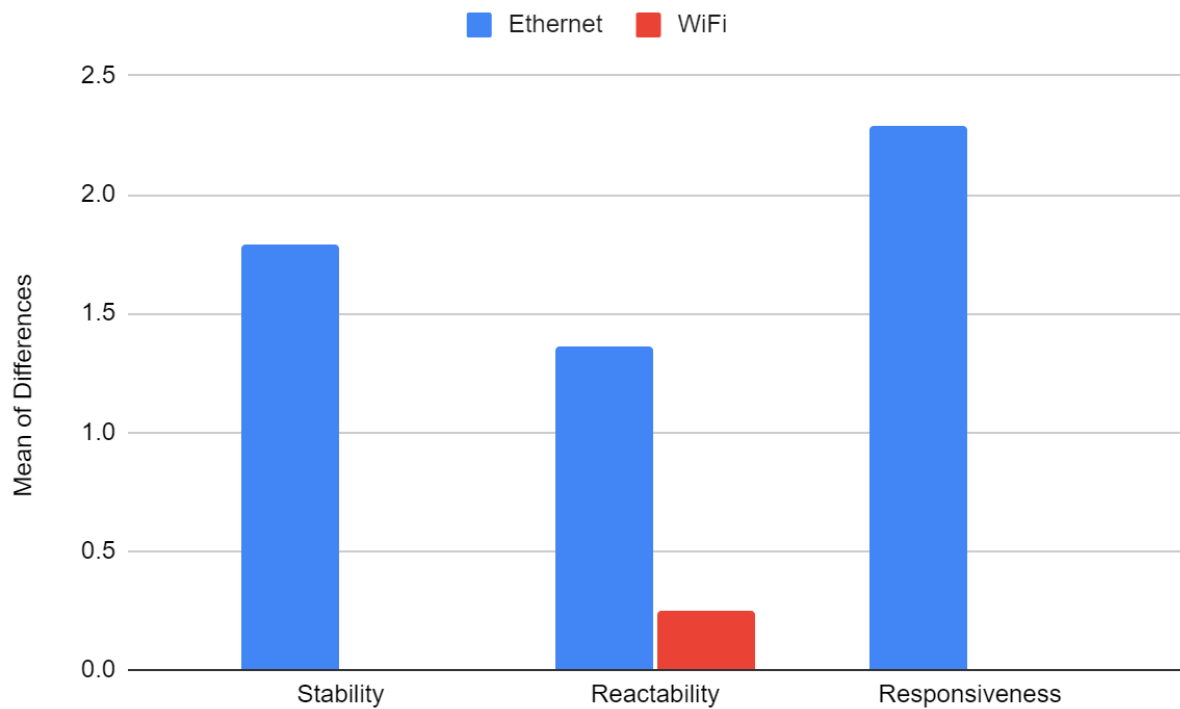
*Netcode Perception in Relation to Experience and Proficiency*





**Figure 29**

*Netcode Perception in Relation to Experience and Proficiency*



As we predicted, the scores for people that use Ethernet showed a very significant disparity between the quality of netcodes, while the people that use Wi-Fi rarely noticed a difference, if at all.

Another interesting question is the participant's definition of their connection at home. In this question, participants defined their connection in one of four ways:

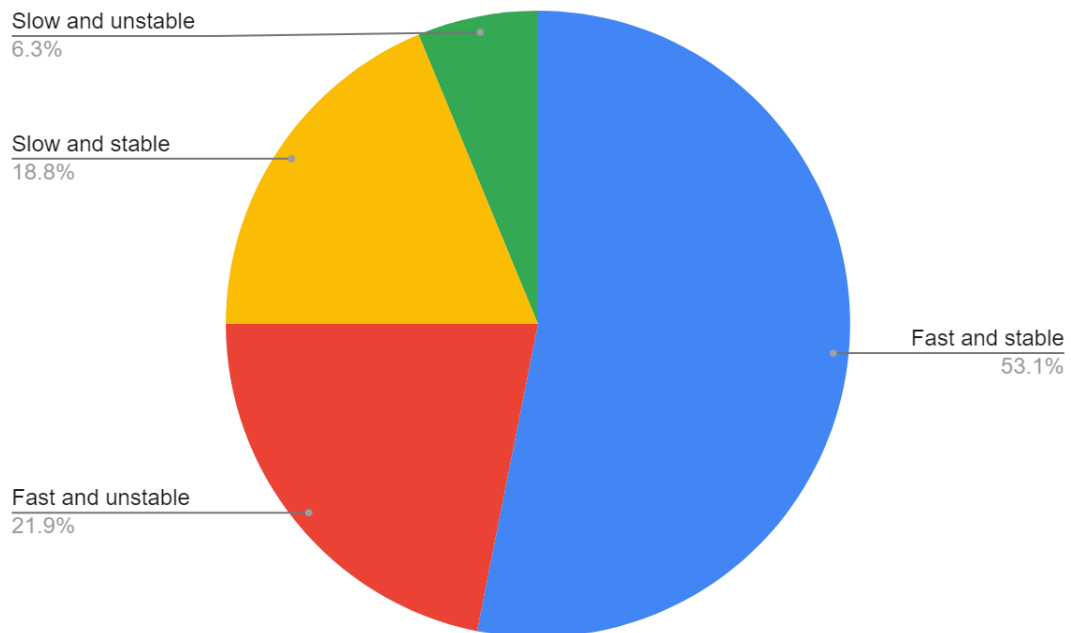
- Fast and stable
- Fast and unstable
- Slow and stable
- Slow and unstable

Figure 30 shows that 75% of people described their connection as 'fast', and roughly 70% of those people described their connection as stable as well. Interestingly, of the people with fast and stable connections, 35% of them use Wi-Fi, which likely shows a lack of understanding in how connection quality works, because as previously mentioned, Wi-Fi is less stable due to it

being wireless. As for the other results, the vast majority of people that did not define their connection as fast and stable stated that they use Wi-Fi at home.

**Figure 30**

*Connection Perception at Home of Participants*



#### 4.2 Statistical Analysis

To analyze our data, we will perform a power test, to check our chance of rejecting the null hypothesis correctly, followed by a paired t-test to test for statistical significance. Since the data is different for the three metrics, we must use both of these processes for each of them, and then determine the better netcode based on the results. We will also use a 95% confidence interval for statistical significance ( $p = 0.05$ ), along with 32 degrees of freedom as our sample size is 32. This means our threshold for statistical significance is 1.694.

**Table 1***Metric 1: Connection stability in relation to netcode*

	<b>Simulation 1: Dolphin</b>	<b>Simulation 2: Slippi</b>	<b>Mean/Standard Deviation of Differences</b>
Mean Score	3.97	4.94	0.97
Standard deviation	1.60	1.16	1.98

With a sample size of  $n = 32$ , we obtain a power of 0.87, which means we have an 87% chance of rejecting the null hypothesis correctly. We then obtain a t-score of 2.77 which is greater than the threshold value of 1.694, meaning our value from the first metric is statistically significant.

**Table 2***Metric 2: Capacity for reaction in relation to netcode*

	<b>Simulation 1: Dolphin</b>	<b>Simulation 2: Slippi</b>	<b>Mean/Standard Deviation of Differences</b>
Mean Score	3.59	4.31	0.72
Standard deviation	1.50	1.40	2.07

This results in a t-score of 1.97, which once again is statistically significant, but our power for this metric is 63%.

**Table 3***Metric 3: Input responsiveness in relation to netcode*

	<b>Simulation 1: Dolphin</b>	<b>Simulation 2: Slippi</b>	<b>Mean/Standard Deviation of Differences</b>
Mean Score	3.56	4.56	1.00
Standard deviation	1.74	1.37	2.31

The t-score is 2.31, and our power is 82% for this metric, meaning it is statistically significant.

Since all three metrics are statistically significant, and all of the average scores for rollback netcode are higher than Delay-Based netcode, we can reason that rollback netcode is superior for the online player experience. From our evidence, it is also evident that players with a higher level of proficiency and experience in fighting and competitive games perceive the differences in netcode to a higher degree.

By nature, fighting games are competitive, and we have empirical evidence indicating that netcode quality is highly valued, if not essential to giving the players the best possible game experience, which means we can reject the null hypothesis and conclude that rollback netcode leads to a better player experience in fighting games.

## 5. Conclusions and Limitations

As we have noted throughout the paper, online modes for video games are incredibly prevalent in the industry, and despite this, fighting games remained offline for the greater part of the decade due to their poor netcodes. In fighting games, players value competition and all the skills required to win, such as reactions and precision. Competition stems from players who have a desire to win, and fighting games provide players with a plethora of options and possibilities that allow players to constantly improve.

To ensure that players feel rewarded for the practice they put in, the game must also work in a manner where it allows the players to display their precision and skill, meaning the game must be responsive and consistent, so that the players that put more work in their practice have a higher chance of winning. An unresponsive game experience adds uncertainty in outcome, which is exactly what rollback netcode attempts to improve.

However, as mentioned, rollback netcode requires a high level of understanding of multiplayer game programming as well as ample preparation prior to and during development. Rollback netcode inherently has its challenges, such as ensuring that both simulations are playing near the same frame, maintaining game responsiveness, reducing the effects of lag, and avoiding desyncs. Despite all this, if rollback netcode is built correctly, it allows for intercontinental matchmaking as well as seamless online gameplay experiences.

As we noted in the results section, the data clearly reflects that we can reject the null hypothesis and accept our hypothesis because the data heavily favors rollback netcode in providing the best gameplay experience across all our metrics. More importantly, the data shows that the difference is way more apparent for players with more experience with competition and fighting games, which indicates that rollback netcode aids in forming a competitive esports scene, which can lead to more exposure and success for the game itself.

In the second metric used, however, we mentioned that the question may have been unclear, as participants may not exactly know what they were reacting to. A potential solution could be to set up a reaction test within the game, which is possible with the training tools. The test would report the reaction speed in frames, which would be an objective way to measure the difference between netcodes.

Apart from this, we could've also asked if the participants were more accustomed to playing games on PC or on consoles, because as we mentioned, consoles generally limit their FPS to 60 by using VSync. This is especially interesting because fighting game tournaments are generally played on consoles, meaning they play on higher input latency caused by the hardware. It would not be surprising to see fighting game players push to make PCs the standard for tournaments, but it would likely cause problems for the budget required to run tournaments.

Fighting games are all about competition within their communities. Rollback netcode can connect communities across countries and continents by making more online matches playable. There are cases where rollback netcode may be too costly, as seen with *Mortal Kombat X* and *Dragon Ball FighterZ*, but if rollback netcode is thought out and planned for, the lower development costs may be worth the effort, as was the case for *Skullgirls*. Regardless, we have proven that rollback netcode indeed leads to a more positive player experience in every way.

## Bibliography

Bornemark, O. (2013) Success Factors for E-Sport Games. Sweden, Umea University. Retrieved 27th of June 2022. Retrieved from:

<https://webapps.cs.umu.se/uminf/reports/2013/001/part1.pdf#page=7>

Cannon, T. (2012) Fight the Lag! The Trick Behind GGPO's Low-Latency Netcode. Game Developer Magazine, Retrieved 29th of June 2022. Retrieved from: [ggpo-gdmag-article.pdf](https://www.gamedeveloper.com/game-development/fight-the-lag-the-trick-behind-ggpo-s-low-latency-netcode)

Cannon, T. (2017). EVO 2017: GGPO PANEL. [Video]. YouTube.

Retrieved from: <https://youtu.be/k9JTIn1SVQ4>

Carter, J. (1993), "Insert Coin Here: Getting a Fighting Chance", *Electronic Games*, no. 10, archived from the original on April 2, 2016, Retrieved December 16th, 2014.

Retrieved from:

<https://archive.org/details/Electronic-Games-1993-07/page/n15/mode/2up?view=theater>

Chen, K.-T., Huang, P. & Lei, C.-L. (2006). How sensitive are online gamers to network quality? In Communications of the ACM (Vol. 49, Issue 11, pp. 34–38). Association for Computing Machinery (ACM). DOI: <https://doi.org/10.1145/1167838.1167859>

Childers, J. (2019). 15 Video Games That Pioneered Online Multiplayer Before It Was Popular. GameRant.

Retrieved from:

[https://gamerant.com/earliest-video-games-online-multiplayer/#:~:text=12%20Neverwinter%20Nights%20\(1991\)&text=Long%20before%20Bioware%20broke%20hearts,game%20capable%20of%20online%20multiplayer.](https://gamerant.com/earliest-video-games-online-multiplayer/#:~:text=12%20Neverwinter%20Nights%20(1991)&text=Long%20before%20Bioware%20broke%20hearts,game%20capable%20of%20online%20multiplayer.)

Cohu, C. (2021) Rollback Networking in Peer-To-Peer Video Games. Retrieved 27th of June 2022. Retrieved from:

<https://cameroncohu.com/wp-content/uploads/2021/05/Rollback-Networking-in-Peer-to-Peer-Video-Games.pdf>

Core-A-Gaming. (2016). Analysis: Why Fighting Games are Hard. [Video]. YouTube. Retrieved from: [https://www.youtube.com/watch?v=AGrIR\\_jlLno](https://www.youtube.com/watch?v=AGrIR_jlLno)

Core-A-Gaming. (2018) Analysis: The Effects of Salt. [Video]. YouTube. Retrieved from: [www.youtube.com/watch?v=ZbjSeBOP-MA&](https://www.youtube.com/watch?v=ZbjSeBOP-MA&)

Core-A-Gaming. (2020) Analysis: Why Rollback Netcode is Better. [Video]. YouTube.

Retrieved from:

[https://www.youtube.com/watch?v=0NLe4IpdS1w&list=PLWibhIYLOq-T7XwgHe2y2hBE8Zr\\_yeODi&index=6](https://www.youtube.com/watch?v=0NLe4IpdS1w&list=PLWibhIYLOq-T7XwgHe2y2hBE8Zr_yeODi&index=6)

Dolphin Emulator Project. (n.d.). *Dolphin Netplay Guide*. Dolphin Emulator. Retrieved from: <https://dolphin-emu.org/docs/guides/netplay-guide/>

Ehlert, A. (2021) Improving Input Prediction in Online Fighting Games. Stockholm, Sweden, KTH Royal Institute of Technology. Retrieved 25th of June 2022.

Retrieved from: <https://www.diva-portal.org/smash/get/diva2:1560069/FULLTEXT01.pdf>

Fiedler, G. (2010). What Every Programmer Needs to Know About Game Networking. Retrieved 28th of June 2022. Retrieved from:

[https://gafferongames.com/post/what\\_every\\_programmer\\_needs\\_to\\_know\\_about\\_game\\_networking/](https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/)

Gifford, K. (2010). Being The Very Best at Fighting Games. 1UP. Archived from the original on June 29, 2011. Retrieved 8th of June 2022.

Retrieved

from: <https://web.archive.org/web/20110629070056/http://www.1up.com/news/fighting-games>

Gill, P. (September 24, 2020). "Street Fighter and basically every fighting game exists because of Bruce Lee". *Polygon*. Archived from the original on March 10, 2021. Retrieved March 24, 2021.

Retrieved from:

<https://www.polygon.com/gaming/2020/9/24/21440150/bruce-lee-movies-street-fighter-fighting-games>

Glazer, J. & Sanjay, M. (2015). Multiplayer Game Programming. Crawfordsville, Indiana, Pearson Education. Retrieved 15th of June 2022.

Retrieved from:

[https://www.academia.edu/43559397/Multiplayer\\_Game\\_Programming?pop\\_sutd=false](https://www.academia.edu/43559397/Multiplayer_Game_Programming?pop_sutd=false)

Gordon, J. (2022). Dragon Ball FighterZ developers previously said it'd be difficult to add rollback netcode, but could that change in the future? Event Hubs.

Available Online:

<https://www.eventhubs.com/news/2022/jan/04/dbfz-developers-difficult-online-change/>

Grey, J. (2021). Would you buy a new fighting game if it did not have rollback netcode? Event Hubs. Retrieved 30th of June 2022.

Retrieved from: <https://www.eventhubs.com/news/2021/aug/01/rollback-netcode-buy/>

Gupta, S., Gallear, D., Rudd, J. & Foroudi, P. (2020). The Impact of Brand Value on Brand Competitiveness. *Journal of Business Research*. Retrieved 3rd of July 2022.

DOI: 10.1016/j.jbusres.2020.02.033.

Hills, D. (2021). After years of debate, Katsuhiro Harada explains how Tekken 7's variable rollback netcode actually works and why it feels different. Event Hubs.

Retrieved from:

<https://www.eventhubs.com/news/2021/jun/12/tekken-rollback-netcode/>



Howarth, J. (2022). 50+ Amazing Video Game Industry Statistics (2022), Exploding Topics. Retrieved 30th of June 2022. Retrieved from:

<https://explodingtopics.com/blog/video-game-stats#video-game-industry-growth>

Hunt, C. (2002). TCP/IP Network Administration. Third Edition. *O'Reilly Media*. Retrieved 20th of June 2022. Retrieved from:

[https://www.gob.mx/cms/uploads/attachment/file/84434/02.-\\_TCPIP\\_Network\\_Administration.pdf](https://www.gob.mx/cms/uploads/attachment/file/84434/02.-_TCPIP_Network_Administration.pdf)

Huynh, M. & Valarino, F. (2019). An analysis of continuous consistency models in real time peer-to-peer fighting games. Kristianstad, Sweden, Kristianstad University. Retrieved 28th of June 2022. Retrieved from:

<https://www.diva-portal.org/smash/get/diva2:1322881/FULLTEXT01.pdf>

Marino. (2021). Special Attack (Concept). Giant Bomb. Retrieved 30th of June 2022. Retrieved from:

<https://www.giantbomb.com/special-attack/3015-327/#:~:text=Special%20Attacks%20are%20an%20integral,much%20damage%20as%20Super%20Attacks.>

Meyer D. & Zobrist G. (1990). "TCP/IP versus OSI". In *The battle of network standards*, IEEE, p16-19, DOI: 10.1109/45.46812

Orland, K. (2011). Interview: How A Fighting Game Fan Solved Internet Latency Issues.

*Gamasutra*. Retrieved 28th of June 2022. Retrieved from:

[https://gamasutra.com/view/news/34050/Interview\\_How\\_A\\_Fighting\\_Game\\_Fan\\_Solved\\_Internet\\_Latency\\_Issues.php](https://gamasutra.com/view/news/34050/Interview_How_A_Fighting_Game_Fan_Solved_Internet_Latency_Issues.php)

Parish, J. (2009). Fighter's History. *IUP.com*. Retrieved 28th of June 2022. Retrieved from:

<https://archive.ph/20120720141819/http://www.1up.com/features/essential-50-street-fighter-ii>

Pusch R. (2019). Explaining how fighting games use Delay-Based and rollback netcode. *ArsTechnica*. Retrieved 1st of June 2022. Retrieved from:

<https://arstechnica.com/gaming/2019/10/explaining-how-fighting-games-use-delay-based-and-rollback-netcode/5/>

Sasaki, C. (2006). "Street Fighter II Lag | Playfeed".

Retrieved from: <http://www.gearlive.com/games/article/street-fighter-ii-lag08021040/>.

Stallone M. (2019). GDC: 8 Frames in 16ms: Rollback Networking in Mortal Kombat and Injustice 2. [Video]. YouTube.

Retrieved from: <https://www.youtube.com/watch?v=7jb0FOcImdg>

Sun, R. (2019). Game Networking Demystified, Part I: State vs. Input.

Retrieved from: <https://ruoyusun.com/2019/03/28/game-networking-1.html>

Terrano, M. & Bettner, P. (2001). 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. Gamasutra.

Retrieved from:

[https://www.gamasutra.com/view/feature/3094/1500\\_archers\\_on\\_a\\_288\\_network\\_.php](https://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php)

Tseng, P.-H., Wang, N.-C., Lin, R.-M., & Chen, K.-T. (2011). On the battle between lag and online gamers. In 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR). 2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR 2011). IEEE.

DOI: <https://doi.org/10.1109/cqr.2011.5996093>

Unity Technologies. (2022). Script Execution Order Settings. Unity Technologies.

Retrieved from: <https://docs.unity3d.com/Manual/class-MonoManager.html>

## Ludography

Arc System Works. (2017). *Dragon Ball FighterZ* [video game][download][Steam], Bandai Namco.

Arc System Works. (2021). *Guilty Gear Strive* [video game][download][Steam], Arc System Works, Bandai Namco.

Autumn Games, Marvelous Inc. (2012) *Skullgirls*. [video game][PC Version]. Autumn Games.

Bandai Namco Entertainment. (2015). *Tekken 7*. [video game][PC Version], Bandai Namco, Bandai Namco Studios.

Bandai Namco Entertainment. (2018). *Soul Calibur VI*. [video game][PC Version], Bandai Namco, Bandai Namco Studios.

Capcom. (1991). *Street Fighter II*. [video game][Arcade Version], Capcom.

Capcom. (1992). *Street Fighter II Turbo*. [video game][Arcade Version], Capcom.

Capcom. (2015). *Ultra Street Fighter IV*. [video game][Arcade Version], Capcom.

Capcom. (2016). *Street Fighter V*. [video game][PC Version], Dimps, Capcom.

Fornace, D. (2017). *Rivals of Aether*. [video game][PC version], Dan Fornace.

Higinbotham, W. & Dvorak, R. (1958) *Tennis for Two* [video game][Android], Brookhaven National Laboratory.

Nintendo. (2001). *Super Smash Bros. Melee* [video game][Nintendo GameCube], HAL Laboratories, Nintendo.

Nintendo. (2018). *Super Smash Bros. Ultimate* [video game][Nintendo Switch], Sora Ltd, Bandai Namco, Nintendo.

NetherRealm Studios. (2015). *Mortal Kombat X*. [video game][PC Version], Warner Bros Interactive Entertainment.

NetherRealm Studios. (2019). *Mortal Kombat 11*. [video game][PC Version], Warner Bros Interactive Entertainment.

Russell, S et al. (1962). *Spacewar!* [video game][Android], Russel, S.

## **Glossary**

**Active frames:** The frames where an attack can damage an opponent if the attack and an opponent collide.

**Block stun frames:** Amount of frames a character is unable to act for when they block an attack.

**Frame:** The smallest unit of time in video games. This is commonly 1/60 of a second (roughly 16 milliseconds)

**Hit-stun frames:** Amount of frames a character is unable to act for when hit by an attack.

**Netcode:** A colloquial umbrella term that refers to the technical implementation of real-time networking and state synchronization in online video games (Ehlert, 2021).

**Recovery frames:** The frames after an attack necessary for the character to be actionable again. This allows the defending player to respond when the attacker misses or is blocked.

**Startup frames:** The amount of frames required for the attack to be active. This can also be thought of as the anticipation of the attack.

**True Combo:** A fighting game term that refers to a combo that is guaranteed if executed correctly; the defending player cannot do anything to stop it once the first hit has landed.

## Appendix

Collection of Frame Data for various fighting games

<https://dustloop.com/wiki/index.php>

Collection of Game Networking Resources

<https://github.com/MFatihMAR/Game-Networking-Resources>

Data from rollback netcode questionnaire

<https://forms.gle/D7zgyL5ryuNjoA6PA>

Spreadsheet of data from questionnaire

<https://docs.google.com/spreadsheets/d/1cX8mcBAH6jdvxP4TEk-1SIzJCEnPa4DKSkysv6POHxA/edit?resourcekey#gid=698085191>

Database for Online Super Smash Bros. Melee games, and Emulator with Rollback Netcode

<https://slippi.gg>

Frame Data for Super Smash Bros. Melee

<https://meleeframedata.com>

Glossary of fighting games terminology

[https://en.wiktionary.org/wiki/Appendix:Glossary\\_of\\_fighting\\_games](https://en.wiktionary.org/wiki/Appendix:Glossary_of_fighting_games)

Steam database for concurrent players

<https://steamdb.info/graph/?tagid=1743>