Rollback Networking in Peer-To-Peer Video Games

Cameron Cohu

February 26, 2021

Abstract:

Video game networking solutions vary greatly between genres. This paper focuses specifically on the fighting game genre, which is traditionally peer-to-peer. The paper synthesizes literature regarding rollback and delay-based networking solutions, comparing the two and their positives and negatives. It also gives a close look at rollback networking, how it works in a peer-to-peer architecture, and why it is widely considered to produce a superior player experience over delay-based networking.

1. Introduction

The goal of video game networking is to create the illusion that two or more players are sharing a single environment [1]. This is obviously not true, an online game is actually many different simulations playing out simultaneously, but creating the suspension of disbelief is the magic that networking engineers must perform. Not only should these simulations stay in sync, but they must feel responsive and alive. Failing to do so will create frustration at best or an unplayable product at worst, and for no genre is this more true than fighting games.

1.1 A World Disconnected

Fighting games are a genre of games that pit players together in a one-on-one battle. Players control a character, using movement and well-timed inputs to attack the other player. Fighting games are known for rewarding tight timings and combos. Because of their relative difficultly and one-on-one nature, they are considered highly competitive games and are known for holding live, in-person tournaments where players of various skills get together and play. In this sense, fighting games are highly dependent on communities. One can only play if they have someone to play against. In order to understand the importance of this, let's look at a quick case study of two games: *Super Smash Bros. Ultimate* for the Nintendo Switch and *Super Smash Bros. Melee* for the Nintendo GameCube.

In March of 2020, Covid-19 hit the United States and caused nationwide lockdowns. Among the many in-person events that had to be cancelled, fighting game tournaments ceased to exist overnight. Nearly a year has passed since then, and there have still been no in-person tournaments for any fighting game. *Ultimate*, the newest fighting game from Nintendo, has had no significant competition since the lockdowns. Only one national online tournament series was attempted, and it was dropped after its first major tournament due to poor gameplay quality. *Melee*, a 20-year-old game that has been kept alive by a grassroots community, has had multiple major tournaments since the lockdowns, including a 4-week championship league and most recently a charity tournament which raised over $250,000 for Games for Love. These two games are from the same franchise, have similar game mechanics, and comparably sized communities. What could account for the difference in how well these games performed during the lockdowns? *Ultimate* uses a networking solution that is common to the industry called "delay-based networking". *Melee* uses a solution called "rollback networking", and it has made all the difference in the world.

2. Background

2.1 The Game Loop

In order to understand these networking solutions, we first need to understand how fighting games are built. In the typical fighting game, the game loop is played once per frame [2]. The game loop consists of sampling player input, updating the game state, and rendering the results on screen. Fighting games typically run at 60 frames per second, which means that the entire game loop must be acted out in less than 16.66ms. Time in this sense is not continuous but discrete, with the smallest unit of time (as far as the game is concerned) being one frame. This is closely tied to game logic, as everything from animation length, hitbox duration, hit-stun duration, and other critical temporal concepts are all measured in frames. For example, in *Melee*, there is a character named Fox whose forward-tilt attack is measured in 26 frames, with 4 frames of start-up, 4 active frames, and 18 frames of recovery. Because frames are so tied to logic, fighting games are most often developed on consoles where hardware is consistent between players and the amount of frames per second can be more easily controlled.

The game loop is relatively straightforward when considered only locally. All player's inputs, regardless of whether the player is a human or AI, can be sampled at the beginning of the loop and processed in time to render the frame. Things get complicated, however, when considering online multiplayer modes. As stated earlier, the goal of online games is to create the illusion that two or more separate simulations are actually one. Keeping these simulations in sync is no small task. Consider what would happen if the game logic simply played out without any consideration for networking differences. Let's say two players have a latency of 66ms, or 4 frames. On frame 0 player 1 (P1) inputs an attack that activates on its 5th frame, while player 2 (P2) inputs a block that activates on its 2nd frame. P2's block does not reach P1 until frame 4, and therefore doesn't activate until frame 6 (4 frames of latency and 2 frames of start-up). From P1's perspective, their attack activates on frame 5 beating the block on frame 6, and the result is that P1 lands a hit. From P2's perspective, their block activates on frame 2, while P1's attack doesn't activate until frame 9, and as a result P2 blocks P1's attack. What has happened here is called a desync, where the two simulations are now completely different [3]. For a video game, or any shared system, this is an unacceptable scenario that would usually result in aborting the simulation. So what can be done to keep the two simulations from desyncing?

2.2 Delay-Based Networking

Perhaps the simplest solution would be to pause the game until all inputs are collected. This is called "Lockstep" networking, and for games that are not time sensitive this works splendidly [4]. For example, in a turn-based game the turn will not progress until all player's decide on what their move is. Since turn-based games are often strategic and the player is expected to think for a few moments, waiting a half-second or so for latency is not even noticed by the players. However, translating this solution directly into a fighting game, where "turns" are expected to be 16ms, will turn the game into a PowerPoint.

The next step, then, is to add what is called input latency [3]. Input latency adds a set amount of fabricated latency frames to the local player. Let's take our earlier desync example, only now with 5 frames of input latency added. P1 inputs an attack and P2 inputs a block, both on frame 0, but neither will begin until frame 5. Because there are only 4 frames of network latency, by the time frame 5 arrives, both players have received the other's input and both inputs begin simultaneously. Because P2's block is faster than P1's attack, P2 successfully blocks the attack. The game plays out identically in both simulations; the two are in sync.

What was just described is a lockstep model with input latency, or better known as "Delay-Based Networking" [3]. Delay-based networking has been used for decades [5] in video games and is still a staple of the fighting game genre. One of its greatest traits is its simplicity. A developer can implement a delay-based online mode without changing anything about the core game loop [2]. It is also robust and CPU light, which was particularly essential in the early days of online games [6]. Since online modes have become a standard in the video game industry, these appealing characteristics have led to many studios to use delay-based solutions for their games.

There is one more key concept of delay-based networking to cover, and that is variable input latency. The assumption with delay-based networking is that the remote player's inputs will arrive before the input latency is up. In other words, each frame needs inputs from both players to progress. If there is a spike in latency, or a poor connection overall, the game will be forced to pause until it receives the remote player's input. This is a worst-case scenario, so to help mitigate pausing many systems implement variable input latency [3]. In our earlier example, the input latency was set to 5. Rather than keeping the input latency in one constant spot, however, the game detects connection quality and determines an input latency that it feels is appropriate, usually the lowest it can get away with. If there is a spike in network lag, the system will simply increase the input latency rather than pause the game. When the system determines the connection is stable again, the input latency is reduced.

This brings us to the problems with delay-based networking. Fighting games are based on incredibly precise inputs, with many strict timings that are often as small as 6, 3, or even 1 frame(s) (timings with 1-frame windows are said to be "frame perfect"). Delay-based networking, while simple, comes at the cost of what software users desire above all else: responsiveness [6]. Having a fixed input latency causes frequent pauses, and variable input latency changes the timing of inputs on the fly, often leaving high input latency for much longer than is actually needed. As a result, delay-based networking has left in players something to be desired. For years this was simply the reality for anyone wishing to play peer-to-peer games online. But more recently, another method has begun taking the fighting games industry by storm.

3. Rollback Networking

3.1 How Rollback Works

Enter rollback networking. Rollback networking takes a much different approach to keeping the two simulations in sync. For starters, rollback does not require both player's inputs at the beginning of the frame [2]. Rather than waiting for the remote player's input, rollback tries to predict what the remote player will input and render the game as though this were true. When the real input arrives, rollback checks if the prediction it made was correct. If so, then the game continues as though nothing happened. If the inputs differ, then the system will "rollback" the game state to a previous confirmed point, simulate the new inputs, and render the correct game state; all within one frame.

Let's consider the desync example one more time, where network latency is 4 frames. P1 inputs an attack with a 5-frame start-up and P2 inputs a block with a 2-frame start-up, both on frame 0. P1 does not receive P2's input until frame 4, and until then assumes P2 is inputting nothing. When the packet arrives on frame 4, P1's game rolls back to the last confirmed frame, which in this case is frame 0. Knowing the inputs for both players now, P1 simulates the game from frame 0 to the present (frame 4), now knowing that P2 is blocking. The game then renders frame 4 with P2 blocking, and when P1's attack activates on frame 5 P2 successfully blocks it. A similar thing happens on P2's machine. From P2's perspective, P1 does nothing until frame 4. When the attack input is received, the game quickly rolls back and catches up, rendering P1's attack 4 frames in progress. P2 has already been blocking, and when frame 5 arrives the attack is blocked. Again, both simulations are in sync.

3.2 The Pillars of Rollback

Suffice to say, the process of implementing a rollback system is more complicated than its delay-based counterpart. One system to help implementation is GGPO ("Good Game, Peace Out"), a middleware development tool that abstracts some of the rollback architecture [2]. Tony Cannon, creator of GGPO, writes that a game wishing to implement rollback networking must meet three criteria: The game must be deterministic, the game must update the game state independently of input sampling and visual rendering, and the game must be able to save and load the game state on demand.

3.2.1 Determinism

For a game, or any simulation, to be deterministic, it must come to the same outcome given the same inputs and starting state on any machine [7]. This is easier said than done, as there are a number of challenges in making a game deterministic. These include making sure floating point math is done in the same order on every machine every time, or avoiding it altogether. It means making sure the execution order of functions is always the same, that the order of items in any hash or dictionary is guaranteed, and that all random elements are seeded the same, among other things.

While deterministic games are more difficult to make, it is a must for rollback systems because rollback depends on sending only input data and simulating the rest on the fly. The "real" frames for one player and the simulated "rolled back" frames of another must leave the two games completely in sync. Otherwise, even small differences in game state could cause lingering effects that only grow as the match progresses [5]. Deterministic simulations are not only important for rollback, but any online solution that expects to send only input data over the network [7]. This includes most peer-to-peer games, since solutions that involve sending game state data over the network often work more intuitively with a client-server architecture.

3.2.2 Independent Simulation

The ability to update game states without sampling input or rendering graphics is also crucial to rollback networking. As described in the section on the game loop, most fighting games are built such that the game samples input, calculates game state, and then renders graphics in a set order. Separating game state calculation such that it can possibly be performed many times within a single frame is a fundamental change to the game core and is one of the reasons rollback networking is more complex than delay-based, where no core logic needs to be changed. Simulated frames are stripped down versions of the typical fighting game frame, only calculating game state without any visual, audio, or saving included. In his GDC talk "8 Frames in 16ms: Rollback Networking in Mortal Kombat and Injustice 2" [6], Michael Stallone goes in depth about how they struggled to optimize their simulated frames such that they could support the 7 rollback frames they were shooting for.

In addition to running many simulated frames within a single real frame, the game must also be capable of running real frames without input from remote players. To get around this, rollback networking uses a prediction algorithm to predict what the remote player's input will be and checks if it was correct once the real input arrives [2]. The goal of said prediction algorithm is to correctly predict as many inputs as possible. What does such a complicated algorithm look like?

Astonishingly simple, actually. Most prediction algorithms simply assume the remote player is making the same inputs they were on the last confirmed frame. This proves to be incredibly accurate in real life situations. Even in the fastest games, players won't change inputs more than 5 times a second [2]. This means the algorithm would successfully predict 55 of 60 frames, or 92%. And that is a worst-case scenario. Additionally, predicting like this means that rollbacks tend to cut into the startup frames of an option rather than beginning unintended options and undoing them. The latter would be much more visually jarring and take agency away from the player since inputs are usually not valid during attack startups. While the ideal algorithm would obviously predict 100% of the frames successfully, in practice this method is highly reliable and far simpler.

3.2.3 Serialization

The final key criterion for implementing rollback networking is the ability to save and load the game state at any given time. The first step towards accomplishing this is creating a way to efficiently serialize the game state. This process will vary greatly from game to game, but however this is done it is important that from this data the entire game state can be recreated exactly as before. Whenever the game receives input from the remote player, it should save the game state and use this as a checkpoint to rollback to if any predicted inputs within the next few frames are incorrect. Regardless of whether the new inputs cause a rollback or not, the frame they are processed should be serialized as a checkpoint [6]. In other words, the most recent confirmed frame should always be saved. More liberal saving can be used to reduce rollbacks, but this should only be done if you have the extra time, not by default.

3.3 Fixed Input Buffer

There is one more step commonly taken to improve rollback networking, and that using a fixed input buffer. This is similar to what is used in delay-based networking, but in rollback input latency is never adjusted based on connection quality. Three frames is a common choice for this because it covers most latency situations most of the time [6]. Games can afford to set the input latency so low because the consequences of latency spikes are much less severe [2]. In a delay-based system, a fixed latency would result in the game constantly freezing as it waits for inputs. Rollbacks are much more subtle, only leaving visual artifacts that often go totally unnoticed to players. With input latency combined with input prediction, rollbacks will only occur when a latency spike and a change in input overlap, overall greatly reducing the number of networking hiccups and effectively hiding most lag.

4. Implementation Challenges

4.1 Unexpected Difficulties

Needless to say, implementing rollback networking is more complicated than delay-based networking. In addition to building your game around the criteria stated earlier, there are numerous edge cases and oddities to account for. One of these is accounting for game clock synchronization. While developers are hopeful that both machines will run the game at a smooth and steady 60 frames per second, often reality is more complicated. When systems overheat, background processes activate, or particularly hefty frames need to be calculated, simulations can drift out of sync and need to be pushed together quickly to avoid a total desync. For example, GGPO calculates “frame advantage”, an estimate of how many frames behind the local player is compared to the remote player [2]. If the difference between the local and remote player’s frame advantage is greater than 1, GGPO will slow frames down slightly until the two are more in sync again. Similarly, Netherrealm Studios faced a bug where rolling back 7 frames (the maximum they allowed) bogged the game such that its frames took longer, which caused a maximum rollback feedback loop [6]. They also solved this by artificially lengthening the frames of the player who was ahead in order to get the two back in sync. Tony Cannon, creator of GGPO, believes that not solving these clock synchronization issues is the reason that *Street Fighter V*’s online mode performs so poorly [8].

Other problem areas that arise with rollback implementation are visual and audio. By its nature, rollback networking is slightly visually jarring whenever a rollback occurs. However, this can get much more complicated with certain mechanics. For example, GGPO might predict a move that needs to be rolled back. If this move comes with a sound effect it will need to be silenced, preferably in the most subtle way possible [2]. The opposite problem is also true, where a move must skip the first few frames because it wasn’t predicted. The visual portion of the move will need to be rendered at the correct frame, and the audio portion might need to start a few milliseconds into the effect (again, in the least jarring way the developer can manage). These can be solved either by cutting audio effects into frame sized chunks (i.e. 16.6ms) or by keeping a queue of created audio effects and comparing it to the current game state, among other solutions. Stallone mentions in his GDC talk that many of their particles were calculated non-deterministically, and they had to hash and save created particles to avoid recalculating them every frame [6]. He also mentions that certain events should be put off slightly and never rolled back once they activate, such as pausing, scripted sequences, and the end of matches. In an interview with Ricky Pusch, designer Adam Heart suggests that gameplay design choices can also go a long way to mitigate the effects of rollback, such as extending the average startup time in your game by a few frames [3]. While audio and visual problems don't typically affect the game state, smart solutions to them will go a long way in making most rollbacks imperceptible to the average player.

4.2 Player Reception

Given the added complexity needed to implement rollback, one would wonder whether it is worth it to pursue over delay-based solutions. The short answer is yes, absolutely. Firstly, it's important to realize that rollback is not as difficult to implement as it seems; so long as you know you are doing it from the start. Implementing rollback requires many changes to the game core which, as with any software project, can be done much more easily when the system is still on paper and not yet built. The opposite, however, is also true: rollback is very difficult to implement if the game is already close to completion. Many studios have tried to implement rollback in an update and given up [3], and Stallone from Netherrealm estimates it took them 7-8 man years to develop the initial release of rollback (in a patch) [6].

In their case study comparing delay-based and rollback solutions, Martin Huynh and Fernando Valarino had participants interact with delay and rollback solutions under varying controlled latency settings (i.e. 50ms with 5% packet loss) [9]. They found under most latency settings, participants felt more in control of their character and that the game was more responsive using rollback over delay. They also perceived less latency using rollback despite the fact that latency was identical. The differences in user satisfaction were also exaggerated with higher latency. That is to say, users didn't detect much difference between the solutions in perfect conditions but noticed larger differences the worse connection quality became. Most felt that delay networking became frustrating at 100ms latency, while rollback held up better at the same latency.

The beta tests from Netherrealm's initial rollback release would agree with those conclusions [6]. As briefly described earlier, the launch of Netherrealms rollback came with a bug that was frequently causing many players to rollback 7 frames, every frame, in a feedback loop. Despite this bug being present, the beta testers gave the update rave reviews, and 95% agreed it was better than the delay-based solution that was previously in the game. Stallone attributes this to better responsiveness: even under massive latency player inputs are never adjusted and local gameplay feels smooth. Only visual artifacts show evidence of latency, and in a game genre that prizes precision above all else most players seem to think this is a worthwhile trade.

5. Conclusion

Rollback networking is an elegant and efficient way to hide the network latency which will inevitably appear in any online game. It allows for the player to have consistent, reliable feedback, something that in invaluable in the fighting game genre. While it is objectively more complex than delay-based solutions, more and more studios are beginning to adopt it for their online modes.

This is important because fighting games, as stated in the introduction, depend heavily on communities to survive. The ability to play another person in a responsive environment is absolutely necessary for the life of the game, and rollback is the solution that makes this most possible. Rollback enables fighting game communities to bring people together in even the toughest scenarios, and in the end that's what games were always about.

6. References

[1] I. Dubrovin, "Shared world illusion in networked videogames," Bachelor's Thesis, 2018.
[2] T. Cannon, "Fight the Lag! The Trick Behind GGPO's Low-Latency Netcode," *Game Developer Magazine*, Sept. 2012. [Online] Available: https://drive.google.com/file/d/1nRa3cRBQmKj0-SEyrT_1VNOkPOJWNhVI/view
[3] R. Pusch, "Explaining How Fighting Games use Delay-Based and Rollback Netcode," *Arstechnica*, Oct. 18, 2019. Available: https://arstechnica.com/gaming/2019/10/explaining-how-fighting-games-use-delay-based-and-rollback-netcode/
[4] G. Fielder, "What Every Programmer Needs To Know About Game Networking," *Gaffer On Games*, Feb. 24, 2010. Available: https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/

[5] M. Terrano and P. Bettner, “1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond,” *Gamasutra*, Mar. 22, 2001. Available: https://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php
[6] M. Stallone, “8 Frames in 16ms: Rollback Networking in Mortal Kombat and Injustice 2,” Game Developers Conference, 2018. Available: https://www.youtube.com/watch?v=7jb0FOcImdg
[7] R. Sun, “Game Networking Demystified, Part II: Deterministic,” *Ruoyu Sun’s*, Mar. 29, 2019. [Online]. Available: https://ruoyusun.com/2019/03/29/game-networking-2.html
[8] T. Cannon, “GGPO Panel,” EVO, 2017. Available: https://www.youtube.com/watch?t=2038&v=k9JTIn1SVQ4
[9] M. Huynh and F. Valarino, “An analysis of continuous consistency models in real time peer-to-peer fighting games,” Dissertation, 2019.