[**Replication in networked games: Latency (Part 2)**](https://0fps.net/2014/02/17/replication-in-networked-games-latency-part-2/)

Posted on [February 17, 2014](https://0fps.net/2014/02/17/replication-in-networked-games-latency-part-2/)by [mikolalysenko](https://0fps.net/author/mikolalysenko/)

[The last post](https://0fps.wordpress.com/2014/02/10/replication-in-networked-games-overview-part-1/) in this series surveyed replication in network games at a high level. In this article and the next, I want to go deeper into the issues surrounding replication. [One of the most annoying aspects of online gaming is latency](https://www.youtube.com/watch?v=ZcP9jiCbakw). [Latency](http://en.wikipedia.org/wiki/Latency_(engineering)), or lag, is the amount of time between when a user pushes a button and when the state of the game updates. To quantify the effects of lag, we refer to the following experiment by Pantel and Wolf:

L. Pantel, L. Wolf. (2002) “[On the impact of delay in real-time multiplayer games](http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/wolf.pdf)” NOSSDAV 2002

In that experiment, they measured the performance of players in a racing game with varying input delays. Amongst other things they conclude that,

* Latency is negatively correlated with performance and subjective immersion
* Input latency above 500ms is not acceptable
* Below 50ms the effects of latency are imperceptible

And these conclusions are validated [by other experiments](ftp://ftp.cs.wpi.edu/pub/techreports/pdf/04-11.pdf). So given that latency is bad, we come to the topic of this post which is:

*How do we get rid of lag?*

Fighting lag

At first, eliminating lag may seem impossible since there is no way for a client to know what some remote player did until their input has completed a [round trip](http://en.wikipedia.org/wiki/Round-trip_delay_time) over the network. Underlying this is the requirement that all events occur sequentially within a single shared frame of reference.

If we want to beat the round-trip-time limit, then the only solution is to give each player their own local frame of reference. Consequently different players will perceive the same events happening at different times. From a physical perspective this is intuitive. After all, [special relativity](http://en.wikipedia.org/wiki/Special_relativity#Relativity_of_simultaneity) tells us that this is how nature works, and [spatially distributed systems like networked games must obey similar constraints](http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#time-clocks). For example, imagine we have two players and they both shoot a particle.  It could be that player 1 observes their own particle as emitted before player 2’s and vice-versa:

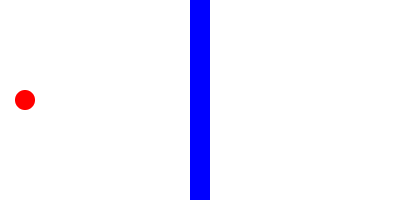
[](https://0fps.files.wordpress.com/2014/02/player1.gif)

Scenario 1: Red player shoots first

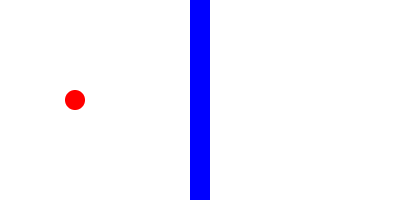
[](https://0fps.files.wordpress.com/2014/02/player2.gif)

Scenario 2: Blue player shoots first

Because the players are not directly interacting, either scenario represents a plausible sequence of events. At the same time though, some events can not be swapped without directly violating the rules of the game.  For example, consider a player opening a door and walking through it; if played in the opposite order, the player would appear to clip through a solid wall:

[](https://0fps.files.wordpress.com/2014/02/consistent.gif)

A plausible ordering of events: a player opens a door and then walks through it

[](https://0fps.files.wordpress.com/2014/02/inconsistent.gif)

An implausible ordering of events: a player walks through a door and then opens it.

While it might not matter in the first scenario who shot first, in the second situation we definitely do not want players ghosting around through doors. It seems like it should be easy to tell the difference, and so we ask the following innocuous sounding question:

*How do we know if a sequence of events is consistent?*

Though this question may sound trivial, in reality it is at the heart of building a concurrent distributed system. In fact, there are multiple answers and the key to eliminating lag lies in subtle details of how one chooses to define *consistency*.

Consistency models for games

A [consistency model](http://static.last.fm/johan/nosql-20090611/intro_nosql.pdf) determines the visibility and apparent ordering of events in a distributed system. [Defining a useful consistency model requires a delicate balance between parallelism and interaction](http://db.cs.berkeley.edu/papers/sigmod13-bolton.pdf). Consistency requirements can be informed by game design and vice versa.  For example, certain features like instant hit weapons or rigid body constraints may be harder to implement in some consistency models than others.

[While](http://en.wikipedia.org/wiki/Weak_consistency) [many](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.3356&rep=rep1&type=pdf) [consistency](http://cs.gmu.edu/cne/modules/dsm/orange/release_con.html) [models](http://cs.gmu.edu/cne/modules/dsm/orange/entry_con.html) [have](https://www.usenix.org/legacy/event/osdi04/tech/full_papers/li_j/li_j.pdf) [been](http://en.wikipedia.org/wiki/Vector-field_consistency) [invented](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.187.8545&rep=rep1&type=pdf) [by](http://cs.gmu.edu/cne/modules/dsm/orange/strict_con.html) [distributed](http://cs.gmu.edu/cne/modules/dsm/orange/pram_con.html) [systems](http://pmg.csail.mit.edu/papers/adya-phd.pdf) [engineers](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.6410&rep=rep1&type=pdf), in the context of games the two most popular are [strict consistency](http://cs.gmu.edu/cne/modules/dsm/orange/strict_con.html) and [optimistic consistency](https://www.cs.purdue.edu/homes/bb/cs542-11Spr/davidson.pdf) (aka [client-side prediction](http://en.wikipedia.org/wiki/Client-side_prediction)). These two methods represent the extremes of the tradeoff space, with the first giving perfect global synchronization for all players and the latter giving the minimum possible latency. What is less well-known (and also what I want to focus on in the next article) are the large and widely underutilized collection of ideas that live somewhere in the middle.

Strict consistency

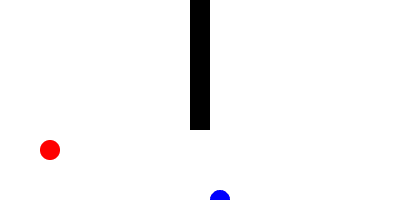
Strict consistency requires that all clients see all events in the same order. It is both the most constrained consistency model and also the easiest to understand. Unfortunately, it also causes lag.  [This consequence is closely related to the famous CAP theorem](http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed), where we recall that the **C**in CAP stands for **C**onsistency in the strong sense (and more specifically [linearizability](http://en.wikipedia.org/wiki/Linearizability)). If we want to beat lag, then this has to be relaxed.

Optimistic consistency

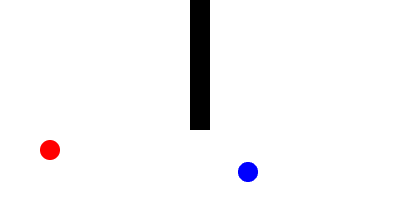
The opposite extreme is to basically toss all consistency requirements out the window and let every client update their model however and whenever they want. This approach is charitably called [optimistic consistency](http://ficus-www.cs.ucla.edu/ficus-members/ashvin/work/optimistic.html) in the distributed systems community or in the game development business [client-side prediction](http://en.wikipedia.org/wiki/Client-side_prediction) or [dead-reckoning](http://en.wikipedia.org/wiki/Dead_reckoning#Dead_reckoning_for_networked_games) (depending on who you talk to). There are plenty of [articles](http://gafferongames.com/game-physics/networked-physics/) [and](http://www.ra.is/unlagged/) [technical](https://developer.valvesoftware.com/wiki/Working_With_Prediction) [documents](http://www.pingz.com/wordpress/wp-content/uploads/2009/11/tribes_networking_model.pdf) describing how client-side prediction is used in games, though the one I most recommend is Gabriel Gambetta’s tutorial:

G. Gambetta. (2014) “[Fast paced multiplayer](http://www.gabrielgambetta.com/fast_paced_multiplayer.html)“

The obvious problem with optimistic consistency is that local replicas diverge. To mitigate these consequences, optimistic replication schemes must implement some form of correction or reconciliation. A general strategy is the [undo/redo](http://infolab.usc.edu/csci599/Fall2008/papers/b-2.pdf) method, which rewinds the state to the point that a particular event was executed and then replays all subsequent events. The problem with undo/redo is that some operations may conflict. To illustrate this, consider the following scenario as viewed by two different players (red and blue). Suppose that the red player fires a laser at the blue player, who tries to dodge behind a barrier.  Then it is possible with optimistic consistency for both players to see different views of events as follows:

[](https://0fps.files.wordpress.com/2014/02/player1hit.gif)

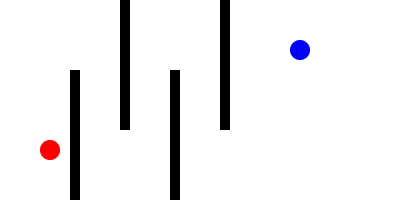
Red player’s perspective: The red player shoots at the blue player and sees the shot hit before the blue player can get to cover.

[](https://0fps.files.wordpress.com/2014/02/player1miss.gif)

Blue player’s perspective: The blue player runs for cover and sees the red player shoot and miss

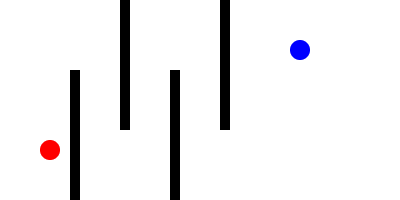
In optimistic consistency, there is no systematically prescribed way to determine which sequence of events is correct. It is up to the programmer to add extra logic to handle all the possibilities on a case-by-case basis. In effect, this is equivalent to defining a weak consistency model implicitly. But because optimistic consistency does not start from any principled assumptions, teasing out an acceptable conflict resolution scheme is more art than science. External factors like game balance greatly inform what sort of actions should have priority in a given scenario.

But the problems with optimistic consistency do not stop at conflict resolution. Unlike databases which operate on discrete objects, games need to present a spatially and temporally continuous environment. Directly applying corrections to the state of the game causes remote objects to teleport. To illustrate this effect, consider a scenario where the blue player watches the red player walk through a serpentine corridor. From the red player’s perspective, the world looks like this:

[](https://0fps.files.wordpress.com/2014/02/walk-nolag.gif)

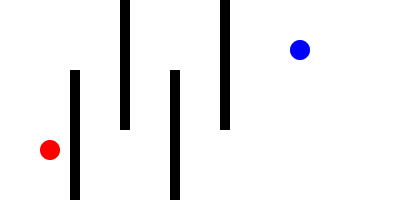
The red player walks through a maze with no lag.

However, if the blue player is using optimistic consistency and resolving remote corrections by directly updating the state, then the rendered view of the red player’s trajectory will skip around:

[](https://0fps.files.wordpress.com/2014/02/walk-lag.gif)

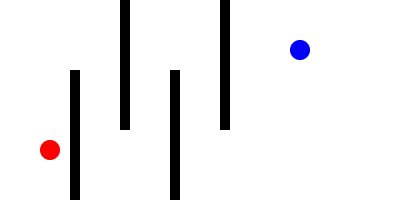
Directly applying corrections to the state causes visible discontinuities in the motion of remote objects. Here the blue player observes the red players motion delayed by 4 frames.

A [commonly proposed solution](http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php) is to smooth out the discontinuities using interpolation (for example [exponential damping](http://en.wikipedia.org/wiki/Damping)). Here is the same trajectory rendered with a damping ratio of 2.5:

[](https://0fps.files.wordpress.com/2014/02/walk-damped.gif)

The remote player’s path is smoothed using a damping ratio of 2.5 to remove discontinuities.

Other popular interpolation strategies include using [splines](http://en.wikipedia.org/wiki/Cubic_Hermite_spline) or path planning to hide errors. Still, interpolation (like conflict resolution) is limited with respect to the latencies and inconsistencies it can hide.  Under a larger delay, damping can cause glitches like players sliding through walls and solid objects:

[](https://0fps.files.wordpress.com/2014/02/walk-severe-lag.gif)

The blue player observes the red player with a lag of 8 frames using the same damping ratio. Larger deviations in the red player’s trajectory are noticeable.

While it is impossible to eliminate correction errors, their severity can be reduced with great programmer effort and conscientious design choices. Choosing optimistic consistency as a networking model also increases the engineering costs of a game engine. At minimum, an optimistically replicated game must maintain at least 3 different subsystems relating to the game logic:

1. First, there is the core game logic which describes how events translate to state updates.
2. Second, there is the conflict resolution code which determines how remote updates are handled.
3. Finally there is the interpolation logic, which must seamlessly hide state corrections from the player.

Each of these systems interact with one another and changes in one must be propagated to the others. This increased coupling slows down testing and makes modifications to the game logic harder.

Local perception filters

Fortunately, there is a third option that is both faster than strict consistency and simpler than optimism. That this is possible in the first place should not be too surprising. For example, [causal consistency](http://db.cs.berkeley.edu/papers/sigmod13-bolton.pdf) gives faster updates than strict consistency while maintaining most of its benefits. However, causal consistency – like most models of consistency in distributed systems – applies to discrete objects, not geometric structures. To apply causal consistency to games, we need to incorporate space itself. One of the pioneering works in this area is Sharkey, Ryan and Robert’s local perception filters:

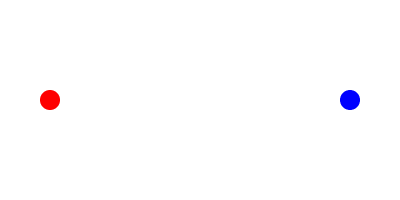
P.M. Sharkey, M.D. Ryan, D.J. Roberts. (1998) “[A local perception filter for distributed virtual environments](http://www.scribd.com/doc/207432762/Sharkey-Et-Al-1998-A-Local-Perception-Filter-for-Distributed-Virtual)” Virtual Reality Annual International Symposium

Local perception filters hide latency by rendering remote objects at an earlier point in time. This time dilation effect spreads out from remote players to all nearby objects. To illustrate this effect, consider a situation again with two players (red and blue) where red is shooting at blue. In this case the red player sees the shot slows down as it approaches the remote player:

[](https://0fps.files.wordpress.com/2014/02/shoot-red.gif)

Red sees the bullet slow down as it approaches blue before speeding up as it passes.

Meanwhile the remote player sees the bullet shot at a higher than normal velocity and then decelerate to a normal speed:

[](https://0fps.files.wordpress.com/2014/02/shoot-blue.gif)

The blue player sees the bullet decelerate

An important property of local perception filters is that they preserve causality, assuming that all interactions are local and that no object travels faster than some fixed velocity c. As a result, they technically satisfy the requirements of causal consistency. This means that there is no need to implement any special correction or client-side prediction models: *local perception filters just work*.

However, they are not without their own draw backs. The locality and speed limit rules out “[railgun](http://quake.wikia.com/wiki/Railgun_(Q2))” like weapons or other instant-hit effects. More subtly, the propagation of rigid constraints violates the c speed limit, and so rigid body dynamics is out too. Finally, while local perception filters can help reduce lag, they do not eliminate it. Standing next to an extremely laggy remote player will slow down your inputs substantially. Some discussion of these consequences can be found in the follow up paper by Ryan and Sharkey:

M.D. Ryan, P.M. Sharkey. (1998) “[Distortion in distributed virtual environments](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.9080&rep=rep1&type=pdf)” Proceedings of Virtual Worlds

Also, unlike prediction based consistency, local perception filters make it easy to implement some fun special effects in multiplayer games. Some interesting examples include [Matrix-style bullet time](http://dl.acm.org/citation.cfm?id=1016551) and [Prince of Persia’s instant rewind](https://www.youtube.com/watch?feature=player_detailpage&v=gu3fiuOLQGE#t=25). Of course it is questionable how practical/fair these effects would be since they necessarily inflict a substantial amount of lag on all player besides the one using rewind/bullet time.

Finally, it is also worth pointing out that the concept of using local time dilation to hide latency appears to have been independently discovered several times.  For example,

X. Qin. (2002) “[Delayed consistency model for distributed interactive systems with real-time continuous media](http://www.eng.auburn.edu/users/xqin/pubs/jos02.pdf)” Journal of Software

Conclusion

In this article we surveyed three different techniques for dealing with latency in networked games, though our review was by no means exhaustive. Also some of these methods are not mutually exclusive. For example, it is possible to combine optimistic replication with local perception filters to offset some of the drawbacks of a purely optimistic approach.

In the end though, selecting a consistency model is about making the right trade-offs. In some situations, maybe the strict input latency demands justify the complexity of and glitches that come with optimistic replication. In other situations where most interactions are long range, perhaps local perception filters are more appropriate. And for slower paced games where latency is not a great concern strict consistency may be sufficient and also easier to implement.

Next time

In the next article, we will talk more about space/time consistency. I’ll also present a more rigorous formalization of local perception filters as a consistency model and prove a few theorems about networking for games. Finally, I’ll write about how to implement local perception filters in a video game.