Dealing with Latency in Networked Games

Asked 16 years, 3 months ago Modified 5 years, 2 months ago

Viewed 11k times

30

I'm thinking about making a networked game. I'm a little new to this, and have already run into a lot of issues trying to put together a good plan for dead reckoning and network latency, so I'd love to see some good literature on the topic. I'll describe the methods I've considered.

Originally, I just sent the player's input to the server, simulated there, and broadcast changes in the game state to all players. This made cheating difficult, but under high latency things were a little difficult to control, since you dont see the results of your own actions immediately.

This GamaSutra article has a solution that saves bandwidth and makes local input appear smooth by simulating on the client as well, but it seems to throw cheat-proofing out the window. Also, I'm not sure what to do when players start manipulating the environment, pushing rocks and the like. These previously neutral objects would temporarily become objects the client needs to send PDUs about, or perhaps multiple players do at once. Whose PDUs would win? When would the objects stop being doubly tracked by each player (to compare with the dead reckoned version)? Heaven forbid

two players engage in a sumo match (e.g. start pushing each other).

This gamedev.net bit shows the gamasutra solution as inadequate, but describes a different method that doesn't really fix my collaborative boulder-pushing example. Most other things I've found are specific to shooters. I'd love to see something more geared toward games that play like SNES Zelda, but with a little more physics / momentum involved.

- Note: I'm not asking about physics simulation here -- other libraries have that covered. Just strategies for making games smooth and reactive despite network latency.

network-programming latency dead-reckoning

Share

Improve this question

Follow

edited Dec 4, 2008 at 19:13

martinatime

2,487 ● 1 ● 17 ● 23

asked Sep 3, 2008 at 20:32

Nick Retallack

19.5k ● 19 ● 94 ● 115

5 Answers

Sorted by: Highest score (default)

21

Check out how Valve does it in the Source Engine: http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

If it's for a first person shooter you'll probably have to delve into some of the topics they mention such as: prediction, compensation, and interpolation.

Share Improve this answer

Follow

answered Sep 3, 2008 at 20:48

Cristian Sanchez

32k ● 11 ● 58 ● 63

1 This works for simple shooters (as CS and Quake) without any advanced physics where the server can do all client movements in advance. If you introduce cars in the game for instance, this algorithm becomes unfeasible due to the overhead re-simulation of all possible interacting body "islands". – Jonas Byström Aug 11, 2012 at 10:31

Btw. also note that "pushing rocks" (example from the question) becomes a horrible exercise if two players engage in such a frivolous action simultaneously. – Jonas Byström Aug 11, 2012 at 10:44

11

I find this network physics blog post by Glenn Fiedler, and even more so the response/discussion below it, awesome. It is quite lengthy, but worth-while.

In summary

Server cannot keep up with reiterating simulation whenever client input is received in a modern game physics simulation (i.e. vehicles or rigid body dynamics). Therefore the server orders all clients latency+jitter (time) ahead of server so that all incomming packets come in JIT before the server needs 'em.

He also gives an outline of how to handle the type of ownership you are asking for. The slides he showed on GDC are awesome!

On cheating

Mr Fiedler himself (and others) state that this algorithm suffers from not being very cheat-proof. This is not true. This algorithm is no less easy or hard to exploit than traditional client/server prediction (see article regarding traditional client/server prediction in @CD Sanchez' answer).

To be absolutely clear: the server is not easier to cheat simply because it receives network physical positioning just in time (rather than x milliseconds late as in traditional prediction). The clients are not affected at all, since they all receive the positional information of their opponents with the exact same latency as in traditional prediction.

No matter which algorithm you pick, you may want to add cheat-protection if you're releasing a major title. If you are, I suggest adding encryption against stooge bots (for

instance an XOR stream cipher where the "keystream is generated by a pseudo-random number generator") and simple sanity checks against cracks. Some developers also implement algorithms to check that the binaries are intact (to reduce risk of cracking) or to ensure that the user isn't running a debugger (to reduce risk of a crack being developed), but those are more debatable.

If you're just making a smaller indie game, that may only be played by some few thousand players, don't bother implementing any anti-cheat algorithms until 1) you need them; or 2) the user base grows.

Share Improve this answer

Follow

edited Oct 17, 2019 at 17:10

Vaillancourt

1,408 ● 1 ● 12 ● 42

answered May 7, 2009 at 15:23

Jonas Byström

26.1k ● 23 ● 104 ● 152

I think this is a great recommendation. Certainly material that it takes a while to crunch through but very worth while. On the down side I think it's worth noting that that solution is not very cheat proof, in fact it was not a concern for the type of cooperative gameplay they were developing this for I believe. – Chris May 7, 2009 at 15:38

He mentioned this, but it is really not any less cheat-proof (client/server) than traditional prediction, it just requires that the server performs result calculation and sanity checks - all clients are ahead of server. – Jonas Byström Jun 15, 2009 at 16:52

2

we have implemented a multiplayer snake game based on a mandatory server and remote players that make predictions. Every 150ms (in most cases) the server sends back a message containing all the consolidated movements sent by each remote player. If remote client movements arrive late to the server, he discards them. The client the will replay last movement.

Share Improve this answer

Follow

edited Dec 16, 2015 at 23:05

Ted Bigham

4,333 ● 1 ● 28 ● 34

answered Mar 3, 2015 at 17:50

drusilabs

21 ● 1

This looks like a statement about what you did its better to convert it into an specific answer for this question – Ram Mar 3, 2015 at 18:15

I was not my intention. Actually i don't think there is a best solution. Latency issues are difficult to deal with and varies depending on the scenario. I just tried to explain my experience. – drusilabs Mar 3, 2015 at 18:51

0

Check out Networking education topics at the XNA Creator's Club website. It delves into topics such as network architecture (peer to peer or client/server), Network Prediction, and a few other things (in the context of XNA of course). This may help you find the answers you're looking for.

http://creators.xna.com/education/catalog/?contenttype=0&devarea=19&sort=1

Share Improve this answer

Follow

answered Sep 16, 2008 at 16:48

Joel Martinez

47.7k • 26 • 134 • 185

0

You could try imposing latency to all your clients, depending on the average latency in the area. That way the client can try to work around the latency issues and it will feel similar for most players.

I'm of course not suggesting that you force a 500ms delay on everyone, but people with 50ms can be fine with 150 (extra 100ms added) in order for the gameplay to appear smoother.

In a nutshell; if you have 3 players:

- John: 30ms
- Paul: 150ms
- Amy: 80ms

After calculations, instead of sending the data back to the clients all at the same time, you account for their latency and start sending to Paul and Amy before John, for example.

But this approach is not viable in extreme latency situations where dialup connections or wireless users could really mess it up for everybody. But it's an idea.

Share Improve this answer

Follow

answered May 7, 2009 at 15:32

Andrioid

3,390 ● 4 ● 28 ● 32

2 This can be exploited. A user could modify their client to wait a bit before volleying when you try to detect their latency. If you then sent them packets early, they'd get them before the other players. – Nick Retallack Jun 3, 2009 at 6:18

Latency is the least of my problems if my users are modifying their own clients. However it wouldn't matter; if you say that you have 500ms and the server assumes you have a latency of 500ms and you respond within 50ms. Your response will get delayed according to the delay to synchronize all the clients. Of course, this depends on how the latency settings are implemented. If I was modifying my client, I'd focus on automating gameplay or other exploits. – Andrioid Jun 3, 2009 at 10:05