GDC Session Summary: Halo networking

March 11, 2011

This is a summary of a GDC talk given by a senior networking engineer at Bungie, David Aldridge. In this talk he explained the core ideas behind the netcode in Halo 3 and how it was improved for Halo: Reach. This talk was two hours long, so my summary is quite long as well.

The talk started with a practical definition for netcode: "Technology that helps players sustain the belief that they are playing a game together." This definition acknowledges the fact that every player is actually playing a slightly different game (due to network latency and staggered state update information), but Halo needs to support the illusion of a shared world as much as it can.

To achieve this illusion, the Halo team had four goals for the netcode:

Scalability: Network games must work on low-end connections, even with 16 players and dozens of vehicles and active objects.
Reliability: The game must be fair and consistent despite packet loss and other connection problems
Security: All important information must be under the strict control of the server, so no clients can cheat
Speed: Players should perceive as little lag as possible

Scalability

To make the game scale well from detailed 1v1 fights to 8v8 vehicle-based skirmishes, the team prioritizes what is important to each client. The priority of each object determines how often its state information is sent to the client. For example, if the client is shooting at a target, he cares very much about the target's position, so it will be updated as frequently as possible. However, if a ragdoll is rolling down a hill behind him, he doesn't care about that much at all, so it will update much less often.

guarantee holds up with any amount of bandwidth or packet loss.

Cosmetic events like bullet hits and special effects are handled differently from state information. They are prioritized even more strictly, and are only sent once, so there is no guarantee that they will arrive at all. This means that a low-bandwidth player in a very complicated game may miss a lot of the special effects, but the game itself should still work well. If the same player played in a less complicated game, then a lot more of the special effects events would have space to get through. These two features, prioritization and separation of state and events, allow Halo networking to scale effectively to different network connections and game sizes.

Reliability

Network reliability depends on the underlying network protocol, such as UDP and TCP. UDP just sends packets of data as fast as possible with no reliability guarantees, while TCP takes some extra time to guarantee that all packets arrive and stay in the same order. For Halo they chose to build their own protocol on top of UDP, based on the protocol used in Starsiege Tribes (described in this paper).

This protocol allowed them to mix and match reliability guarantees in a flexible way. For example, some packets need guaranteed arrival (the game is over, client requests grenade throw), some packets only need guaranteed ordering (player 3 is at this position, now player 3 is at this position), and some packets need no guarantees at all (sparks spawned at this location, bullet hole appears here). By using guaranteed packets for the critical gameplay notifications, they kept the gameplay reliable without wasting bandwidth on state update acknowledgements or resending cosmetic events.

Security

Cheating can be a big problem with online games, especially major e-sport titles like Starcraft or Halo. There are many ways to cheat, ranging from simple cheats like altering the host internet connection to complex tasks like modding the game itself.

Halo achieves a base level of security by keeping most major decisions on the host system. For example, if a player presses the 'throw grenade' button, then the client asks the host for the server.

Unfortunately, this method can't stop aimbots, nor does it prevent the host itself from cheating. It sounds like Bungie mostly addresses that with a team dedicated to reviewing cheat-flagged replays and judiciously applying the banhammer.

Speed

Minimizing lag starts by optimizing bandwidth usage using profiling tools, making sure the game sends as little data as possible. In Halo: Reach, they added network profiling information to all replays, so they could look at any replay and view the packets sent to every player at every frame. Using this information they identified bottlenecks, allowing them to reduce bandwidth usage to 20% of what they used for Halo 3.

However, there's no optimization that can change the speed of light. If the round trip time from the client to the host is 100 ms, then everything will have a 1/10 second delay, and there's nothing anyone can do to change that. However, there are many techniques to hide this lag from the players, which are all based on prediction.

Prediction is the technique of changing client state *before* receiving authentication from the server. A simple example of prediction is shooting a gun. When the game receives input that the player has pulled the trigger, it *immediately* plays the muzzle flash effect and the gunshot sound. The client still has to wait for host authentication to actually damage anything, but the immediate cosmetic feedback makes it feel responsive. All direct player control is predicted in this way, including aiming and movement.

A more complicated example is the grenade throw. In early versions of the game, when the player pressed the "throw grenade" button, the client would wait for host permission before playing the throw animation, making the throw feel unresponsive. To address this, they made the button press immediately predict the throw animation. However, the client still had to wait for host authentication before actually displaying the grenade object itself. Fortunately, the throw animation was distracting enough that players did not actually perceive the lag there.

else. Mr. Aldridge said that the way they handled this in Halo could fill up another entire talk, but the short version is that it sends target-relative gunshot information rather than world-relative. For example, if your crosshair is over someone's head, the game tells the server "I shot directly at the head of this player", and the server just checks if there is a clean shot from your gun to the player's head.

In Halo: Reach testing, the players had a dedicated "lag" button they would press whenever anything seemed laggy, so the developers could get real data about what lag was perceived. Whenever an event was receiving a lot of lag reports, the network engineers tried different ways to hide the lag, until the lag reports stopped. When most of the lag reports synced up precisely with testers losing games, they figured they had done enough.

Conclusion

The talk ended with a series of lists:

Four rules of gameplay networking:

Decide what needs host authentication and what can be predicted
Always ask, "Where am I hiding the lag? Can players perceive it there?"
Don't be afraid to change game mechanics to improve networkingReserve time to iterate

Three rules of network optimization

Measure twice, cut once (use profiling tools)
Look at the big picture, compression can only take you so far
Change gameplay to require less networking (e.g. don't network ragdolls)

Best practices

Use flow and congestion control (low-priority never starves high-priority)
Record connection quality and choose quality hosts when matchmaking
Use host migration to keep a game going if the host drops
Test on a large scale using a multiplayer beta or demo
Perform regular internal playtests with traffic shaping
Have full-time network game testers (e.g. with "report lag" buttons)

This talk made me want to try out networking on Overgrowth, even though the single-player is still priority one. The major challenge for player-vs-player networking would be hiding the latency in close range combat, because reversals and counters depend on split-second timing. There are a few points that this speaker mentioned that could help with that: first, we could change the game mechanics slightly to allow for more time to sync. For example, we could add brief dramatic pauses after important impacts or dodges.

We might not even have player-vs-player multiplayer in Overgrowth, but if we do, I would like to make sure it's as lag-free as possible. Do you have any ideas to hide the latency in melee combat, or any links to information about how other fighting games address this challenge?

Posted in Game Tech by David Rosen

OVERGROWTH ALPHA 121 ART ASSET OVERVIEW #8

EMAIL UPDATES:

Email address

SIGN UP

© 2025 Wolfire Games

Home

Blog

Games

Contact

Privacy

Site designed by Fully Illustrated