Quake 3 Networking Primer

This section gives a brief overview of what goes on between the client and the server. It's not complete, and it may not be entirely accurate, since most of the information I learned second-hand, or inferred from the game source and from extensive testing. Still, it should be adequate. It certainly is consistent with all of my observations and tests so far.

It's also very necessary to understand if you want to implement any kind of lag compensation at all. Many mod-authoring readers will need to adjust the Unlagged code to fit their games. That will be considerably easier if they have a solid understanding of what's going on and how it works.

Terminology

In order to make sure what I write (and you read) brings to your mind images similar to what I imagine when I'm writing about them, we'll have to go through some terminology. Hopefully it's not too dry.

Snapshot: a chunk of data sent from the server to the client that represents what's going on on the server, at the end of every server frame. (The term's reference to photography is very apt.) The client's representation of this data (in models or polygons rendered, and sounds played) is the starting point for how the player perceives the game world.

Since the client game runs at a different rate than the server (usually at a faster rate), it cannot always represent this data exactly as it receives it. It must *interpolate* or *extrapolate*.

A snapshot may contain (but is not limited to): positions (origins) and velocities of other players, origins and velocities of non-player entities such as rockets, sounds to be played and effects to be drawn (events), scoreboard information, and custom client commands.

We'll not concern ourselves with exactly what is included in a snapshot (or excluded according to the client's potentially visible set) or any compression techniques. Our biggest concern is with what player origins are sent.

Command: One command is sent from the client to the server every client frame. A command consists of the following data:

- a timestamp: the client clock time that the command took place (corresponds with cg.time in the client game code)
- the player's view angles (Euler coordinates in pitch-yaw-roll order) in fixed-point format
- a bit field that describes which buttons are currently being held down
- a weapon number
- the player's desired forward, right, and up movement

Aside from custom commands such as "quit," this is all the server uses to react to player input.

Interpolate: to estimate a value between two known values. Generally, the Quake 3 client game determines other player origins by interpolating between two known origins, which it obtains from two corresponding snapshots.

Extrapolate: to estimate a value beyond one known value, using additional data. For example, rocket origins are extrapolated using a base origin, a velocity, and a time delta.

Prediction: when the client or server takes a guess at how something in the future will happen. The kinds of prediction we talk about are of course special cases of extrapolation. And normally, when we talk about Quake 3 prediction, we refer only to the client game predicting where its own player will be by the time the client's commands reach the server. (Unlagged has other uses for prediction, such as smoothing out players who would otherwise be skipping around.)

Prediction error: when the prediction is wrong. It must be corrected for, which introduces some of the most severe visual problems when it happens on the client – depending on how far ahead the client was predicting.

Frame: a regularly-scheduled (or semi-regularly-scheduled) period of time in which the server or client gets things done. (In case you were wondering, that's about as specific a definition as we can get.)

Process

There's no need to describe every tiny detail of the process. Here's a rough sketch of what goes on on the server, though:

1. The server accepts client commands as they arrive. It makes one call to the VM per client command. This is the only time that players are ever updated. (That's why, when someone has a bad connection from their client to the server, they'll freeze in place or skip.) (See ClientThink() in g_active.c)
2. At 50ms intervals (if sv_fps is 20), a VM call is made to run a frame. All non-player objects (and bots) are advanced at this time. (See G_RunFrame() in g_main.c)
3. After the VM call to advance non-player objects, the server prepares and sends a snapshot to every client.
4. Repeat.

On the client:

1. The client engine makes a VM call to draw the active frame. (See CG_DrawActiveFrame() in cg_view.c)

2. Almost immediately, the client game checks for a new snapshot. If there's a new one, it sets cg.snap to the old one and cg.nextSnap to the new one. If there's not one at all, it sets cg.snap to the old one and cg.nextSnap to NULL. (See CG_ProcessSnapshots() in cg_snapshot.c)
3. The client game processes new commands and predicts its own client's state. Barring prediction error, this is the state the client will be in by the time its commands reach the server and are processed. (See CG_PredictPlayerState() in cg_predict.c)
4. All entities in the old snapshot (cg.snap) are processed. This involves interpolating or extrapolating states, firing events, and adding audio and visual representation. (See CG_AddCEntity() in cg_ents.c)
5. More processing is done that doesn't apply to lag compensation, and the frame is actually rendered.
6. After the VM call to draw the active frame, the client engine processes input and sends a command to the server. This command will be processed client-side on step #3 in the next iteration.
7. Repeat.

There's obviously a lot more going on than what's listed, but I've listed what we're concerned with for dealing with the many faces of “lag.”

Next: A Solution