MARCH 9TH, 2009

QUAKE ENGINE CODE REVIEW : PREDICTION (3/4)

We just saw the NetChannel abstraction for network communication. Now are are going to see how lacenty was compensated via Predictions. Here is some additional material:

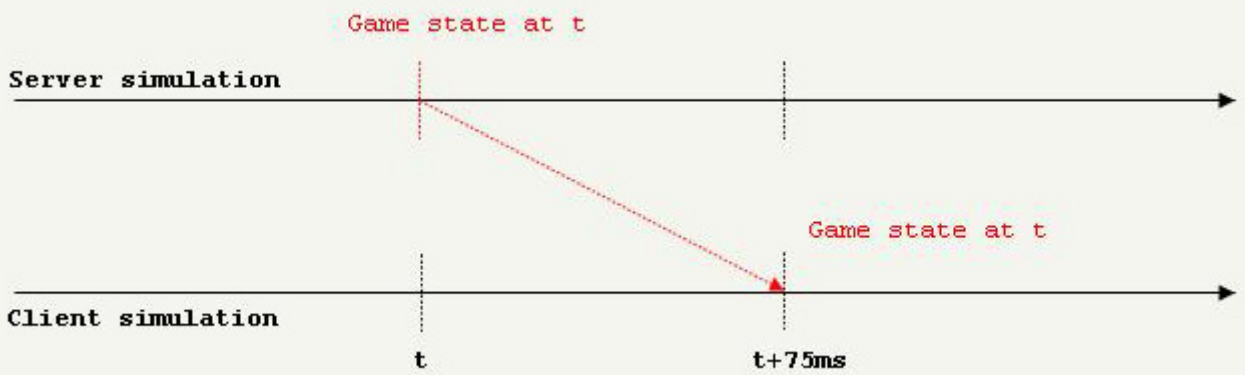
- An article by John Carmack himself.
- An other article (archive) by Valve describing Half-life engine (Half-life uses Quake engine).

This article is in four parts :

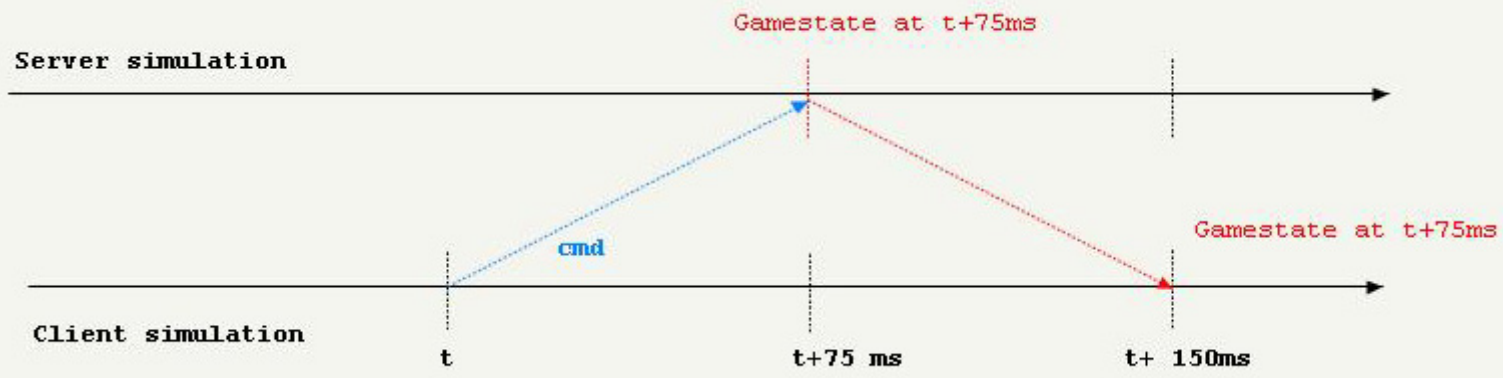
Prediction

Prediction is probably the hardest, the least documented and the most important piece of Quake World Engine. The goal of the prediction is to beat latency, namely compensate the delay it takes for the medium to transmit informations. This is done on client side, the process is called "Client Side Prediction", there is no Lag Compensation technique on server side.

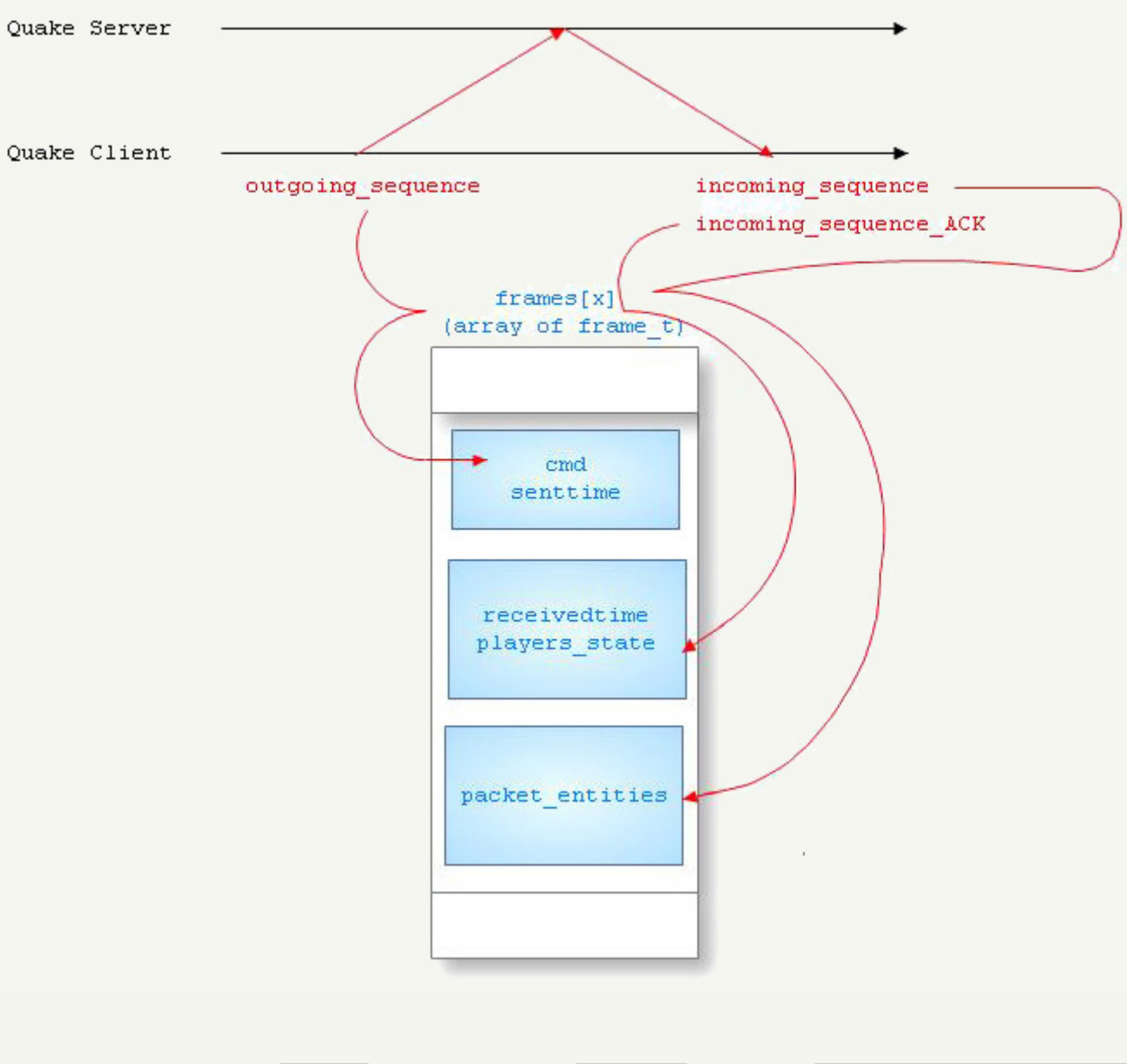
The issue:



So gamestate is latency/2 old. If we include the time to sent the command, we have to wait a round trip (latency) to see the result of our actions:



The key to understand Quake prediction system is to understand how NetChannel populate the "`frames`" variable (an array of `frame_t`).



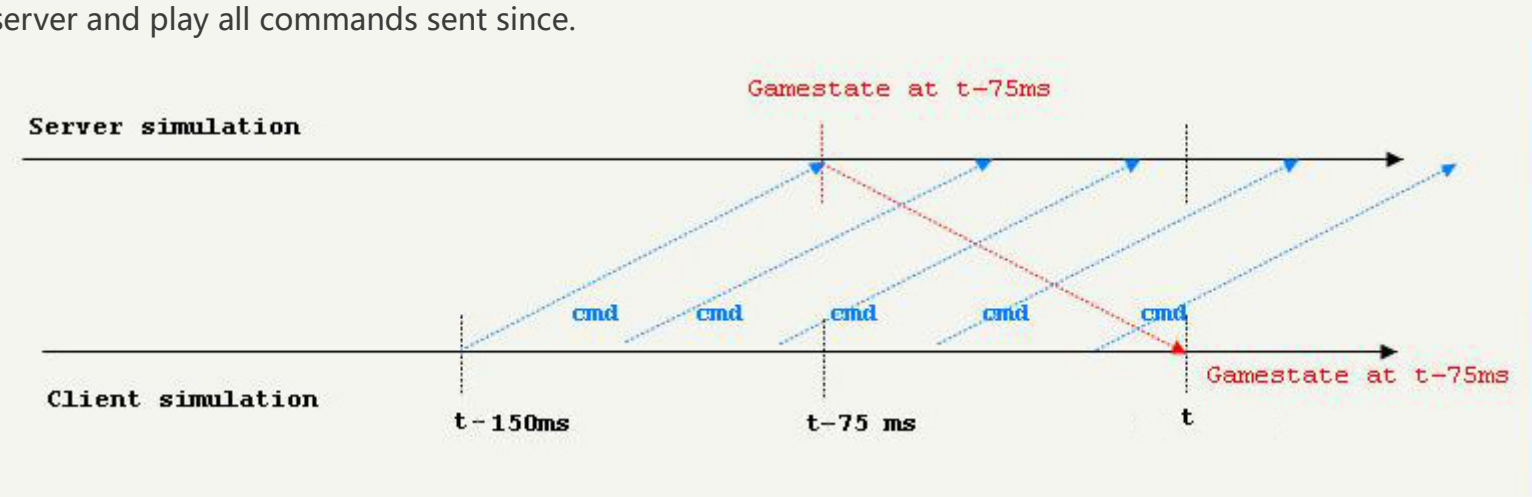
Every command sent to the server is saved in `frames`, along with the `senttime`, at index `netchannel.outgoingsequence`.
When the server acknowledge the reception of the command via `sequenceACK`, we can retrieve the sent command and calculate latency:

```
latency = senttime-receivedtime;
```

At this point, we have the world the way it was a latency/2 ago. On a NAT latency is fine (<50ms) but on Interet, it's huge (>200ms) and prediction have to be done to simulate the world right now. This is done differently for local player and other players

Local player

For local player, latency is pretty much reduced to 0, by extrapolating what will be the server state. This is done by using the last received state from the server and play all commands sent since.



The client hence predict what will be its position on the server at t+latency/2.

From a code perspective, this is done in the `CL_PredictMove` method, first Quake engine decide the sentime limit for playable commands:

```
cl.time = realtime - cls.latency - cl_pushlatency.value*0.001;
```

Note: `cl_pushlatency` is a cvar set on client side, equals to minus the client latency in milliseconds. Hence we can pretty much conclude: `cl.time = realtime`.

Then every other players are turned solid via `CL_SetSolidPlayers (cl.playernum);` (so collision can be tested) and commands sent since the last received state are played until: `cl.time <= to->senttime` (collision are tested each iteration via `CL_PredictUsercmd`).

Other players

For other players, Quake engine doesn't have the "commands-sent-but-not-yet-acknowledged" so extrapolation is used instead. Starting from the last known position, `cmd` are extended to predict used position. Angle rotation are not predicted, only position.

Quake World also takes into account other player's latency. Each client's latency is sent along with worldupdate.

Code

The prediction and collision code can be summarized as follow:

```
CL_SetUpPlayerPrediction(false)
CL_PredictMove
|       /* Local player is moved */
|       CL_SetSolidPlayers
|       |       CL_PredictUsercmd
|       |               PlayerMove
|       Interpolate linearely
CL_SetUpPlayerPrediction(true)
CL_EmitEntities
        CL_LinkPlayers
        |       /* Other players is moved */
        |       for every players
        |       |       CL_SetSolidPlayers
        |       |       CL_PredictUsercmd
        |       |               PlayerMove
        CL_LinkPacketEntities
        CL_LinkProjectiles
        CL_UpdateTEnts
```

This part is complicated because not only Quake Work perform predictions on players but also it has to perform collision detection on predictions.

`CL_SetUpPlayerPrediction(false)`

The first call do not perform any prediction, it only setup other players as they were received from server (so in the past at t-latency/2).

`CL_PredictMove()`

This is where the local player movement is performed:

- Orientation is not interpolated and is full realtime.
- Position and velocity: all commands sent until now (`cl.time <= to->senttime`) are applied to the last position/velocity received from the server.

More on Position and velocity update:

- Other players are first turned solid (in there last know position, set in `CL_SetUpPlayerPrediction(false)`) via `CL_SetSolidPlayers`.
- Engine loops against sent commands, checking for collision and predicting the position via `CL_PredictUsercmd`, collision against other players are also tested.
- Resulting position and velocity are stored in `cl.sim*`, this will be used later for POV setup.

`CL_SetUpPlayerPrediction(true)`

In the second call, other players position are predicted at the current time on server side(but no movement is performed yet). Position is extrapolated with the last known commands and last know position.

Note: There is a bit of a problem here: Valve recommendations (for `cl_pushlatency`), end up having local player predicted at t+latency/2 on server side. However, other players position is predicted at t on server side. Maybe `cl_pushlatency` best setting for QW was -latency/2 ?

`CL_EmitEntities`

Here is where visibiliy edicts are generated. They will feed the rendered.

- **CL_LinkPlayers :** Other players movment is performed, other players are turned solid in turn and collision detection is performed against their predicted position.
- **CL_LinkPacketEntitiesPacket :** entities from last state received from server are predicted and linked to visibility edicts. This is why the missile you fire is lagged.
- **CL_LinkProjectiles :** Nails and stuff
- **CL_UpdateTEnts :**Standard Light beams and entities update.

Return to main Quake Source Exploration page.