Quake: An Example Multi-User Network Application — Problems and Solutions in Distributed Interactive Simulations

Shawn Bonham, Daniel Grossman, William Portnoy, & Kenneth Tam*
University of Washington Seattle, WA 98195

May 31, 2000

Abstract

Research efforts into the engineering of real-time, distributed, virtual environments for use over the Internet have flourished since DARPA's first steps in this direction fifteen years ago. The video game "Quake" (from idSoftware), having been the first successful *game* in this genre, has seen its interface and network usage characteristics become something of a standard in the network entertainment world. As Quake-like games which promise more detailed character skins, richer environment textures, and "realer" physics are released in rapid succession, such applications are quickly becoming the most data-intensive, latency-abhorring, and generally demanding distributed programs running on today's networks.

With this in mind, we have chosen the freely available "Quake I" and "QuakeWorld" sources as the subjects for a series of experiments to determine what transport and network layer strategies result in the best game play for the user, and why. In modifying these sources, we have attempted to define the term "acceptable gameplay" in networked first person shooters, and obtained quantitative data capturing this notion. Furthermore, we have quantitatively demonstrated that the later QuakeWorld code exhibits, as predicted, better network performance than the earlier Quake I code.

Keywords: Distributed Interactive Simulation, Real-Time, Multi-User Internet Applications

1 Introduction

Today's Internet has witnessed the proliferation of multi-user network applications in a variety of forms: from "Virtual WhiteBoard" collaboration and business teleconferences to educational multicasts and entertainment-oriented virtual realities. As expected, the session quality in these interactions varies from one participant to another according to their particular mix of Internet and PC performance parameters and the performance demands of the program at hand. Unfortunately, some of the most useful

*Graduate Students, Department of Computer Science and Engineering

such programs — the ones hinging on real-time, multi-user, and audio/video-intensive environments — consume the most resources, and certain users simply do not have the processor power or local network speed needed to participate.

Nonetheless, a flourishing research goal of recent years has been to provide every *viable* user in such Distributed Interactive Simulations (DISs) with some level of "good" performance, whether that be in terms of quick network response to user input, low transmission error rates, or one of many other desired efficiencies.

1.1 Recent Work

The last twenty years have given rise to numerous research efforts in distributed interactive simulations, starting with DARPA's work on SIMNET between 1983 and 1990. In partnership with the U.S. Army, DARPA created a massive graphical simulator for thousands of users (or artificial users) to interact on a virtual battlefield extending over hundreds of virtual square miles. SIMNET was designed to incorporate possibly crucial information from every action of every independent entity in the simulation. SIMNET's well-tuned communication protocols therefore became the basis for the DIS (Distributed Interactive Simulation) Standards developed by the IEEE in the early 1990's — standards which insist on scalable, reliable communication over heterogeneous networks. [6]

Encouraged by SIMNET's technological advances and IEEE's standards, companies set to work conjuring up marketable applications based on DARPA's techniques. As more and more firms got on the bandwagon, software and hardware were developed for a variety of uses such as one-to-many streaming broadcast and many-to-many, low-content multicast (e.g. WhiteBoard). These efforts were, of course, sidestepping the truly difficult problems of high-content, many-to-many, **real-time** simulation, but due in part to DARPA's published results, the networks community soon realized that real-time simulations would prove a useful and lucrative arena for research and development.

1.2 Quake

And so when idSoftware released its interactive, real-time, multiplayer game "Quake" (early in 1996), it broke ground in the entertainment world as the first game designed specifically for real-time play over the Internet. Within the spontaneous Quake networks that were soon to be ubiquitous in the Internet are found nearly every imaginable performance hurdle for DIS: multi-packet datagrams sent from dozens of clients to the game server, which must aggregate these and, in turn, broadcast the entire state of the game arena and all of its players *back* to every participant; widely varying latencies; horribly restricted bandwidths; overloaded router queues; and, not uncommonly, intercontinental distances between the clients and the server. Despite these considerable impediments to achieving the virtual reality desired by its players, the series of Quake games has excelled in its tasks, and they made perfect examples not only to the rest of the computer gaming industry, but also to the networking industry at large, of how to build the incredible variety of distributed, multi-user applications which would soon be demanded by customers from industry to academia.

2 Quake and QuakeWorld

Moreover, simply observing idSoftware's recorded mistakes and brilliant recoveries can be provide a deep, worthwhile look into the inner workings of the today's Internet.

2.1 Why Quake and QuakeWorld?

Quake became one of the most popular games of all time, but more importantly, it brought the experience of an Internet multiplayer game to the masses (having more impact than games such as Netrek [1]). Though Quake had some problems with Internet play over a high latency connection such as a modem, it was very popular. The ability of QuakeWorld to mask some of the effects of high latency connections brought even more players into the fold.

Later games by idSoftware such as Quake II and Quake III built on the knowledge and experience gained from the first two releases, but the improvements to Internet play in these latter games were not as dramatic as those between Quake and QuakeWorld. Other games have struggled with the Internet multiplayer problem [4], but few games have been as successful as the Quake series.

Quake creator John Carmack's 1999 "Christmas present to the Internet" — the free distribution of the source codes to Quake and QuakeWorld after Quake III had been shipped [8] — was thus also a godsend to the academic Internet application researcher.

Given access to the source code for such a high-profile, successful Internet multiplayer game as Quake — and to the code for the much-improved QuakeWorld — we and others have chosen to examine these two programs as case studies in, first, mediocre and, second, quite good distributed, Internet-based interactive simulations.

2.2 Quake

How exactly does Quake interact with its networks? Let us briefly present a generalized description of the Quake network loop. Servers for Quake send periodic updates for the positions of each character in the game to each player. The clients, in turn, send move request messages to the authoritative server whenever the player on that client commands movement via the mouse or keyboard. The server then, once more, sends the newly-updated positions back to all of the clients. One important note is that, in the original version of Quake, the player's computer does not render a player's movement until receipt of the server echo. Even so, this scheme works nicely, assuming prompt and reliable delivery of packets.

However, in short, packet delivery in the Internet has some flaws. Since Quake was designed to support multiplayer games across a variety of interconnected networks, in order to operate at all, numerous design decisions/tradeoffs were made. We discuss several below.

2.2.1 Client/Server Topology

Though not originally a main focus of discussions on the details of Internet Quake play [2], the topology of the network connections between players and server has a huge impact on gameplay quality. One of two

typical communication paradigms is used in contemporary network games: *peer-to-peer* or *client-server*.

Peer-to-peer games appear to be the easiest to use in an Internet game architecture. Basically, each player's computer runs its own simulation of the game, with every peer in the network collecting all of its player inputs and distributing them to all of the other players. Upon receipt of each timepoint's data, all players' inputs are applied identically to the simulations running in parallel on each player's computer.

There are two clear disadvantages to this approach. First, each peer is run in lockstep with the others (to insure identical simulation state on each peer), meaning that lost network messages will freeze a game until every peer receives every message. Early multiplayer games (like Doom, also by idSoftware) used precisely this peer-to-peer approach, limiting the range of play to local LANs. Second, every peer needs to send input messages to, and receive input messages from, every other peer. This *explosion* of input messages at each peer proved grossly inadequate for the low bandwidth modem connections of five years ago (and even for the faster connections of this century).

Quake, on the other hand, uses a client-server model for its network topology. The *authoritative* server is the only computer that receives input messages from the clients, and it is thus the only computer responsible for simulating the game world. If input messages are dropped on the way from a client to the server, then only that client's gameplay quality suffers. Likewise, if position update messages are dropped on the way from the server to some client, the only that client's gameplay is impaired. In addition to foregoing the restrictions of lock step simulation, only the server need have great enough bandwidth to receive and send game state information to every client.

2.2.2 UDP

IP networking provides *best-effort delivery* of packets across the Internet, whereas the TCP transport layer makes delivery reliable. This reliability, however, introduces latencies into the data stream due to missed packets that must be resent or to packets received out of order. But for real-time games such as Quake, which (generally) pass homogeneous and independent packets between clients and server, TCP reliability is not at all useful, since resent or disordered packets that arrive even a few seconds too late are carrying entirely obsolete data.

Thus, one of Quake's most important optimizations for Internet play is the use of UDP as its main data stream protocol. Position packets consist of absolute (x,y,z) coordinates, rather than coordinates relative to previous frames', allowing character position in the arena to be correctly determined even if earlier packets have been dropped. In addition, time-stamping of packets allows the server and clients to determine which packets are relevant in the event of out-of-order reception. As idSoftware realized, reliable data transfer in these types of games is mostly useless (of course, certain discontinuous data, such as increments in players' scores or the death of a player, must be reliably acknowledged) and the faster packet delivery obtained through UDP is far preferable to TCP's reliability guarantees.

2.2.3 Minimal Client-Side Game Logic

Quake also provides a minimal client-side proxy for the server-side game simulation. Each player is allowed to change his or her orientation (but not position or certain other characteristics) without having to wait for the server's next game state echo. This division of labor keeps the game running smoothly on every client's screen, without hiding any pertinent data from the master server.

2.2.4 Quantization

Quake reduces the size of network messages by quantizing the components of floating point numbers to increments of a quarter unit. This can cut the size of data updates in half, but makes measuring performance a little difficult, as discussed below.

2.2.5 Position Message Minimization

Quake's final optimization is to send updates only for the components of state vectors that have changed since the last broadcast. The bandwidth required for position updates is thus further reduced.

2.3 QuakeWorld — Successes and Failures

Despite Carmack's initial insight into network limitations that would affect Quake's gameplay, Quake's multiplayer performance was only mediocre. He has been quoted as follows:

> While I can remember and justify all of my decisions about networking from DOOM through Quake, the bottom line is that I was working with the wrong basic assumptions for doing a good Internet game. Unfortunately, 99% of the world gets on with a slip or ppp connection over a modem, often through a crappy overcrowded ISP. This gives 300+ ms latencies, minimum. Client. User's modem. ISP's modem. Server. ISP's modem. User's modem. Client. God, that sucks. Ok, I made a bad call. I have a T1 to my house, so I just wasn't familiar with PPP life. [2]

2.3.1 Client-Side Simulation

As opposed to Quake, in building QuakeWorld Carmack acknowledged the quality of the typical user's Internet connection: it "sucks". Most people who connect to the Internet do so through a relatively low-bandwidth, high-latency modem. In this context, waiting for the server's round-trip-time delayed response to each keystroke before updating a player's state on the local host, simply does not work. The solution to this problem is a tradeoff: by accepting the chance that the server might have a different view of the game world than the client, the client may *independently* simulate parts of the world in order to make the gaming experience smoother for players with high latency connections. That is, QuakeWorld will simulate the client character for several seconds past the point at which a dropped or late packet was supposed to have arrived. Once beyond this buffer zone, however, the game will indeed freeze, as it is likely that further independent simulation will throw the client's view of the world even further

from that of the server (which maintains the only "true" world state). Additionally, client A will not send packets to the server if it believes that the server and other clients can correctly predict A's future positions based on currently available data. These two techniques, known together as *dead reckoning*, serve to significantly reduce network traffic in massively multiplayer games.

2.3.2 Scoping Character Position Updates

Part of Quake's surface visibility preprocessing is to calculate which locations (and polygons) are visible from every location in the map (the complexities here are beyond the scope of this paper and have been omitted). This subspace of the world is called the *potentially visible set* [10] and is used to minimize overdraw on the screen as well as to level performance throughout the game world.

QuakeWorld calculates a slightly broader set of locations called the *potentially* **hearable** *set*. Using this PHS, the server scopes the character position updates that are sent to each client [3] based on each client's current location within the game world. If character A might possibly see character B within a certain upcoming time frame, then updates about character B are sent to character A's machine. The reason for using the potentially hearable set rather than the potentially visible set is to prevent characters from appearing out of nowhere in the case of dropped packets: if the initial packets that inform client A of character B's entry into A's potentially hearable set are dropped, it is likely that later updates will still arrive before character B is actually visible on the A's screen. Naturally, due to packet loss, character B may still be rendered in an incorrect position on A's screen; however, any Quake player will affirm that this tradeoff is preferable to not seeing B's approach at all.

3 Network Simulations

Even in the presence of all the above arguments for why QuakeWorld should perform better than Quake (over the imperfect Internet), for students of the Internet it is indeed educational to examine whether these arguments hold up in practice. Thus we desired to experiment with Quake and QW on several simulated internetworks, using DummyNet [7] as an analytical tool.

3.1 DummyNet

DummyNet is a FreeBSD kernel extension that simplifies the real-world acquisition of an application's performance under dynamic network conditions. First, DummyNet requires no changes to the applications being studied, an invaluable feature when evaluating non-open source applications (and a feature preferable to extra hacking even with open-source software).

Second, DummyNet provides reproducible results because the network traffic from the application being tested need be routed only through the DummyNet code (and not through the random and uncontrollable Internet itself) in order to simulate any desired network conditions. Finally, DummyNet allows the user to vary the network topology and parameters in a marked variety of ways, broadening the usefulness of the tool.

3.2 Hypotheses

To test the underlying networking theory behind Carmack's improvements from Quake to QuakeWorld, we used DummyNet to test the following hypotheses regarding gameplay:
1) With high latencies (greater than 100 milliseconds), the control of a Quake character will be practically unresponsive, as the client will receive server updates too long a time (i.e. > 100 ms) after the keyboard control to move the character is activated.
2) With high packet loss, the control of a Quake character will be discontinuous as the client's display "jumps around" to match the intermittent character state information received from the server.
3) With high packet loss, the control of a QuakeWorld character will be continuous (but not completely accurate) despite the dropped update packets from the server.

4 Results

4.1 Simulations Done

We performed several network simulations using Quake and QuakeWorld, both modified to capture and print out network and rendering data during the normal progression of the game. We constructed a makeshift network through a DummyNet terminal using several Pentium Pro/Windows NT4 workstations. With this generic topology, a variety of network conditions were tested.

The Quake server code was modified to print out character world-positions upon constructing each packet to send to the clients. Similarly, we modified the client code to print out identical information upon decoding packets received from the server. It is assumed that because the newly decoded values on the client are incorporated into the very next displayed graphics frame that this is an appropriate time to also record a timestamp in the client's printout.

QuakeWorld, on the other hand, had a slightly different modification. QW records the positions of characters within the game world after the client-side *prediction* of the character's future path has been done, and before each frame is displayed by the graphics routines.

The client and server run on separate clocks (mainly because they are typically run on separate machines). The clock for each game starts at zero based on the start time of the process running the game. Thus, since the client and server have very different ideas of game time (because the client and server processes have been running for some arbitrary period of time before connecting to each other), they will record different timestamps for their views of character position vs. time, even though the position data between the two are in direct correspondence (because they share the same game world). After gathering the data, therefore, the correspondences are found by normalizing the timestamps against the start time of the game on each process. This is a reasonable approach: we assume that hardware clocks run at roughly the same speed over periods of a couple of minutes.

The aforementioned experiments were performed by increasing network delay and the percentage of packets dropped (all straightforward tasks for DummyNet), and then comparing the client's and server's views of character positions within the game world.

4.2 Delay

Delay was varied over a range of times, and the visible results were markedly different for Quake and QuakeWorld. In the case of Quake, the screen character's responses to movement commands were delayed by approximately the length of DummyNet's imposed delay. This, of course, occurs because the client runs in lockstep with the server. QuakeWorld's characters, on the other hand, moved immediately when the command to move was given because of QW's support for client-side prediction of the game world, independent of immediate feedback from the server. QW was thus **far** more comfortable to play and presented a much cleaner virtual reality than did Quake under the same delay conditions. Still, QuakeWorld's characters moved somewhat more jumpily than they did when there was no delay, due to quick movements unaccounted for in the local host's extrapolations.

4.3 Dropped Packets

Data was recorded in Quake and QuakeWorld using two DummyNet setups: one with no dropped packets (beyond that introduced by the local LAN), and one with 50% dropped packets. Character position data was captured and analyzed.

Clearly, running on a local LAN with no added packet dropping, we see a smooth game in both Quake and QuakeWorld. The interesting data comes when we start dropping packets artificially through DummyNet to simulate network congestion.

In Quake, a graph of the samples of the character position as the client received them vs. the samples of the character position as the server sent them shows an obvious consequence of 50% dropped packets: less than half of the samples of the character position function get through to the client. Unfortunately, Quake leaves the data at that and simply displays the character at the last position received from the server — hence the extremely jumpy gameplay we observed.

However, QuakeWorld performs far better. Even though it receives only 50% of the character position samples sent from the server, the position function that the client uses to render the character on the screen is much smoother. The only notable imperfections occur when the game character makes sharp movements: since client A's linear extrapolation of character A's position diverges from the server's calculations based on actual keystrokes, when a server update packet finally reaches client A, character A's extrapolated position is changed to its "correct" position in the course of one rendering frame.

4.4 Other Issues

Although not often considered, the time it takes to put a frame of graphics on screen and the time between when a control is pressed and when the computer processes it both often appear to be network latency and should be considered in the design of any pseudo-realtime distributed interactive application. In addition, it is evident that the Quake server does not send updates as evenly spaced as they could be, and some smoothing here could keep network buffers from overflowing and help to keep the client and server in "voluntary" lockstep.

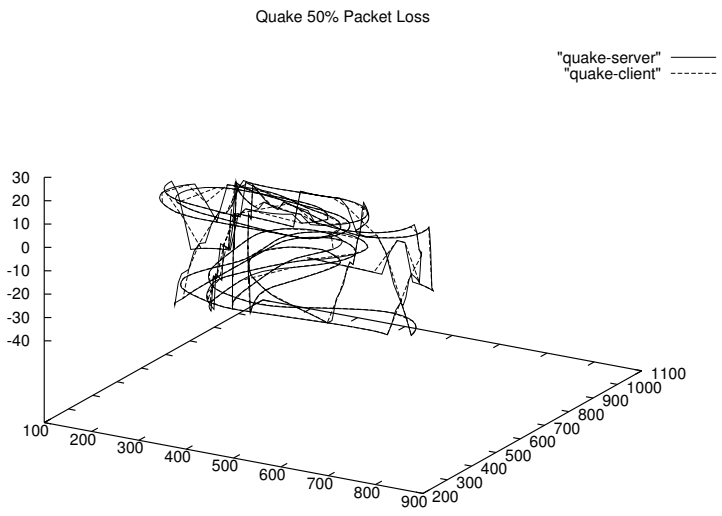

Figure 1: This plot of the client's 3-D game position over time demonstrates a sometimes marked disagreement between server and client

4.5 Further Optimizations

4.5.1 Further Client-Side Simulations

Internet play can be further improved by using smarter client-side proxies for the server-simulated game objects. Unreal Tournament [9], by Epic Games, imbeds this concept into their game scripting language, allowing the game player to control their character in the face of a total blackout of packets from the server for several seconds. This results in crystal-clear gameplay, as long as the client's simulation of the game world corresponds to the server's simulation. Due to improved extrapolation techniques in Unreal, this desired correspondence is often a reality.

4.5.2 Multicast Groups

Network traffic can be minimized by exploiting multicast groups [5] on the Internet. Rather than having the server send character position updates to each of the clients separately, it can send the updates to a multicast channel to which all the player clients are subscribed. The multicasting could be done with a source specific tree initially based at the server, significantly lowering the amount of bandwidth required at the server. One could also envision creating multicast groups based on certain network- or game-specific properties. For example, multicast groups between teams or "clans" could be created, allowing for efficient planning and conversation during the game.

5 Conclusions

Networked applications such as games will be able to make use of increased bandwidth of networks in the future to distribute rich content. However, due to physical transmission limits for EM signals, latency will always be somewhat of a problem. A constantly growing set of established techniques are available to help battle the effects of latency, and continued work in this area will make an amazing array of networked applications possible.

6 Acknowledgements

We thank the University of Washington for funding this research project, and we are indebted to idSoftware for its continuing work on distributed interactive simulations, providing us with hours and hours of enjoyable research.

References

[1] Netrek Continuum. (http://www.netrek.org).

[2] John 'JCal' Callaham for GAMESPY. The History of QuakeWorld (http://www.gamespy.com/articles/quakeworld_a.shtm).

[3] Thomas A. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 85–92. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[4] Peter Lincroft. The Internet Sucks: Or, What I Learned Coding X-Wing vs. Tie Fighter (http://www.gamasutra.com/features/19990903/lincroft_01.htm).

[5] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Donald P. Brutzman, and Paul T. Barham. Exploiting Reality with Multicast Groups. *IEEE Computer Graphics and Applications*, 15(5):38–45, September 1995.

[6] D.C. Miller and J.A. Thorpe. Simnet: The Advent of Simulator Networking. In *Proceedings of the IEEE*, pages 1114–1123, August 1995.

[7] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. Technical report, Dipartimento di Ingegneria dell'Informazione, Universita di Pisa, Italy, 1997.

[8] Ravi Swamy. idSoftware Releases Quake 1 Source Code Under the GPL (http://linuxtoday.com/stories/14111.html).

[9] Tim Sweeney. Unreal Networking Architecture (http://unreal.epicgames.com/network.htm).

[10] Seth J. Teller and Carlo H. Séquin. Visibility Preprocessing for Interactive Walkthroughs. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 61–69, July 1991.