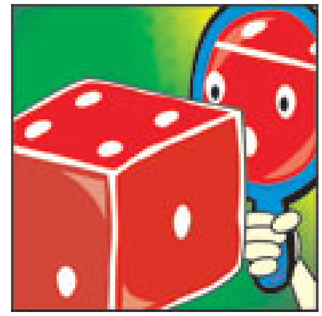
Crossroads
The ACM Student Magazine



Game-State Fidelity Across Distributed Interactive Games

by **_Aaron McCoy_**, **_Declan Delaney_**, and **_TomasWard_**

Introduction

Distributed interactive games offer players a three dimensionalvirtual world experience. Within this virtual world, players interactwith each other and with their environment in real-time. They experiencethe same events, but from different viewpoints.

As interactive games have evolved, they have driven thetechnologies underlying them. The distinguishing feature of anydistributed interactive game is the network: the medium bywhich information is exchanged and shared between participants. Thenetwork impacts not only the design and development of distributed interactive games, but also their potential entertainment value.

In this article, four of the fundamental networking issues and theireffect on the design and operation of distributed interactive gameswill be discussed. In addition, a description of the differentcommunication architectures used in distributed interactive games willbe provided. Finally, as an illustrative example, these issues will berelated to

Unreal Tournament, a popular distributed interactive game.

The Dawn Of A New Age

In the not too distant past, distributed real-time virtualenvironments were science fiction; novels envisioned a world without rules or boundaries, a so-called'cyberspace', a place where the only limitations were that of thehuman mind [6]. A significant milestone inthe history of distributed virtual environments was the Multi UserDungeon (MUD), completed by Rob Trubshaw and RichardBartle at Essex University, in 1978 [4]. MUD allowed people all over the world to interact with each other,share environments, and implement their own environments. More recently,technological advances in processing power and graphicscapabilities, coupled with the widespread availability of theInternet, has led to the diffusion of distributed virtualenvironments within the public domain.

Networked, multi-player games drew mainstream attention with therelease of Doom in 1993 [8][10]. Doom enabled players to compete with each other in bothfour-player local area network (LAN) games and direct head-to-headmodem games. In 1996, the developers of Doom released Quake [8][10], another milestone in thehistory of networked multi-player games. Quake was a pioneer of theclient-server architecture widely used in modern onlinegames; it introduced players to a larger type of interactiveenvironment than previously available. The popularitynetworked multi-player games were receiving prompted the developmentof more complex distributed games [5][9]. There are now numerous online, dedicated gamingservices using high-bandwidth connections that provide server hostingfor the most popular games [5].

In this article, we will investigate the networking featuresthat must be considered by distributed games developers. We willstart by briefly examining the potential network architectures andthen investigate four issues that affect the maintenance ofgame-state fidelity across all participants in a distributedapplication. We will then investigate these issues as they apply toone of the most popular distributed games, Unreal Tournament (UT)[3].

Speak *Friend* And Enter

Game-state fidelity is a measure of the consistency ofthe game-state among all participants within the game. In thiscontext, the game-state refers to a description of the sharedenvironment and all dynamic objects within this environment at aninstant of time. Distributed interactive games require as muchinformation as possible to be exchanged and processed in real-timein order to maintain a reasonably consistent game-state. One of theissues for game developers to consider is how they want thisinformation to be communicated between participants in the game.There are two main communication architectures in use today:peer-to-peer architectures (**Figure 1**) and client-server architectures (**Figure 2**).

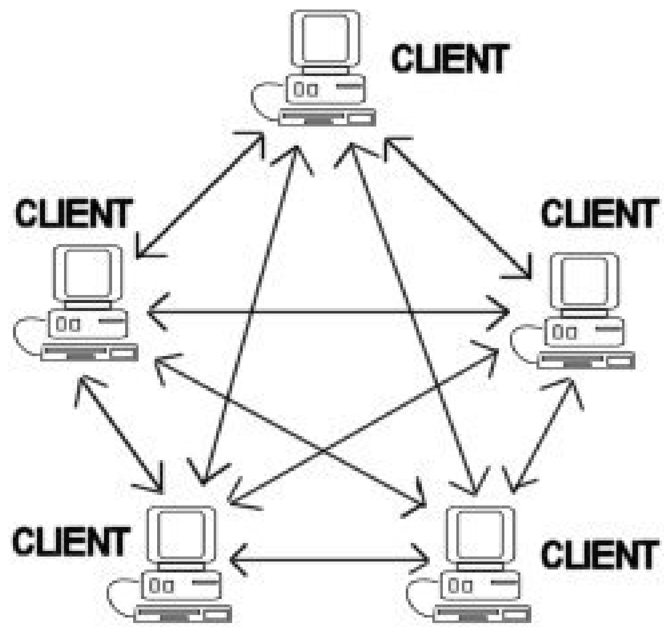


Figure 1: Peer-to-peer Networking Architecture.

With peer-to-peer architectures, communication is sent directlybetween participants in the game. Players wishing to update otherplayers about their activity, must send a message to all of theother players so that they can then update their game-stateinformation, and hence maintain global game-state fidelity.

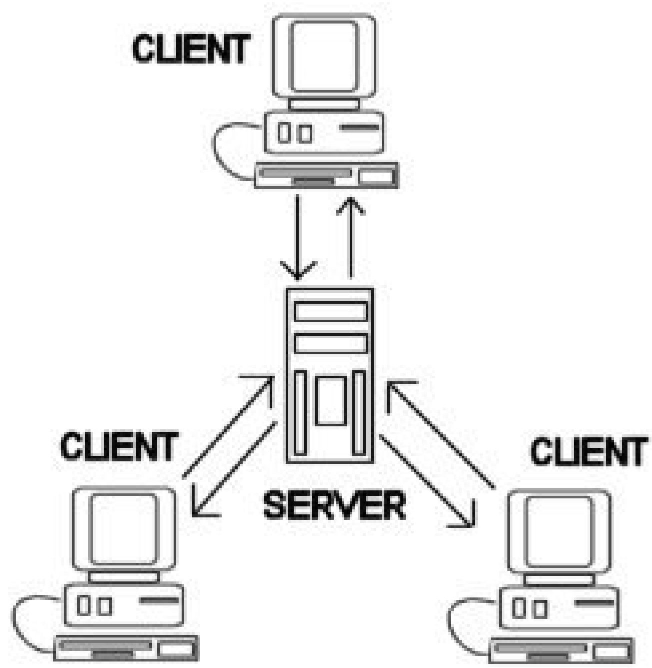


Figure 2: Client-Server Networking Architecture.

With client-server architectures, one machine is designated asthe server and is responsible for maintaining the overallgame-state and making the game-play decisions. In this respect, theserver determines the game-state. The rest of the machines aredesignated as clients, and they are responsible for rendering aview of the shared virtual world to players and for updating theserver with details of any actions that the players are performing.If a player wishes to update other players about his or her activity,the player sends a message to the server advising of the changes,and then the server distributes the information to all of the otherclients. Communication is never sent directly between clients, italways passes through the server.

Peer-to-peer architectures offer theoretically better scalability [**12**], but in general, overall game-state fidelityand player interactions are harder to keep consistent because no oneplayer has complete authority over the game-state. In contrast,client-server architectures offer less scalability, but theyimplicitly provide the ability to maintain accurate game-statefidelity and manage player interactions.

Regardless of the architecture chosen, the network itself raisesa number of technical issues that are critical to distributedapplications. We will now focus our interest on these.

The Four Horsemen Of The Apocalypse

The Internet poses a wealth of challenges for games developers.In particular, four issues affect datatransfer over any network, regardless of the network architecture[15].

Network Latency

Network latency is a measure of the time it takes for a packetof information to travel from one computer to another across anetwork.

Network latency arises for a number of reasons. First, a lowerlimit is imposed by the finite speed of light, which results indata traveling at about two thirds the speed of light in a vacuumthrough a fiber optic cable [2]. Second,the endpoint computers introduce delays when they process the data[1]. Finally, the network introduces delays asthe data propagates through network routers before reaching itsdestination.

Network latency represents one of the greatest challenges to thedevelopment of distributed interactive games. As the networklatency increases, maintaining game-state fidelity between the gameparticipants becomes more difficult, with each player's view of theshared virtual world becoming increasingly different depending onhow up-to-date their information is. Network latency means thatgame developers must assume that all information received byparticipants is already out-of-date when it arrives.

Network Bandwidth

Network bandwidth is the rate at which a network can deliverdata from a source point to a destination point.

The type of channel used to transport data determines theavailable bandwidth, and it is also limited by the hardware used totransmit the data [16].

The available bandwidth limits the amount of information thatcan be shared and exchanged between participants per unit time. Ifa player is connected through a low bandwidth line, which is oftenthe case for a home connection, they will not be able to receivecomplete information relating to every other participant in thevirtual world. As a result, it is up to the game developers toallocate the available bandwidth to networked players. Given thesebandwidth limitations, it is the main goal of a distributedinteractive game to enable the communication of sufficientgame-state information to enable players to determine events withina reasonable level of accuracy.

Network Reliability

Network reliability is a measure of how much data is lost by thenetwork during the journey from source to destination host.

Network data loss occurs for two main reasons [**15**]. First, data can be lost as it travels along thenetwork transmission channels. This is the most obvious but alsoleast frequent cause of data loss. The most frequent cause of dataloss is due to the network routers that transfer the data. If anetwork router receives too much data for it to handle, it willdiscard all incoming packets that arrive while the queue is full.This policy is known as drop-tail (DT) [**13**].

In distributed games development, network reliability does notpose as much of a problem as one might first assume, provided thatthe rate of data loss throughout the game remains low. The reasonfor this is because dynamically changing information within thegame is usually being updated at a very fast rate amongparticipants, so any data that is lost is usually replaced quitequickly.

Network Protocol

A **network protocol** describes the set of 'rules' that twoapplications use to communicate with each other.

A network protocol consists of three components. First, the**packet format description** allows the endpoints of thecommunication channel to identify the various individual parts ofdata that are contained within the information stream. Second, the**packet semantics description** allows the communication endpoints tounderstand the various individual parts of data. Finally, the**error-handling description** governs how the communication endpointsshould respond to various error scenarios that may occur duringdata transmission.

The basic protocol for Internet transmission is the InternetProtocol (IP) [**7**]. However, applications almost never use IP directly. Instead, they use one ofthe higher-layer transport protocols that are built on top of IP.The two most common transport protocols used in distributedinteractive games are the Transmission Control Protocol (TCP) andUser Datagram Protocol (UDP) [**16**].

TCP offers reliable, connection-oriented, stream-based transportof data. It guarantees the delivery of all data in the correctorder by using a system of positive acknowledgement withre-transmission. It can only send data between two directlycommunicating hosts.

In contrast, UDP offers unreliable, connectionless, packet-basedtransport of data. Data can be sent from a single host to anynumber of different hosts without having to set up individualconnections, by using either IP broadcasting or multicasting.

Distributed interactive games are real-time systems, and it isthis need for real-time information exchange and processing thatusually influences the choice of communication protocol. Thereliability and ordering guarantees provided by TCP introduce extraoverhead. These guarantees are not necessarily required as it isoften more important for data to be delivered quickly than it isfor it to be delivered reliably. With UDP, distributed interactivegames can send out data as soon as it is generated without havingto wait to make sure the data is ordered and without having tosubsequently wait for an acknowledgement to ensure that the datawas delivered. In addition, the ability to broadcast and/ormulticast UDP data packets to multiple sources greatly aids theability for information to be distributed quickly andefficiently.

Ultimately, the choice of protocol depends on the specificrequirements of the distributed interactive game. Recently,developers have been using multiple protocols together to providedifferent levels of service for data transport, with the choice ofprotocol being determined by the measure of how crucial the data isin maintaining overall game-state fidelity.

So, Just Who Is *In Charge*?

Distributed interactive games target real-time interactivitybetween participants and between participants and theirenvironment. This interactivity highlights an interesting problem:who determines what events happen, how the events occur, and whatthe final outcomes of those events are? This is directly related tothe problem of maintaining game-state fidelity betweenparticipants. Due to out-of-date or incorrect game-stateinformation, it is entirely possible that one or more players willincorrectly conclude that an interaction of some sort took place,when in fact it may not have. This can lead to participantsdisagreeing about whether an interaction actually occurred.Furthermore, even if all the participants agree that an interactiondid take place, they may not all agree about the specific detailsof the interaction.

The distributed interactive game must manage these interactionsand provide accurate detection and resolution of collisions amongparticipants. Examples of such collisions may include directplayer-to-player contact, player-to-environment contact, or perhapsweapon fire/damage. Usually a client-server architecture isadopted, so that the server is *in charge*, and it alonedetermines the true game state. In classical peer-to-peerarchitectures, no one client can be considered *in charge* aseach client communicates with all other clients, so game-states aredetermined individually.

Let us now turn our attention to Unreal Tournament (UT) anddiscover how they have dealt with the four networking issues wehave described previously.

Totally *Unreal*

Unreal Tournament's design architecture essentially consists oftwo parts [**18**]. First, there is theunderlying game engine itself. This engine provides most of thegame mechanics and the graphics capabilities. It also provides thegeneralized network code. Secondly, in Unreal Tournament "the 'gamestate' is self-described by an extensible, object-orientedscripting language which fully decouples the game logic from thenetwork code. The network code is generalized in such a way that itcan coordinate any game which can be described by the language" [**18**]. This scripting language is known asUnrealScript, and it provides developers with a built-in, fullyobject-oriented language with which to program events intothe game. UnrealScript is based on a C++/ Java variant and containssimilarities to both languages. The power and usefulness ofUnrealScript lies in its implicit support for game-specificparadigms, such as concepts of time and state within the gameenvironment. Ninety percent of the code that governs game-statedynamics in UT was written in UnrealScript [**14**].

UT's Communication Architecture

UT uses a permissible-client-server architecture to maintaingame-state fidelity. This is a standard client-server architecturewith one notable exception: whenever a player wishes to perform anaction, he/she must first ask permission from the server. Clientscannot perform an action, such as firing a weapon, without gettingpermission from the server to do so (the one exception to this ismovement, which is predicted by each client - this is detailedfurther below). This ensures that no disagreements can arisebetween clients as to whether an interaction takes place or not.All events are, ultimately, determined by the server.

The server is completely authoritative over the flow of play,and in addition gameplay logic (code that evolves the game-state)should only be carried out on the server. This means that theserver's game-state can always be regarded as the only truegame-state, and the game-states that exist on client machines areapproximations to the server's state.

The main advantage of UT's permissible client-serverarchitecture is that it provides a means of minimizing the effectof game-state inconsistency among clients. This is because theserver is kept aware of all activities being performed.

The main disadvantage of this architecture is that it introducesextra lag, or delay, in the response time of the game to variousclient events. For instance, if a player fires a weapon, there willbe a noticeable delay between when the player presses the firebutton and when the weapon actually fires onscreen. This is becausethe client machine is waiting for permission from the server beforeit can fire the weapon. This lag is compounded by the fact that ifthe player is playing over a connection that has a high latencytime the noticeable delay between firing will be increased.

UT's Network Protocols

UT employs a network driver that is layered on top of the UDPprotocol. This network driver provides some of the services of TCPby handling point-to-point connections and positiveacknowledgements. Hence it replicates at the application layerTCP's mechanisms when reliability is requested, but always sendsUDP packets over the network. By utilizing both types of transportprotocol, UT gets the best of both worlds. It can send and receiveunreliable, packet-based data as well as being able to set up TCPconnections and send reliable, stream based data. Data that iscritical to maintaining game-state fidelity between participants inthe game is sent reliably, to ensure that it is delivered to thedestination. Data that is not so critical, and hence can reasonablyafford to be lost, is sent unreliably, with no guarantees ofreceipt at the intended destination.

UT and Network Latency

If UT used the permissible client-server architecture for everyaction that the player can take, then player movement would be veryslow and sluggish. For instance, if the player was playing on anetwork connection that had a 200ms round trip time between itselfand the server (100ms each way), then after the user pressed themovement key they would not see themselves move onscreen until200ms later. This would be extremely frustrating and would reducethe enjoyment of the experience for the participant.

To eliminate the above client-movement lag caused by networklatency, UT uses a form of client prediction or dead reckoning [17] that can best be described as a "lock-steppredictor/corrector algorithm " [18] (thishigh-level feature is actually implemented in the UnrealScriptlanguage rather then the engine's generalized network code). When aplayer performs a movement and requests permission from the serverto do so, the client machine actually predicts where the playerwill move to while it is waiting for permission from the server tomove. As a result, both the client and the server execute the samemove for the player. However, the server has the last say, so thatwhen it is finished executing the requested movement, it sends theresult back to the client. In the meantime the player is viewingthe client prediction on screen. If this position differs from theserver's player position, the client must correct the player'sposition. It does so by using convergence algorithms to smoothlyconverge to the true game-state position.

At any point in time, the UT client is predicting ahead of whatthe server has told it, by an amount of time equal to half it'sround-trip latency value. As a result the local player movementdoesn't appear to lag. The client movement prediction usuallymirrors the client movement determined by the server. Only in rarecases (such as when a player is getting hit by a weapon or bumpinginto another player) does the player's location need to becorrected by the client.

UT and Network Bandwidth

Bandwidth limitations impose restrictions on the amount ofinformation that can be transmitted between the server and theclient. In order for UT to allocate bandwidth resources efficientlyand effectively, it utilizes a load-balancing technique thatprioritizes actors. Actors are objects capable of interacting withany other objects in the game. Players, software opponents (a.k.a.BOTs), and movable environment objects are all actors. Each actoris assigned a network priority value, indicating how important itis for maintaining game-state fidelity between participants. Theavailable bandwidth is then allocated based on the ratio of networkpriorities. If, for example, an actor has a priority value that istwice the priority of another actor, it will get updated twice asoften. With this system, the most important information, such asplayer movements and weapons fire, will be updated more frequentlyand given higher bandwidth preference than information that is notvery important, such as world objects that have little or no effecton game-play.

In order to complement the above-mentioned load-balancingtechnique, UT also utilizes a technique known as relevant sets. Atany one time, a player will only ever be interacting with a smallsubset of all actors within a game environment. In addition toprioritizing each individual game actor based on how important theyare to game-play, UT also prioritizes actors based on how importantthey are to individual players. The server calculates the set ofactors that it deems are relevant to or capable of affecting eachclient and stores them in a relevant set. Using this system,clients will not receive redundant information from the serverabout actors that are of no consequence or relevance to it. This isa form of relevance filtering [15].

UT and Network Reliability

As already stated above, UT uses both the UDP transport protocoland a form of the TCP transport protocol provided for by it's ownnetwork driver. The decision as to which game-state informationshould be sent reliably and which information should be sentunreliably is left to the sole discretion of the game developersthrough the high-level UnrealScript scripting language. Byproviding both types of protocol, UT ensures that the influence ofnetwork reliability on the game-state fidelity is kept to aminimum. Critical information is sent reliably, ensuring that ifthe information is lost it will be re-transmitted.

Conclusion

This article has explored some of the more important networkingissues that relate to the development of distributed interactivegames, and it has provided descriptions of how they may affect theoperation of such games. For the interested reader, an additionalcase study is available detailing the development of a game calledX-Wing vs. Tie-fighter, and how the designers of that game copedwith the networking issues described above [11]. Developing distributed real-time games for useover the Internet is difficult. Bandwidth, latency and reliabilityvary tremendously in such a heterogeneous network. Game developershave absolutely no direct control over the limitations imposed bythe Internet. All they can do is react to these limitations andwork to provide the best software solution possible. It is atestament to their talent, creativity, and ingenuity that games suchas Unreal Tournament exist.

References

1

Blow, J. A look at latency in networked games. In *GameDeveloper Magazine.* July 1998.

2

Cheshire, S. *It's the Latency, Stupid.* <**http://rescomp.stanford.edu/~cheshire/rants/Latency.html**>.

3

Unreal Tournament is copyright (c) 1999 - 2002 Epic Games Inc, <**http://www.epicgames.com**>.

4

Friedl, M. *Online Game Interactivity Theory.* Charles River Media Inc, 2003.

5

Gamespy Arcade. <**http://www.gamespyarcade.com**>.

6

Gibson, W. *Neuromancer.* Ace Books, New York, 1985.

7

Gulati, S. *The Internet Protocol Part One: TheFoundations.* <**http://www.acm.org/crossroads/columns/connector/july2000.html**>.

8

Kushner, D. The Wizardry of id. *IEEE Spectrum* Vol.39No.8, August 2002.

9

Huebner, D. Industry Watch - The Buzz about the Game Biz. In*Game Developer Magazine.* June 2001.

10

Doom and Quake are copyright (c) id Software, Inc. <**http://www.idsoftware.com**>.

11

X-Wing vs. Tie-fighter is copyright (c) LucasArts EntertainmentCompany LLC. <**http://www.gamasutra.com/features/19990903/lincroft_01.htm**>.

12

Macedonia, M. R., Zyda, M. J., Pratt, D. R., Brutzmann, D. P.,and Barham, P. T. Exploiting reality with multicast groups. In*IEEE Computer Graphics and Applications* Vol.15 No.5,September 1995, pp. 38-45.

13

Pentikousis, K. *Active Queue Management.* <**http://www.acm.org/crossroads/columns/connector/july2001.html**>.

14

Reinhart, B. Epic Games Unreal Tournament. In *Game DeveloperMagazine.* May 2000.

15

Singhal, S., and Zyda, M. J. *Networked Virtual Environments:Design and Implementation.* ISBN 0-201-32557-8, Addison-Wesleyand ACM Press, 1999.

16

Stallings, W. *Data and Computer Communications.* FifthEdition, Prentice Hall, Upper Saddle River N.J., 1996.

17

Standard for Distributed Interactive Simulation - ApplicationProtocols. *IEEE* 1278.1, 1995.

18

The Unreal Technology Webpage. <**http://unreal.epicgames.com**>.

Biographies

Aaron McCoy (**amccoy@eeng.may.ie**) received his B.Sc. degree from the NationalUniversity of Ireland, Maynooth in 2002, specializing in computerscience and theoretical physics. His main areas of interest are distributed systems, networked virtual environments and the use ofartificial intelligence in interactive games.

Declan Delaney (**decland@cs.may.ie**)received his B.E. and M.Eng.Sc. degrees from University College Dublin, Ireland in 1991 and 1998 respectively. After working in industry for eight years he iscurrently pursuing a doctorate in network latency maskingtechniques.

Tomas Ward (**tomas.ward@eeng.may.ie**) received his B.E., M.Eng.Sc. and Ph.D degreesfrom University College Dublin, Ireland in 1994, 1996 and 1999 respectively. From 1999-2000 he taught at the Department of Computer Science at the National University of Ireland, Maynooth. Currently he teaches in the Department of Electronic Engineering at NUI Maynooth. His research interests lie in the areas ofhuman-computer interaction and distributed interactive systems.